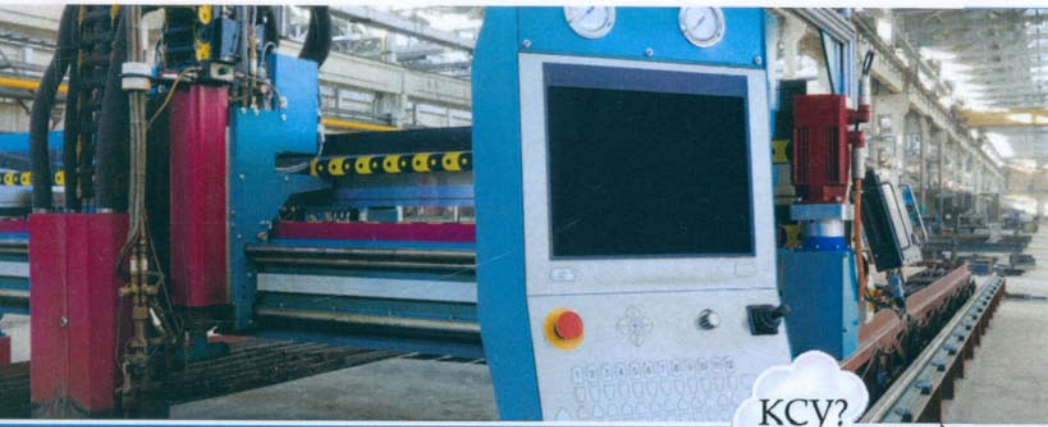
С. М. Москвіна, Т. В. Грищук

Комп'ютерні технології та програмування

Елементи професійного програмування



Частина 4

Міністерство освіти і науки України
Вінницький національний технічний університет

С. М. Москвіна, Т. В. Грищук

КОМП'ЮТЕРНІ ТЕХНОЛОГІЇ ТА ПРОГРАМУВАННЯ
Частина 4
ЕЛЕМЕНТИ ПРОФЕСІЙНОГО ПРОГРАМУВАННЯ

Навчальний посібник

Вінниця
ВНТУ
2014

УДК 681.3.06(075.8)
ББК 32.973я73
М82

Рекомендовано до друку Вченою радою Вінницького національного технічного університету Міністерства освіти і науки України (протокол № 10 від 23.05.2013 р.)

Рецензенти:
В. М. Лисогор, доктор технічних наук, професор
В. А. Лужецький, доктор технічних наук, професор
О. М. Ткаченко, кандидат технічних наук, доцент

Москвіна, С. М.

М82 Комп'ютерні технології та програмування. Частина 4. Елементи професійного програмування : навчальний посібник / С. М. Москвіна, Т. В. Грищук – Вінниця : ВНТУ, 2014. – 195 с.

В начальному посібнику розглянуто основи розробки професійних програмних продуктів на мові С. Весь матеріал викладено у формі прикладів, що супроводжуються якісними зрозумілими коментарями. Призначений для самостійної роботи студентів денної форми навчання напряму підготовки 6.050202 - "Автоматизація та комп'ютерно-інтегровані технології".

УДК 681.3.06(075.8)
ББК 32.973я73

ЗМІСТ

ПЕРЕДМОВА

Мова програмування С, елементи професійної роботи з якою покладені в основу даного навчального посібника, створена у 1972 році працівником фірми Bell Laboratories Денісом Рітчі (Dennis V. Ritchie) при розробці операційної системи UNIX. Мова проектувалася як робочий інструмент для системного програмування з орієнтацією на розробку добре структурованих програм. Вдале поєднання лаконічності конструкцій та багатство можливостей дозволило мові С швидко розповсюдитися та стати найбільш популярною мовою прикладного та системного програмування. Так, компілятори мови С працюють майже на всіх типах сучасних ЕОМ в операційних системах UNIX, Linux, Mac OS, OS/2, Windows, Solaris тощо.

Навчальний посібник призначений для студентів напряму підготовки 6.050202 – «Автоматизація та комп'ютерно-інтегровані технології», що вивчають дисципліну «Комп'ютерні технології та програмування» в другому та четвертому навчальному триместрах.

Для початку роботи з посібником студенти повинні знати:

- основи алгоритмізації і програмування задач;
- методи та алгоритми розв'язування типових задач на числах, масивах, матрицях;
- основи структурного програмування;
- синтаксис і семантику мови програмування С.

Головний шлях до оволодіння майстерністю програмування будь-якою мовою програмування – це виконання самостійних практичних завдань. Даний навчальний посібник призначений допомогти студенту на простих прикладах вдосконалити свої навички в розробці професійних програмних продуктів. Приклади задач програмування, викладені в даному посібнику, ілюструють головні аспекти виконання курсової роботи з дисципліни «Комп'ютерні технології та програмування».

В посібнику основна увага приділяється особливостям розробки графічного інтерфейсу користувача, особливостям обробки файлів даних великих розмірів та роботі з графікою засобами структурного програмування. Сучасні технології візуального програмування, представлені такими програмними продуктами, як, наприклад, Microsoft Visual Studio, дозволяють розв'язувати дані задачі легко та просто, не змушуючи програміста витрачати зайві сили. Але основна **мета посібника** – це навчання принципам структурного програмування для самостійного вдосконалення навичок програмування. Саме тому студентам пропонується виконувати свої програмні розробки на високому рівні деталізації, не вдаючись до засобів візуального програмування.

Посібник складається з восьми розділів. В першому розділі викладено основні принципи роботи з графікою мовою програмування С. На простих прикладах студенти можуть не тільки навчитись рисувати графічні приміт-

тиви, а також вдосконалити свої навички з використання функцій стандартних бібліотек середовища програмування Turbo C++ та в розробці власних функцій для рисування графічних об'єктів.

Другий розділ посібника містить детальні інструкції зі створення авторської заставки до програмного продукту з елементами анімації.

В третьому розділі посібника розглянуто принципи проектування та програмування багатоієрархічного меню користувача в текстовому режимі. Наведено приклади створення головного меню, висхідного меню, діалогових вікон для внесення інформації користувача, вікон-повідомлень та основних елементів управління.

В четвертому розділі розглядається подання інформації користувачу у вигляді графіка функції. Показано, як в графічному режимі отримати графік аналітично заданої функції і таблицю її значень в окремих вікнах інтерфейсу користувача.

В п'ятому та шостому розділах посібника розповідається про основи роботи з текстовими даними. Детально розглянуто питання роботи з файловими потоками даних та питання обробки текстової інформації з використанням функцій стандартної бібліотеки string.h.

В сьомому розділі викладено основи розробки професійних програм для роботи з базами даних. Робота з базами даних ґрунтується на вмінні студентів використовувати структуровані типи даних мови С, вмінні працювати з файловими потоками даних і володінні навичками обробки та подання текстової інформації.

В останньому (восьмому) розділі посібника детально розглянуто задачу організації роботи з базою даних через графічний інтерфейс користувача. Керуючись поясненнями та коментарями, студенти зможуть легко оволодіти технологією розробки файл-серверних програмних додатків.

В кінці кожного розділу наведено питання для самоперевірки та практичні завдання, що допоможуть більш ґрунтовно засвоїти матеріал.

Посібник завершується словником термінів, до кожного з яких подається переклад англійською мовою та коротке тлумачення. Вивчення наведених термінів корисно для поглиблення теоретичних знань з програмування та інформатики.

Автори посібника бажають усім студентам натхнення при вивченні дисципліни «Комп'ютерні технології та програмування» і сподіваються, що ця книга полегшить виконання завдань для самостійної роботи.

1 ОСНОВНІ ПРИНЦИПИ РОБОТИ З ГРАФІКОЮ МОВОЮ ПРОГРАМУВАННЯ С

Сучасні програмні продукти, що використовуються як в промисловості, так і в навчальних комплексах або в професійних тренажерах, характеризуються розвинутим інтерфейсом користувача та привабливими заставками і різнокольоровими рисунками на екрані. Основні принципи роботи з графікою не залежать від компіляторів та середовищ програмування. Тому даний розділ орієнтований як на викладення синтаксису, семантики основних конструкцій програм для роботи з графікою, так і на їх практичне використання при розв'язанні типових задач.

1.1 Загальні відомості

Заголовковий файл graphics.h

Бібліотека **graphics.h** містить набір функцій для забезпечення повного контролю над графічними режимами різних адаптерів дисплеїв: **CGA, VGAHI, EGAHI, MCGA, Hercomohi, PC 3270, ATT400HI** і **IBM 8514HI**. Бібліотека містить більше 50 функцій: як базових (креслення ліній, кіл і т. д.) так і розширюють можливості базових (креслення багатокутників, заповнення фігур, виведення тексту тощо).

Щоб запустити на виконання програму, в якій використовуються функції бібліотеки **graphics.h**, необхідно:

1. Щоб в робочому каталозі знаходились відповідні графічні драйвери (файли з розширенням **.BGI**);

2. А якщо програма використовує штрихові шрифти, то необхідно, щоб в робочому каталозі знаходились файли шрифтів (файли з розширенням **.CHR**);

3. Крім того, в налаштуваннях середовища програмування (компілятори) потрібно задати шлях до файлу **graphics.h**.

Файли BGI і вміст файлу graphics.h

Файл **BGI** – це файл, що містить графічний інтерфейс (Borland Graphic Interface) фірми Borland. Він забезпечує взаємодію програм з графічними пристроями. До початку роботи програми в графічному режимі дисплея функція `initgraph()` визначає тип адаптера, наявного в ПЕОМ, і завантажує в пам'ять відповідний **BGI**-драйвер, в якому визначені можливі режими роботи.

Функція `closegraph()` вивантажує графічний драйвер з пам'яті і відновлює текстовий режим роботи відеоадаптера. В бібліотеці **graphics.h** є функції, що дозволяють виходити з графічного режиму без вивантаження драйвера (`restorecrtmode()`) і повертатися назад (`setgraphmode()`).

Якщо в складі ПЕОМ є два монітори, то при визначенні графічних ада-

птерів автоматично вмикається графічний режим на тому пристрої, який дозволяє одержати найвищу якість зображення.

В робочому каталозі можуть знаходитися такі файли:

cga.bgi	драйвер для IBMCBA, MCGA;
egavga.bgi	драйвер для IBM EGA, VGA;
gerc.bgi	драйвер для Hercules;
att.bgi	драйвер для ATIT6300\400 рядків;
pc32070.bgi	драйвер для IBM 3270 PC;
ibm8514.bgi	драйвер для IBM 8514.

Вказаний вище набір файлів потрібно мати при складанні програм, які будуть працювати практично на всіх ПЕОМ, сумісних з платформою **IBM**. Якщо розробка програмного забезпечення виконується для заданого типу графічного адаптера, то достатньо мати один файл цього адаптера.

Всі процедури і функції бібліотеки **graphics.h** можна розбити на 7 функціональних груп:

1. Керування графічними режимами та їх аналіз:
`detectgraph()`, `initgraph()`, `closegraph()`, `graphdefaults()`, `cleardevice()`, `installuserdrive()`, `restorecrtmode()`, `setgraphmode()`, `setwritemode()`, `getgraphmode()`, `getmoderange()`, `getmaxmode()`, `getmodename()`, `getdrivername()`, `graphresult()`, `grapherrormsg()`;
2. Креслення графічних примітивів і фігур:
   – управління «поточним вказівником»: `moveto()`, `moverel()`, `getmaxx()`, `getmaxy()`, `getx()`, `gety()`;
   – рисування графічних примітивів: `line()`, `lineto()`, `linerel()`, `arc()`, `getarccoords()`, `circle()`, `sector()`, `ellipse()`, `rectange()`, `drawpoly()`;
   – управління стилем ліній і коефіцієнтом стиснення зображення: `sellinestyle()`, `getlinesettings()`, `setaspectratio()`, `getaspectratio()`;
   – управління кольором і шаблоном заповнення: `setcolor()`, `getcolor()`, `setbkcolor()`, `getbkcolor()`, `getmaxcolor()`, `getpalette()`, `getpalettesize()`, `getdefaultpalette()`, `setpalette()`, `setallpalette()`, `setrgbpalette()`, `setfillstyle()`, `setfillpattern()`, `getfillpattern()`, `getfillsettings()`, `fillpoly()`, `fillellipse()`, `floodFill()`, `pieslice()`, `bar()`, `bar3d()`.
3. Виконання бітових операцій: `putpixel()`, `getpixel()`, `imagesize()`, `getimage()`, `putimage()`.
4. Управління відеосторінками: `setactivepage()`, `setvisualpage()`;
5. Управління графічними вікнами: `setviewport()`, `getviewsettings()`, `clearviewport()`.

6. Управління виведенням тексту: `installuserfont()`, `outtext()`, `outtextxy()`, `settextstyle()`, `settextjustity()`, `setusercharsize()`, `gettextsettings()`, `textheight()`, `textwidth()`.

Функція ініціалізації:

`initgraph(int gdriver[тип адаптера], int gmode[режим графіки], char[30] driverpath[шлях до драйвера ])`.

В бібліотеці **graphics.h** визначені сталі для задання виду графічного адаптера `gdriver` перед викликом функції `initgraph()`:

DETECT=0	автовизначення;
CGA=1	адаптер **CGA**;
MCGA=2	адаптер **MCGA**;
EGA=3	адаптер **EGA** на 256 К;
EGA64=4	адаптер **EGA** на 64 К;
EGAMONO=5	**EGA** з монодисплеєм;
IBM8514=6	адаптер 8514;
Herc Mono=7	адаптер **Hercules**;
ATT400=8	адаптер для ПОЕМ AT8T;
VGA=9	адаптер **VGA**;
PC3270=10	адаптер 3270;
Current Driver=-128	для **GET ModeRANGE**.

Якщо параметр `gdriver=detect`, то система вмикається в режим автовизначення. Якщо параметр `gdriver` містить номер конкретного адаптера, то і другий параметр `gmode` повинен приймати значення автоматично (номер режиму, дозволеного при даному адаптері). Якщо параметр `driverpath =""`, то драйвери повинні знаходитися в поточному каталозі (альтернативно можна вказати шлях до драйверів, наприклад, **C:\Study\TC\BGI**).

Лістинг 1.1. Приклад виклику функції ініціалізації, що використовується найчастіше

```
#include <graphics.h>
int main () {
  int gdriver, gmode; // gdriver – для графічного драйвера,
                      //gmode – для графічного режиму
  gdriver=DETECT; // режим автовизначення
  initgraph (&gdriver,&gmode,""); // ініціалізація графіки
  closegraph (); // вивантаження графічного драйвера з
                 //пам'яті
  return 0;
}
```

Лістинг 1.2. Процедура ініціалізації графічного режиму з урахуванням обробки помилок ініціалізації

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#include<graphics.h>
 void main()
 {
  int gdriver=DETECT, gmode, errorcode;
  initgraph(&gdriver,&gmode,"");
  errorcode=graphresult(); // результат ініціалізації
  /* якщо при ініціалізації виникла помилка */
  if (errorcode!=grOk)
    { printf("Помилка графіки:
             %s\n",grapherrormsg(errorcode));
      printf("Натисніть будь-яку клавішу:");
      getch();
      exit(1);
    }
 line(0,0,getmaxx(),getmaxy());// робота з графікою (рисування
                                //лінії)
 getch();  // затримка екрана (до натиснення будь-якої
          //клавіші)
 closegraph();
 return 0;
 }
```

Якщо у робочому каталозі не знайдено відповідних графічних драйверів (файлів з розширенням .BGI), то на екрані з'явиться таке вікно:

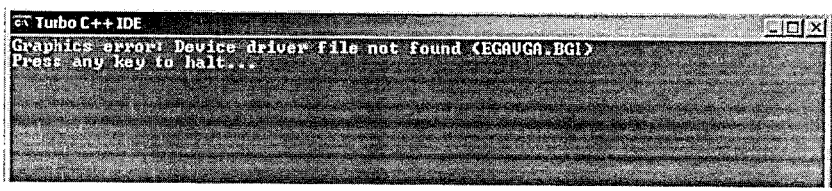


Рисунок 1.1 – Зображення вікна при помилці

На рисунку 1.2 показано вікно, що з'явиться на екрані при успішному виконанні програми.

Очищення графічного режиму виконується функцією `cleardevice()`. Дана функція очищає графічний екран і встановлює вказівник позиції в положення (0,0). В прикладі 1.3 показана організація переходу з графічного режиму в текстовий і в зворотному напрямку (зручно використовувати в складних прикладних програмах, що працюють в цих режимах).

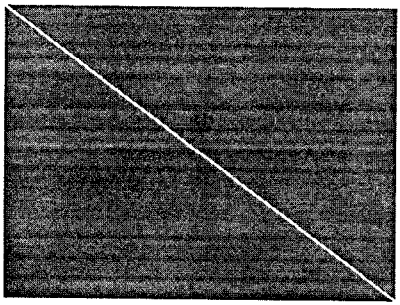


Рисунок 1.2 – Зображення вікна при успішному виконанні програми

Лістинг 1.3. Перехід з графічного режиму в текстовий

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <graphics.h>

int main () {
    int gdriver=DETECT, gmode, errorcode;
    int x,y;
    clrscr ();
    initgraph(&gdriver, &gmode, "");
      errorcode=graphresult();
    if (errorcode!=grOk) {
    printf ("Graphics error: %s\n",
    grapherrormsg(errorcode));
    printf ("Press any key to halt...");
    getch ();
    exit (1);
    }

    x=getmaxx ()/2;
    y=getmaxy ()/2;

// виведення повідомлення на екран
    settextjustify (CENTER_TEXT, CENTER_TEXT);
    outtextxy (x,y, "Press any key for exit graphic's
mode...");
    getch ();
    restorecrtmode (); // перехід в текстовий режим

// виведення повідомлення
    printf ("TEXT MODE\n");
    printf ("Press any key for return graphic's mode...");
    getch ();
```

```
    // повернення в графічний режим
        setgraphmode(getgraphmode());

    // виведення повідомлення
        settextjustify (CENTER_TEXT, CENTER_TEXT);
        outtextxy (x,y,"GRAPHICS MODE");
        outtextxy (x,y+textheight ("W"), "Press any key for
exit...");
        getch ();
        closegraph ();
        return 0;
    }
```

Як видно з прикладу, перехід з графічного режиму в текстовий за допомогою функції `restorecrtmode()` відбувається достатньо просто. Зворотнє ввімкнення графічного режиму здійснюється за допомогою функції `getgraphmode()`, яка повертає номер поточного графічного режиму. При роботі функції `restorecrtmode()` вивантаження графічного драйвера не відбувається, оскільки він залишається в пам'яті активним. Це і є основна перевага функції `restorecrtmode()`.

1.2 Функції рисування графічних примітивів

Лінії і точки:

– функція виведення лінії (відрізка) на екран (в поточному кольорі і стилі) `line (int x1, int y1, int x2, int y2)`, де `(x1,y1)` – координати початку, а `(x2, y2)` – координати кінця відрізка;

– функція виведення лінії з поточної точки в точку з заданими координатами: `lineto (int x, int y);` (відносно поточної позиції). Положення поточного вказівника приймається за початок координат (0,0) і вказується місце розташування кінця відрізка;

– функція виведення лінії з поточного положення вказівника до положення, заданого приростом його координат `linerel (int dx, int dy)`, де `dx, dy` – приріст координат нового положення вказівника; виведення точки за координатами (x, y) заданого кольору – `putpixel (int x, int y, int color)`.

Кола, еліпси, дуги:

– для креслення кіл використовується функція `circle (int x, int y, int radius)`; де `x, y` – координати центра кола, `radius` – радіус кола;

– креслення дуги радіуса `radius` із центром з координатами `(x,y)` від кута `stangle` до `endangle` (в градусах): `arc (int x, int y, int stangle, int endangle, int radius)`.

– креслення еліптичної дуги з аналогічними параметрами `ellipse(int x, int y, int stangle, int endangle, int xradius, int`

yradius); де xradius і yradius – розміри горизонтальної та вертикальної півосей відповідно.

Положення напрямку осі x (зліва-направо) приймають за 0 градусів, від'ємний напрямок осі y – за 90 градусів, оскільки проти годинникової стрілки.

Побудова прямокутників і ламаних:

– креслення прямокутника: `rectangle (int left, int top, int right, int bottom)`, де `(left, top)`, `(right, bottom)` – координати початку і кінця діагоналі. Прямокутник рисується з використанням поточного кольору і поточного стилю ліній;

– функція `bar (int left, int top, int right, int bottom)` рисує прямокутник, внутрішня частина якого залита за поточним шаблоном. Вона використовується в діловій графіці для побудови стовпцевих діаграм;

– функція побудови паралелепіпеда – `bar3d (int left, int top, int right, int bottom, int depth, int topflag)`.

Побудова ламаних

Щоб побудувати фігури з великою кількістю вершин, використовують функцію `drawpoly(int numpoints, const int *polypoints)`.

Ця функція дозволяє креслити на екрані дисплея довільну ламану, яка задана набором координат деякої множини точок. Це може бути як складна геометрична фігура, так і таблична математична функція.

Параметр `numpoints` визначає кількість точок, за якими будується ламана (причому, якщо необхідно нарисувати замкнений багатокутник з N вершинами, то значення кількості точок повинно бути на 1 більше N, n-координата (N + 1)-ої точки повинна бути така ж, як координата першої).

1.3 Встановлення кольорів ліній і фону

Різні адаптери підтримують різну кількість кольорів, які виводять одночасно на екран в графічному режимі, але для всіх **BGI** драйверів ця кількість обмежена діапазоном 0...15:

`Black`	чорний – 0;
`blue`	синій – 1;
`green`	зелений – 2;
`cyan`	блакитний – 3;
`red`	червоний – 4;
`magenta`	фіолетовий – 5;
`brown`	коричневий – 6;
`lightgray`	яскраво-сірий – 7;
`darkgrey`	темно-сірий – 8;
`lightblue`	яскраво-синій – 9;

`lightgreen`	яскраво-зелений – 10;
`lightcyal`	яскраво-блакитний – 11;
`lightred`	яскраво-червоний – 12;
`lightmagenta`	яскраво-фіолетовий – 13;
`yellow`	жовтий – 14;
`white`	білий – 15.

Максимальний номер кольору, що сприймається даним адаптером в поточному графічному режимі, може бути визначений за допомогою функції `getmaxcolor(void)`.

На екрані завжди розрізняють колір фону і колір ліній. Всі функції креслення фігур, якщо не містять в собі явного встановлення кольору, рисують фігури кольором ліній (як символи в текстовому режимі). Цей колір встановлюється функцією `setcolor(int color)`. Колір фону встановлюється функцією `setbkcolor(int color)`. Після використання цієї функції колір фону екрана одразу ж змінюється на заданий, параметр `color<=getmaxcolor()`. За замовчуванням колір фону = `0 (Black)`.

1.4 Керування стилем ліній, палітрою, штрихуванням

При кресленні лінії можна змінювати такі параметри:
- товщину лінії;
- тип лінії (пунктирні лінії, суцільні тощо).

В бібліотеці `graphics.h` визначені такі типи і сталі стилів зображених ліній:

```
void setlinestyle(int linestyle, unsigned upattern, int thickness);
```

`linestyle` – стиль
`upattern` – шаблон
`thickness` – товщина

Для значень параметра `LineStyle`:

`SOLID_LINE=0`	суцільна лінія (візерунок відсутній);
`DOTTED_LINE=1`	крапкова лінія-суцільна штрихова;
`CENTER_LINE=2`	штрихпунктирна лінія;
`DASHED_LINE=3`	пунктирна лінія;
`USERBIT_LINE=4`	тип лінії заданий дійсним шаблоном.

Для значень параметра `Thickness`:

`NORM_WIDTH=1`	товщина лінії в 1 піксель;
`THICK_WIDTH=3`	товщина лінії в 3 пікселя.

Отримати інформацію про поточний стиль ліній можна за допомогою функції:

```
getlinesettings(struct linesettingstype far *lineinfo);
```

Лістинг 1.4. Рисування ліній різних типів

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <graphics.h>
#include <string.h>

int main () {
    int gdriver=DETECT, gmode, errorcode;
    int style, midx, midy, userpat;
    char stylestr [40];

/* масив назв стилів ліній */
    char *lname[]={
  "SOLID_LINE",
  "DOTTED_LINE",
  "CENTER_LINE",
  "DASHED_LINE",
  "USERBIT_LINE"
    };
    clrscr ();
    initgraph (&gdriver, &gmode, "");
    errorcode=graphresult();
    if (errorcode!=grOk) {
  printf ("Graphics error: %s\n", grapherrormsg(errorcode));
  printf ("Press any key to halt...");
  getch ();
  exit (1);
}
midx=getmaxx ()/2;
midy=getmaxy ()/2;

// вибір палітри
userpat=1;
for (style=SOLID_LINE; style<=USERBIT_LINE; style++) {

// вибір стилю ліній
  setlinestyle (style, userpat, 1);

// конвертація змінної style в тип string
  strcpy (stylestr, lname[style]);

// рисування лінії
  line (0,0, midx-10, midy);
```

```
// рисування прямокутника
  rectangle (0,0, getmaxx (), getmaxy ());

// виведення повідомлення
  outtextxy (midx, midy, stylestr);
  getch ();
  cleardevice();
    }
    closegraph ();
    return 0;
}
```

На рисунках 1.3, 1.4 показаний результат роботи даної програми.

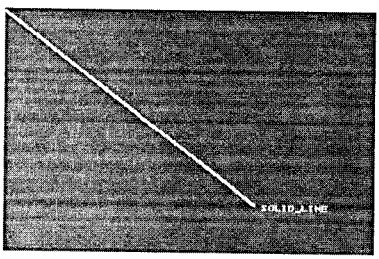


Рисунок 1.3 – Рисування суцільної лінії

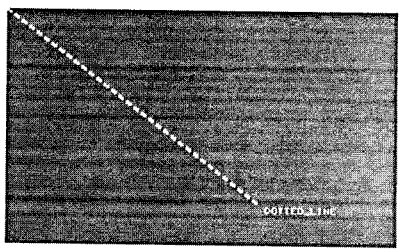


Рисунок 1.4 – Рисування лінії з точок

Лістинг 1.5. Рисування кіл за заданими координатами

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{  int gdriver = DETECT, gmode, errorcode;
   int midx, midy;
   int radius = 100;

 /* ініціалізація графіки */
   initgraph(&gdriver, &gmode, "");

 /* результат ініціалізації */
   errorcode = graphresult();
   if (errorcode != grOk)
   {
      printf("Graphics error: %s\n",
grapherrormsg(errorcode));
      printf("Press any key to halt:");
```

```
        getch();
        exit(1);
    }
    midx = getmaxx() / 2;
    midy = getmaxy() / 2;
    setcolor(getmaxcolor());

//функція рисування кола
    circle(midx, midy, radius);
    getch();
    closegraph();
    return 0;
}
```

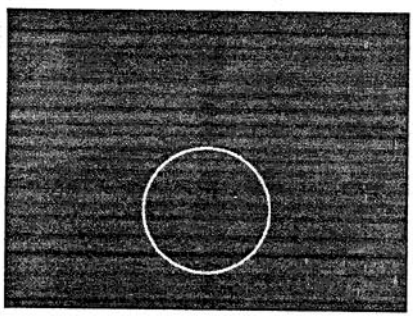


Рисунок 1.5 – Зображення результату роботи виконуваного файлу для прикладу 1.5

Лістинг 1.6. Використання генератора випадкових чисел для рисування прямокутників з різною товщиною контуру

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <dos.h>
    int main () {
    int gdriver=DETECT, gmode, errorcode;
    int X1,Y1,X2,Y2;
    clrscr ();
    initgraph(&gdriver, &gmode,"");
    errorcode=graphresult();
    if (errorcode!=grOk) {
    printf        ("Graphics        error:        %s\n",
grapherrormsg(errorcode));
    printf ("Press any key to halt...");
    getch ();
    exit (1);
    }
```

```
    do {
    getmaxcolor ();
    setcolor (WHITE);
    setlinestyle (random(4),0,random(10));
    X1=random(getmaxx()); Y1=random(getmaxy());
    X2=X1+random(getmaxx()); Y2=Y1+random(getmaxy());
    if (X2>getmaxx()) X2=getmaxx();
    if (Y2>getmaxy()) Y2=getmaxy();
    rectangle (X1,Y1,X2,Y2);
    delay(100);
    } while (!kbhit());
    return 0;
}
```

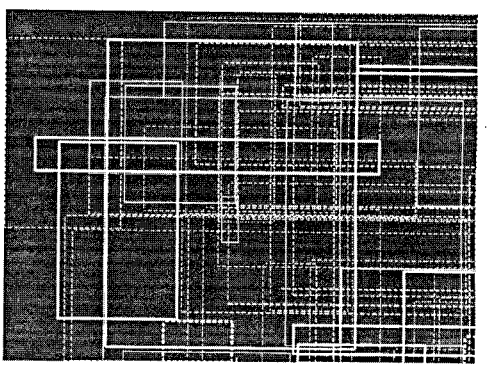


Рисунок 1.6 – Зображення результату роботи виконуваного файлу для прикладу 1.6

Лістинг 1.7. Рисування дуги

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <graphics.h>

int main () {
    int gdriver=DETECT, gmode, errorcode;
    int maxX,x,y;
    clrscr ();
    initgraph(&gdriver, &gmode,"");
    errorcode=graphresult();
    if (errorcode!=grOk)
    {
        printf ("Graphics error: %s\n",
                grapherrormsg(errorcode));
        printf ("Press any key to halt...");
        getch ();
        exit (1);
```

```
    }
    maxX=getmaxx();
    y=10;
    for (x=25;x<maxX-75;x++)
    {
         setcolor (x%16);
         arc (x,y,y%360,x%360,x/10);
         y+=x%2;
    }
    getch();
    closegraph ();
    return 0;
}
```

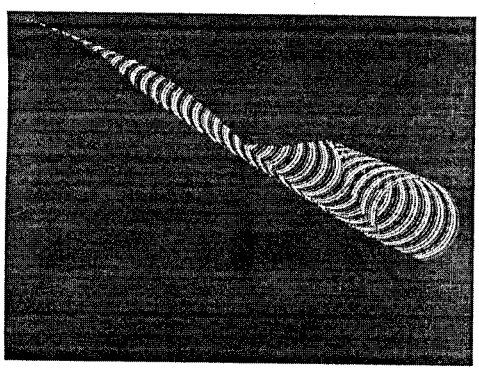


Рисунок 1.7 – Зображення результату роботи виконуваного файлу для прикладу 1.7

Лістинг 1.8. Рисування заповненого прямокутника

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <graphics.h>

int main () {
    int gdriver=DETECT, gmode, errorcode;
    int x1, y1, x2, y2;
    clrscr ();
    initgraph(&gdriver, &gmode,"");
    errorcode=graphresult();
    if (errorcode!=grOk)
    {
        printf ("Graphics error: %s\n",
                 grapherrormsg(errorcode));
        printf ("Press any key to halt...");
        getch ();
```

```
        exit (1);
    }
    x1=getmaxx()/2;
    y1=getmaxy()/4;
    x2=x1+25;
    y2=y1*2;
    setfillstyle (1,3);
    bar (x1,y1,x2,y2);
    getch ();
    closegraph ();
    return 0;
}
```

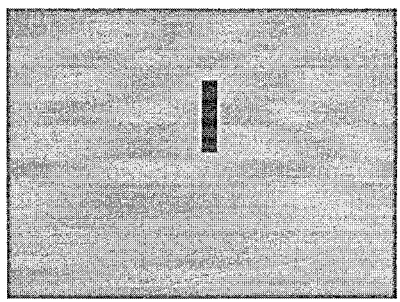


Рисунок 1.8 – Зображення результату роботи виконуваного файлу для прикладу 1.8

Лістинг 1.9. Рисування прямокутного паралелепіпеда

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <graphics.h>

#define FALSE 0
#define TRUE 1

int main () {
    int gdriver=DETECT, gmode, errorcode;
    int x1,y1,x2,y2,depth,top;
    clrscr ();
    initgraph(&gdriver, &gmode,"");
    errorcode=graphresult();
    if (errorcode!=grOk)
    {
        printf ("Graphics error: %s\n",
                 grapherrormsg(errorcode));
        printf ("Press any key to halt...");
        getch ();
```

```
        exit (1);
    }
    x1=getmaxx()/2;
    y1=getmaxy()/4;
    x2=x1+25;
    y2=y1*2;
    depth=10;
    top=TRUE;
    setfillstyle (1,3);
    bar3d (x1,y1,x2,y2,depth,top);
    getch ();
    closegraph ();
    return 0;
}
```

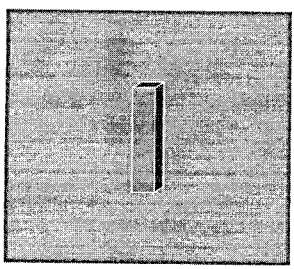


Рисунок 1.9 – Зображення результату роботи виконуваного файлу для прикладу 1.9

Лістинг 1.10. Програма для рисування еліпсів різних розмірів та кольорів

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <graphics.h>
#include <math.h>
#include <time.h>
int main () {
    int gdriver=DETECT, gmode, errorcode;
    int xmax, ymax, radius;
    randomize ();
    clrscr ();
    initgraph(&gdriver, &gmode,"");
    errorcode=graphresult();
    if (errorcode!=grOk)
    {
        printf ("Graphics error: %s\n",
                 grapherrormsg(errorcode));
        printf ("Press any key to halt...");
        getch ();
```

```
        exit (1);
    }
    xmax=getmaxx();
    ymax=getmaxy();
    radius=10;
    while (radius<ymax/2)
    {
        setcolor (1+abs(random(getmaxcolor())));
        ellipse (xmax/2, ymax/2,0,360,radius,50);
        ellipse (xmax/2, ymax/2,0,360,50,radius);
        radius+=10;
    }
    getch();
    closegraph ();
    return 0;
}
```

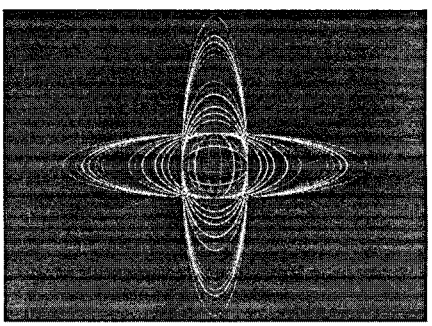


Рисунок 1.10 – Зображення результату роботи виконуваного файлу для прикладу 1.10

Лістинг 1.11. Рисування зафарбованих еліпсів

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <graphics.h>
#include <time.h>
int main () {
    int gdriver=DETECT, gmode, errorcode;
    int x,direction,count,xmax,ymax;
    clrscr ();
    initgraph(&gdriver, &gmode,"");
    errorcode=graphresult();
    if (errorcode!=grOk)
    {
        printf ("Graphics error: %s\n",
                grapherrormsg(errorcode));
        printf ("Press any key to halt...");
```

```
        getch ();
        exit (1);
    }
    xmax=getmaxx();
    ymax=getmaxy();
    x=20;
    direction=1; count=5;
    randomize ();
    setcolor (WHITE);
    while (!kbhit())
    {
        if ((x>xmax-50)||(x<=0))
        {
            direction=1;
            count+=5;
        }
        setfillstyle(random(12),
                     1+abs(random(getmaxcolor())));
        fillellipse (x,ymax/2, count+random(xmax/6),
                     count+random(ymax/4));
        x+=(50-direction);
    }
    getch();
    closegraph ();
    return 0;
}
```

Лістинг 1.12. Креслення лінії

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <graphics.h>
#include <dos.h>

int main () {
    int gdriver=DETECT, gmode, errorcode;
    int cmax,xmin,ymin,xmax,ymax;
    clrscr ();
    initgraph(&gdriver, &gmode,"");
    errorcode=graphresult();
    if (errorcode!=grOk)
    {
        printf ("Graphics error: %s\n",
                 grapherrormsg(errorcode));
        printf ("Press any key to halt...");
        getch();
        exit (1);
    }
    cmax=getmaxcolor();
    xmax=getmaxx();
    ymax=getmaxy();
```

```
    xmin=0;
    ymin=0;
    while ((xmin<xmax)||(ymin<ymax))
    {
        delay (25);
        setcolor(random(cmax));
        line(xmin,ymin,xmax,ymin);
        line(xmin,ymax,xmax,ymax);
        xmin++;
        ymin++;
        xmax--;
        ymax--;
    }
    getch();
    closegraph ();
    return 0;
}
```

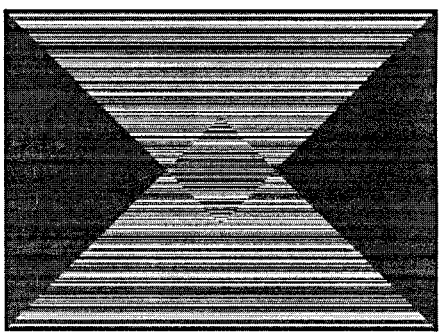


Рисунок 1.11 – Зображення результату роботи виконуваного файлу для прикладу 1.12

Лістинг 1.13. Креслення лінії щодо поточної позиції

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <graphics.h>
#include <time.h>
int main () {
    int gdriver=DETECT, gmode, errorcode;
    int cmax,xmax,ymax,color;
    clrscr ();
    initgraph(&gdriver, &gmode,"");
    errorcode=graphresult();
    if (errorcode!=grOk)
    {
        printf ("Graphics error: %s\n",
```

```
                grapherrormsg(errorcode));
        printf ("Press any key to halt...");
        getch ();
        exit (1);
    }
    cmax=getmaxcolor();
    xmax=getmaxx();
    ymax=getmaxy();
    randomize();
    while (!kbhit())
    {
        moveto(0,0);
        color=1+random(cmax);
        while (xmax>1)
        {
            setcolor(random(color));
            linerel(xmax,0);
            linerel(0,ymax);
            linerel(-xmax,0);
            linerel(0,-ymax);
            xmax-=2;
            ymax-=2;
            moveto(getx()+1,gety()+1);
        }
    }
    getch();
    closegraph ();
    return 0;
}
```

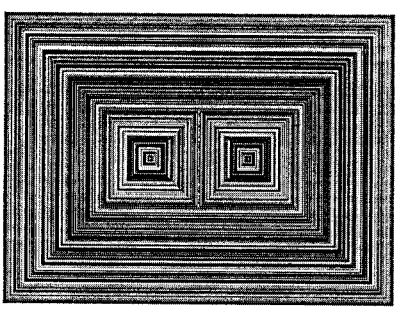


Рисунок 1.12 – Зображення результату роботи виконуваного файлу для прикладу 1.13

Лістинг 1.14. Формування ефекту мультиплікації

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <graphics.h>
```

```
#include <dos.h>
int main () {
    int gdriver=DETECT, gmode, errorcode;
    int dx,dy,x,y,maxx,maxy;
    clrscr ();
    initgraph(&gdriver, &gmode,"");
    errorcode=graphresult();
    if (errorcode!=grOk)
    {
        printf ("Graphics error: %s\n",
                 grapherrormsg(errorcode));
        printf ("Press any key to halt...");
        getch ();
        exit (1);
    }
    maxx=getmaxx();
    maxy=getmaxy();
    dx=maxx/4;
    dy=maxy/4;
    bar3d(dx,dy,maxx-dx,maxy-dy,30,6);
    setwritemode (XOR_PUT);
    do
    {
        x=random(maxx-dx);
        y=random(maxy-dy);
        rectangle(x,y,x+dx,y+dy);
        delay(200);
        rectangle(x,y,x+dx,y+dy);
    } while (!kbhit());
    closegraph();
    return 0;
}
```

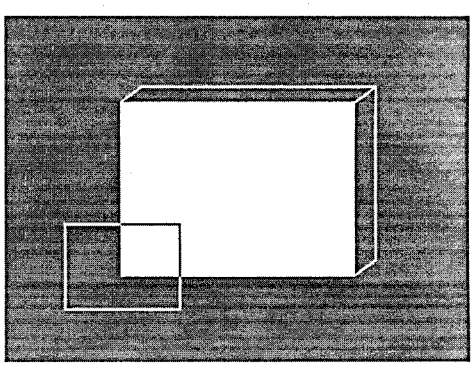


Рисунок 1.13 – Зображення результату роботи виконуваного файлу для прикладу 1.14

Керування кольорами і шаблонами заливання (фарби, палітра, заповнення)

Функція `getcolor()` повертає значення коду поточного кольору.

Функція `getmaxcolor()` повертає значення типу `int`, що містить максимально доступний код кольору, який можна використовувати для звертання до `setcolor()`.

Функція `setpalette( int colornum,int color)` заміщує один із кольорів в палітрі (`colornum` – вказує номер кольору в палітрі).

Функція заливання (шаблон заповнення)

`void setfillpattern(const char far *__upattern, int __color);`

Функція `setfillpattern` встановлює тип заливання і її колір для всіх типів заливання, які виконуються функціями `fillpoly()`, `bar()`, `floodfill()`, `bar3d()`, `pieslice()`.

Функція встановлює зразок і колір заливання (для функцій `fillpoly()`, `bar()`, `bar3d()`, `pieslise()`).

Всі дозволені значення зразків визначені в бібліотеці `graphics.h` у вигляді сталих.

Ось декілька прикладів:

`EMPTY_FILL=0;`	суцільне заливання кольором фона;
`SOLID_FILL=1;`	суцільне заливання поточним кольором;
`LINE_FILL=2;`	заливання лініями;
`HATCH_FILL=7;`	заливання рідким штрихуванням;
`USER_FILL=12;`	заливання визначене програмістом.

`color` – колір, яким виконується шаблон, колір фону при цьому залишається незмінним.

Заливання областей зображення

Існує ряд функцій, які рисують графічні фігури і одразу ж заповнюють їх за заданим шаблоном:

`bar(int left, int top, int right, int bottom)` рисує прямокутник, внутрішня область якого залита за поточним шаблоном.

Лістинг 1.15. Заливання області

```
do
{
     r=random(getmaxcolor());
     setfillstyle(1,7);
     x1=random(getmaxx());
     y1=random(getmaxy());
     x2=random(getmaxx() / 2);
     y2=random(getmaxy() / 2);
    bar(x1,y1,x2,y2);
```

```
        delay(200);
    } while(!kbhit());
```

Функція `bar3d(int left, int top, int right, int bottom, int depth, int topflag)` рисує паралелепіпед, лицьова сторона якого заливається за поточним шаблоном, а глибина задається в пікселях параметром `depth`.

Лістинг 1.16. Заливання області

```
setfillstyle(1,7);
X1=random(getmaxx());
Y1=random(getmaxy());
X2=random(getmaxx());
Y2=random(getmaxy());
bar3d(x1,y1,x2,y2,15,True);
delay(200);
bar3d(x1,y1,x2,y2,15,False);
delay(200);
bar3d(80,100,120,180,15,True);
getch(0);
```

Функція `fillellipse(int x, int y, int xradius, int yradius )` рисує еліпс поточним кольором і заповнює його за встановленим шаблоном.

Приклад:

```
setfillstyle(1,11);
setcolor(7);
fillellipse(x1,y1,x2,y2);
```

Заповнення більш складних геометричних фігур, в тому числі і неправильної форми, виконується функцією `fillpoly(int numpoints, const int far *polypoints)`.

Функція `floodfill(int x, int y, int border)` заливає всю область навколо точки (x, y), обмежену лініями кольору `border`. Наприклад, якщо точка (x, y) знаходиться всередині області, обмеженій колом, то вся область буде залита за шаблоном і кольором, встановленим функціями `setfillpattern()` або `setfillstyle()`. Якщо точка знаходиться поза цією областю, то залитим буде весь екран за винятком цієї області.

1.5 Ефекти мультиплікації

Функція `setwritemode( int mode )` задає режим порозрядного суміщення зображень, тобто режим, в якому при накладанні 2-х точок стирається або ні нижня точка (спосіб зняття верхнього зображення з екрана).

Дві послідовно проведені логічні операції XOR_PUT приведуть біти пам'яті монітора в початковий стан. Це означає, що якщо на екрані присутнє якесь зображення, то його можна використати як фон, нарисувати на ньому довільний рисунок, а потім відновити первинний вигляд. При ініціалізації і після зміни режимів встановлювати режим COPY_PUT.

Режим setwritemode() розповсюджується тільки на рисування відрізками, тобто функціями line(), lineto(), linerel(), rectangle(), drawpoly().

1.6 Робота з фрагментами зображення

Для запам'ятовування в буфері і відновлення з нього прямокутних фрагментів графічного зображення використовуються три функції. Це дуже зручно, оскільки дозволяє оперувати готовими елементами зображень.

```
imagesize(int left, int top, int_right, int bottom);
getimage(int left, int top, int right, int bottom, void far *bitmap);
putimage(int left, int top, const void far *bitmap, int op);.
```

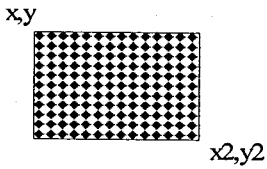
Функція imagesize() дозволяє визначити розмір прямокутного фрагмента в байтах. Прямокутник визначається координатами діагоналі (x1, y1), (x2, y2). Ця функція використовується разом з функцією malloc(). Записується зображення в буфер за допомогою функції getimage(), в якій параметри xl, yl, x2, y2 мають те ж значення, що в imagesize(). Параметр bitmap – нетиповий, повинен бути більше чи дорівнювати 4+розмір пам'яті, відведеної для області екрана (максимальний розмір збережувального зображення не повинен перевищувати 64К) і є зміною-буфером (4 байти = 2 слова, які містять ширину і висоту в пікселях зображення, що запам'ятовується).

Функція putimage(). Найбільш цікавим в цій функції є можливість визначити режим виведення зображення: можна додавати зображення на екрані і зображення в буфері; можна знищити зображення, яке знаходиться в певній області; можна інвертувати зображення, яке міститься в буфері. Ці операції задаються параметром op, для якого визначені такі сталі:

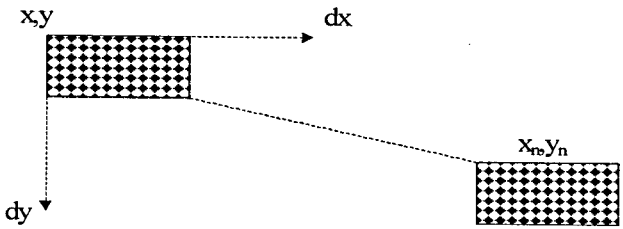
COPY_PUT=0;	операція MOV (заміщення);
XOR_PUT=1;	операція XOR;
OR_PUT=2;	операція OR;
AND_PUT=3;	операція AND;
NOT_PUT=4;	операція NOT.

Для формування динамічної складної картинки необхідно:

1. Вибрати розміри прямокутника, який містить динамічний об'єкт, тобто визначаються координати діагоналі (XI,YI) (X2,Y2).



2. Визначити крок приросту (для організації руху) по осях X і Y: `SxMove` і `SyMove` і кінцеву точка руху (Xn, Yn) в межах використовуваного екрана.



3. Задати тип заливання: функція `setfillstyle();`
4. Вибрати фонову картинку та її заливання;
5. Задати розмір динамічної картинки

```
Size =imagesize(x1,y1.x2,y2);
```

6. Утворити стек (буфер) в буферній пам'яті з однаковим визначенням його початкової адреси `malloc(size);`
7. Завантажити фрагмент в буфер `getimage(x1,y1,x2,y2,p);`
8. В циклі багатократнообраний фрагмент вивантажується з пам'яті функцією `putimage()` зі змінними координатами руху `putImage(sx, sy,p, XOR_PUT); sx=x+dx, sy=y+dy;`
9. Після закінчення рисування необхідно очистити буфер процедурою `free(Size).`

Лістинг 1.17. Робота з фрагментами зображень з використанням функцій

```
imagesize(), getimage(), putimage()
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#define ARROW_SIZE 10
void draw_arrow(int x, int y);
int main(void)
{
   int gdriver = DETECT, gmode, errorcode;
   void *arrow;
   int x, y, maxx;
```

```
      unsigned int size;
      initgraph(&gdriver, &gmode, "");
      errorcode = graphresult();
      if (errorcode != grOk)
      {printf("Graphics error: %s\n",
                grapherrormsg(errorcode));
       printf("Press any key to halt:");
       getch();
       exit(1);
      }
      maxx = getmaxx();
      x = 0;
      y = getmaxy() / 2;
```

/* рисування зображення, яке буде рухатись */

```
      draw_arrow(x, y);
```

/* визначення розміру зображення */

```
      size   =   imagesize(x,   y-ARROW_SIZE,   x+(4*ARROW_SIZE),
y+ARROW_SIZE);
```

/* відведення пам'яті під зображення*/

```
      arrow = malloc(size);
```

/* захоплення зображення*/

```
      getimage(x,         y-ARROW_SIZE,         x+(4*ARROW_SIZE),
y+ARROW_SIZE, arrow);
      while (!kbhit())
      {
```

/* вирізання попереднього зображення */

```
         putimage(x, y-ARROW_SIZE, arrow, XOR_PUT);
         x += ARROW_SIZE;
         if (x >= maxx)    x = 0;
```

/* вставляння нового зображення */

```
         putimage(x, y-ARROW_SIZE, arrow, XOR_PUT);
      }
      free(arrow);
      closegraph();
      return 0;
   }
   void draw_arrow(int x, int y)
   {
```

/* рисування стрілки на екрані */

```
      moveto(x, y);
      linerel(4*ARROW_SIZE, 0);
      linerel(-2*ARROW_SIZE, -1*ARROW_SIZE);
```

```
    linerel(0, 2*ARROW_SIZE);
    linerel(2*ARROW_SIZE, -1*ARROW_SIZE);
}
```

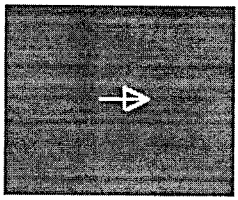


Рисунок 1.14 – Зображення результату роботи виконуваного файлу для прикладу 1.17

Лістинг 1.18. Робота з фрагментом зображення і формування динамічної картинки

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <graphics.h>
#include <dos.h>

const r=10; // радіус рухомої кулі
void GrInit ();// прототип функції ініціалізації графіки

void main () {
    int x1,y1,x2,y2,sx,sy;// змінні для оживлення фрагмента
    int maxx,maxy,sxmove,symove;
    int size;// розмір фрагмента
    char *p; // вказівник на буфер
    GrInit();// виклик функції ініціалізації графіки

/* максимальне поле екрана */
    maxx=getmaxx();
    maxy=getmaxy();

/*координати області екрана, в якому нарисовано кульку і яка буде збереженим фрагментом*/
    x1=(maxx/2)-r;
    y1=(maxy/2)-r;
    x2=x1+2*r;
    y2=y1+2*r;
    sx=x1;
    sxmove=3;// початкова точка руху
    sy=y1;
    symove=-1;// крок переміщення кульки
```

```
      setfillstyle(1,RED);// вибір типу заливання
      pieslice(x1+r,y1+r,0,360,r);// рисування кульки
      size=imagesize(x1,y1,x2,y2);// розмір фрагмента в байтах
   /*розміщення буферу (оператор new відводе місце в size байт в пам'яті, присвоюючи адресу його початку вказівнику p */
      p=new char(size);
      getimage (x1,y1,x2,y2,p); // фрагмент в буфер
      setfillstyle (CLOSE_DOT_FILL,BLUE);// тип заливання фону
      bar (50,50, maxx-50, maxy-50);// фонова картинка
      do { // починається рух кульки
         putimage (sx,sy,p,XOR_PUT);// виведення кульки
         delay (10);// затримка 10 мс
         putimage (sx,sy,p,XOR_PUT);// стирання кульки
   /* обмеження на рух кульки в рамках поля */
         if ((sx<50)||(sx>maxx-50-2*r)) sxmove=-sxmove;
         if ((sy<50)||(sy>maxy-50-2*r)) symove=-symove;
   /* наступна точка появи на екрані кульки, доки не натиснута клавіша */
         sx+=sxmove;
         sy+=symove;
      } while (!kbhit());
      free(p);// вивільнення пам'яті буфера
      closegraph ();
   }
   void GrInit () {
      int gdriver=DETECT, gmode, errorcode;
      clrscr ();
      initgraph(&gdriver, &gmode,"");
      errorcode=graphresult();
      if (errorcode!=grOk) {
     printf ("Graphics error: %s\n", grapherrormsg(errorcode));
     printf ("Press any key to halt...");
     getch ();
     exit (1);
        }
   }
```

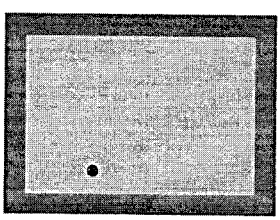


Рисунок 1.15 – Зображення результату роботи виконуваного файлу для прикладу 1.18

1.7 Керування відеосторінками

1. Функція `setvisualpage (int page)`, яка встановлює видимою на екрані відеосторінку номер `page`.

2. Функція `setactivepage(int page)` встановлює перенаправлення всіх графічних операцій на сторінку номер `page`.

Лістинг 1.19. Керування відеосторінками (тільки для адаптерів VGA і EGA)

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main()
{  /* вибір драйвера і режиму */
  int gdriver = EGA, gmode = EGAHI, errorcode;
  int x, y, ht;
  initgraph(&gdriver, &gmode, "");
   errorcode = graphresult();
  if (errorcode != grOk)
  {          printf("Graphics          error:          %s\n",
grapherrormsg(errorcode));
    printf("Press any key to halt:");
    getch();
    exit(1);
  }
  x = getmaxx() / 2;
  y = getmaxy() / 2;
  ht = textheight("W");

  /*  вибір невидимої сторінки для рисування*/
  setactivepage(1);

  /* рисування лінії на сторінці #1 */
   line(0, 0, getmaxx(), getmaxy());

  /* виведення повідомлення на сторінці #1 */
  settextjustify(CENTER_TEXT, CENTER_TEXT);
  outtextxy(x, y, "This is page #1:");
  outtextxy(x, y+ht, "Press any key to halt:");

  /* вибір рисунка для сторінки #0 */
  setactivepage(0);

  /* виведення повідомлення на сторінці #0 */
  outtextxy(x, y, "This is page #0.");
  outtextxy(x, y+ht, "Press any key to view page #1:");
```

```
    getch();
    /* вибір сторінки #1 як видимої */
    setvisualpage(1);
    getch();
    closegraph();
    return 0;
}
```

1.8 Вибір шрифту і стилю

В бібліотеці `graphics.h` є набір шрифтів, які можуть бути використані для виведення інформації:

```
DEFAULT_FONT= 0
TRIPLEX_FONT= 1
SMALL_FONT= 2
SANS_SERIF_FONT= 3
GOTHIC_FONT= 4
SCRIPT_FONT= 5
SIMPLEX_FONT= 6
TRIPLEX_SCR_FONT= 7
COMPLEX_FONT= 8
EUROPEAN_FONT = 9
BOLD_FONT= 10
```

Щоб обрати потрібний шрифт, потрібно використати функцію `settextstyle(int font, int direction, int charsize)`.

Параметр `font` – номер шрифту; `direction` – розташування тексту. Можливі лише дві орієнтації тексту; відзначені такими константами:

```
HORIZ_DIR = 0;
VERT_DIR = 1;
```

`Charsize` – розмір кожного символу.

Для виведення тексту використовують дві функції:

`outtext(const char far *textstring);` – виводить на графічний екран рядок `textstring`, орієнтований відносно позиції поточного вказівника;

`outtextxy(int x, int y, const char far *textstring);` – виводить рядок, орієнтований відносно координат x, y.

Шрифт попередньо може бути встановлений викликом функції `settextstyle()`.

Розмір букв та рядків

Важливо знати вертикальний та горизонтальний розміри рядка, що виводиться, у пікселях. Це дозволяє розташовувати рядок пропорційно роздільній здатності графічного режиму.

Функції

```
textheight(const char far *textstring);
textwidth(const char far *textstring);
```

повертають ширину та довжину рядка `textstring` в пікселях.

1.9 Контрольні запитання

1. Яка бібліотека функції призначена для керування графічними режимами?
2. Які файли мають бути у робочому каталозі для роботи у графічному режимі?
3. Охарактеризуйте функціональні групи бібліотеки `graphics.h`?
4. Яким чином відбувається ініціалізація графічного режиму?
5. Які ви знаєте процедури для рисування графічних примітивів?
6. Для чого призначена функція `drawpoly()`, які її параметри?
7. Який діапазон кольорів BGI драйверів?
8. Як здійснити перехід від графічного режиму до текстового?
9. Які способи заливання зображення ви знаєте?
10. Що ви знаєте про логічні операції `XOR_PUT` і `COPY_PUT`?
11. Які функції використовуються для роботи з фрагментами зображення?
12. Дайте коротку характеристику функціям керування відеосторінками?
13. В якій бібліотеці міститься набір шрифтів для виведення інформації?
14. Які функції використовуються для виведення тексту?
15. Яким чином можна змінити розмір букв та рядків у графічному режимі?

1.10 Практикум з програмування

1. Напишіть функцію ініціалізації графічного режиму з урахуванням обробки помилок ініціалізації.
2. Напишіть функцію для переходу з графічного режиму в текстовий.
3. Напишіть функцію для рисування кола з заданими координатами.
4. Використовуючи генератор випадкових чисел, напишіть програму рисування прямокутників з різною товщиною ліній.
5. Напишіть програму для рисування дуг різними кольорами.
6. Напишіть функцію для рисування заповненого прямокутника заданого розміру.
7. Напишіть функцію для рисування паралелепіпеда, заповненого заданим кольором.
8. Напишіть функцію для рисування різнокольорових заповнених еліпсів.
9. Напишіть програму для рисування рухомого місяця вздовж лінії горизонту нічного неба.
10. Наведіть приклад заливання області.

2 ОСОБЛИВОСТІ РОЗРОБКИ ГРАФІЧНИХ ЗАСТАВОК В ПРОФЕСІЙНИХ ПРОГРАМАХ

Однією з характерних рис сучасних програм є наявність яскравих анімацій, що виводяться на екран та приваблюють увагу користувача. В даному розділі розглядаються основні принципи створення графічних заставок на прикладі розробки заставки відомого у 2000-2003 роках комп'ютерного мультику "Musyanya", розробленого російськими програмістами.

2.1 Загальні відомості

Робота в графічному режимі виконується за допомогою попередньо визначених констант, типів та функцій стандартної бібліотеки **graphics.h**. Для того щоб отримати складний рисунок необхідно подати його у вигляді сукупності найпростіших геометричних елементів.

Розглянемо особливості програмування рисунка, що представлений на рисунку 2.1.

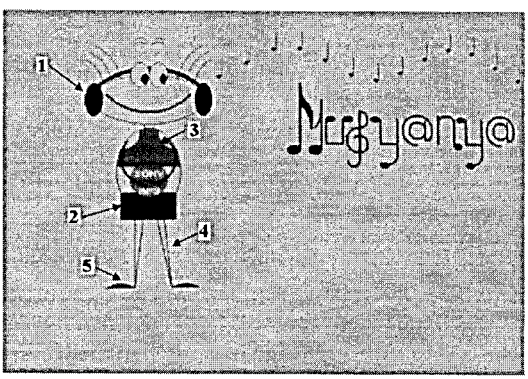


Рисунок 2.1 – Приклад графічної заставки

Щоб спростити цю задачу, розіб'ємо рисунок на дві основи: напис слова "Musyanya" та зображення дівчинки. Ці основи розпадаються на більш прості об'єкти. Наприклад, напис складається з кожної окремо рисованої літери, а зображення дівчинки з таких основних складових людського тіла, як голова, тулуб та ноги. Кожен з цих простих об'єктів можна подати як сукупність ще простіших складових. Такий процес ділення елементів об'єкту продовжується до тих пір, поки вони не являтимуть собою сукупністю найпростіших геометричних фігур, для виведення на екран яких існують готові функції в стандартній бібліотеці.

2.2 Особливості розробки статичної частини графічної заставки

Розглянемо програму для виведення на екран напису, що складається з 6 функцій, кожна з яких відповідає за рисування окремої літери.

Почнемо з функції, яка рисує літеру «М». Ця задача складається з виведення на екран таких найпростіших елементів, як лінія (1, 2), еліпс (3, 4), дуга (5, 6, 7, 8).

```
void S_M(int x1,int y1)
{
setcolor(8);
setfillstyle(1,8);
setlinestyle(0,1,3);
line(x1,y1,x1,y1-80); //Виведення лінії 1
fillellipse(x1-7,y1,7,5);  //Виведення еліпса 3
line(x1+25,y1,x1+25,y1-60);      //Виведення лінії 2
fillellipse(x1+18,y1,7,5); // Виведення еліпса 4
arc(x1,y1-45,313,0,25);     //Виведення дуги 5
arc(x1-26,y1-25,360,50,40);      //Виведення дуги 6
arc(x1+26,y1-75,180,230,25);     //Виведення дуги 7
arc(x1-10,y1-40,340,50,25);      //Виведення дуги 8
setfillstyle(1,8);
setfillstyle(1,8);
floodfill(x1+2,y1-59,8);
}
```

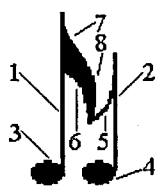


Рисунок 2.2 – Рисування літери «М»

За таким же принципом на екран виводяться всі інші літери: «U», «S», «Y», «A», «N».

Рисування дівчинки в даному випадку знаходиться в основній частині програми, в функції `main()`. Як вже було зазначено, задача виведення на екран дівчинки розпалася на 3 задачі: виведення голови, тулуба та ніг. Для того, щоб нарисувати голову, треба розуміти, що вона складається, зокрема, з контуру обличчя, очей, брів, рота, волосся та навушників.

Контур обличчя являє собою еліпс, центром якого є точка з координатами (170, 120), горизонтальний радіус 60, а вертикальний 30 точок.

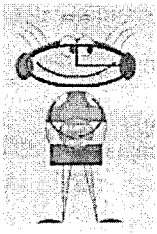


Рисунок 2.3 – Рисування голови

```
setcolor(8);
setfillstyle(1,14);
setlinestyle(0,1,1);
fillellipse(170,120,60,30); //рисує зафарбований еліпс – голову
```

Навушники – це два еліпси, між якими дві дуги, а простір між дугами залитий темно-сірим кольором. Код, завдяки якому рисуються навушники, є таким:

```
setcolor(8);
setfillstyle(1,8);
setlinestyle(0,1,1);
fillellipse(110,120,10,20);      //рисує еліпс – лівий навушник
fillellipse(230,120,10,20);      //рисує еліпс – правий навушник
arc(170,165,40,140,85);
arc(170,170,40,140,85);     //рисує дуги з'єднання двох навушників
floodfill(170,82,8); //зафарбовує відокремлену дугами область
```

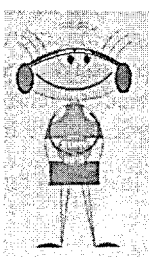


Рисунок 2.4 – Рисування навушників

На даному рисунку очі являють собою еліпси. Для реалізації очей в даному випадку вибраний такий програмний код:

```
setfillstyle(1,15);
fillellipse(160,90,10,15);
          //рисує зафарбований білий еліпс – ліве око
fillellipse(180,90,10,15);
```

```
                //рисує зафарбований білий еліпс – праве око
setfillstyle(1,1);
fillellipse(165,95,4,6); );
                //рисує зафарбований синій еліпс – ліве око
fillellipse(185,95,4,6); );
                //рисує зафарбований синій еліпс – праве око
```

Рот, мабуть, найпростіший об'єкт обличчя, тому що це проста дуга кола, від початкового кута 220 до кінцевого кута 320 градусів з радіусом 60 точок. Точка з координатами (170, 75) використовується як центр кола.

```
setcolor(8);
setlinestyle(0,1,3);
arc(170,75,220,320,60);      //рисує дугу – рот
```

```
//Волосся на голові у дівчинки виконано також за допомогою дуг.
setlinestyle(0,1,1);
arc(260,105,120,170,60);
arc(260,105,120,170,50);
arc(260,105,120,170,40);
arc(80,105,10,60,60);
arc(80,105,10,60,50);
arc(80,105,10,60,40);        //дуги – волосся
```

Голова вже майже нарисована, не вистачає брів, але оскільки це рухома частина, то детальніше їх зображення буде описане нижче.

Розглянемо тулуб, який можна подати як сукупність деталей: контур тулуба, шия, руки, майка, спідниця та плеєр. Як і контур голови контур тулуба являє собою простий еліпс, який з'єднує з головою шия. Шия подана як маленький прямокутник. Спідничка теж зображена досить символічно, у вигляді прямокутника, але вже більшого. Для виведення на екран зафарбованих прямокутників використовується функція `bar3d()`, де параметр глибини паралелепіпеда приймає значення 0. Нижче наведений код для виведення на екран контуру тулуба, шиї та спіднички.

```
setcolor(8);
setfillstyle(1,14);
setlinestyle(0,1,1);
fillellipse(170,205,35,50);      //рисує еліпс – тулуб
bar3d(165,150,175,155,0,1);      //рисує паралелепіпед – шия
setlinestyle(0,1,1);
setfillstyle(1,1);
bar3d(140,230,200,260,0,1); //рисує паралелепіпед – спідниця
```

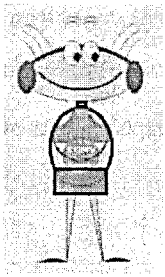


Рисунок 2.5 – Рисування тулуба та спідниці

Для того, щоб нарисувати майку, використовуються вже існуючі лінії тулуба та додаються дві дуги – отвори для рук – та лінія низу. Отримані контури зафарбовуються червоним кольором за допомогою функції `floodfill()`. Це відбувається за допомоги таких програмних інструкцій:

```
arc(140,160,270,5,20);
arc(200,160,173,275,20);
line(135,200,205,200);   //рисує 2 дуги та лінію – контури майки
setfillstyle(1,12);
floodfill(145,190,8); //зафарбовує відокремлену дугами область – майка(3)
```

Музичний CD-плеєр – це просто зафарбований еліпс з однаковими радіусами, що, насправді, є колом. А напис на плеєрі зроблено з використанням функції `outtextxy()`. Отже, все це можна подати у вигляді:

```
setcolor(8);
setfillstyle(1,7);
setlinestyle(0,1,1);
fillellipse(170,215,20,20);        //рисує зафарбований еліпс – музичний CD-плеєр
setcolor(BLACK);
outtextxy(155,205,"SONY");

//виводить рядок тексту – напис на плеєрі
outtextxy(160,225,"MP3");

//руки показані як великі дуги, а пальці – як маленькі лінії.
setlinestyle(0,1,3);
arc(170,190,280,50,30);
arc(170,190,140,260,30);    //дуги – руки
setcolor(8);
line(160,227,165,220);line(167,228,165,220);line(172,225,165,220);
line(165,216,175,220);line(175,210,175,220);line(170,213,175,220);    //лінії – пальці
// ноги можна подати як зафарбовані сектори кола з малим кутом.
```

```
setfillstyle(1,14);
pieslice(155,340,83,90,80);
pieslice(195,340,101,95,81);
```
//виводить на екран зафарбовані сектори // кола – ноги (4)

// нерухома ступня є зафарбованою областю, яка обмежена лінією та дугою.

```
line(155,340,125,340);
arc(140,360,52,120,25);
setfillstyle(1,8);
floodfill(145,339,8);
```
//зафарбовується область, яка обмежена лінією та дугою – нерухома ступня (5)

2.3 Особливості розробки динамічної частини графічної заставки

Нерухома частина зображення вже виконана у вигляді програмного коду на мові С. Залишилося лише відобразити на екрані рух лівої ступні, брів та нот. Для того, щоб створити рух рисунка необхідно прорисувати кожну фазу руху та, плавно заміщаючи їх одна одною, виводити на екран. Для цього рухома частина рисунку зберігається в пам'яті за допомогою функції `getimage()` та для виведення її на екран – функції `putimage()`. Отже, спочатку рисуємо та запам'ятовуємо фази різних рухомих предметів. Наприклад, для збереження вигляду ноти використовується такий програмний код:

```
line(20,5,20,25);
```
//виводить лінію (частина ноти)

//виводить зафарбований еліпс (частина ноти)

```
fillellipse(17,26,3,2);
```

// sizecircle значення числа байтів, необхідних для зберігання прямокутної області екрана

```
sizecircle=imagesize(0,0,30,30);
```
//присвоєння змінній

//виділення пам'яті розміром sizecircle з динамічної області

```
map=malloc(sizecircle);
```

//зберігає в буфері прямокутну область екрана

```
getimage(0,0,30,30,map);
```

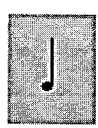


Рисунок 2.6 – Рисування ноти

Щоб рух відбувався неперервно до натиснення будь-якої клавіші, використовується цикл `while(!kbhit())`. В середині цього циклу виводиться на екран попередньо збережена фаза 1, відбувається затримка, перераховуються координати нот та виводиться фаза 2 різних рухливих елементів.

```
while (!kbhit()) //поки не натиснута будь-яка клавіша
{
putimage(140,40,fas1,1);
putimage(195,320,fas3,1);
putimage(140,40,fas2,1);
putimage(195,320,fas4,1);
delay(150);
for (i=0;i<n;i++)
{
if (currentx[i]>229)
putimage(currentx[i],currenty[i],map,1);
currentx[i]+=3;
currenty[i]=50+20*sin((currentx[i]+40)*pi/100);
          if (currentx[i]>620)
          //розрахунок нових координат кожної з нот
{
currentx[i]=230;
currenty[i]=50+20*sin((currentx[i]+40)*pi/100);
}
if (currentx[i]>229)

//виводяться ноти з новими координатами
putimage(currentx[i],currenty[i],map,1);
}
putimage(140,40,fas1,1);
putimage(195,320,fas4,1);
putimage(140,40,fas2,1);
putimage(195,320,fas3,1);
delay(150);
}
```

Після закінчення руху вивільняється пам'ять, яка була виділена під збереження фаз руху, за допомоги операції `free()`.

Лістинг програми рисування графічної заставки

```
#include <stdio.h>
#include <graphics.h>
#include <stdlib.h>
#include <dos.h>
#include <conio.h>
#include <alloc.h>
#include <math.h>
#define Y 180           //визначення константи «Y»
#define x 340          //визначення константи «x»
#define pi 3.1415926  //визначення константи «pi»
#define n 13          //визначення константи «n»
void initgr();       //прототип функції ініціалізації графіки
void S_M(int,int);    //прототип функції, яка рисує літеру «M»
```

```
   void S_U(int,int);    //прототип функції, яка рисує літеру
«U»
   void S_S(int,int);    //прототип функції, яка рисує літеру
«S»
   void S_Y(int,int);    //прототип функції, яка рисує літеру
«Y»
   void S_A(int,int);    //прототип функції, яка рисує літеру
«A»
   void S_N(int,int);    //прототип функції, яка рисує літеру
«N»

   void main()
    {
     unsigned int sizecircle, sizeoffas1, sizeoffas2,
     sizeoffas3,sizeoffas4;
     void *map,*fas1,*fas2,*fas3,*fas4;
     int currentx[n],currenty[n];
     int i;

   //виклик функції ініціалізації графіки
     initgr();

   //встановлює колір фону (15) - білий
     setbkcolor(15);

   //встановлює поточний колір рисунка (8) - темно-сірий
     setcolor(8);

   //встановлює тип лінії та її товщину
     setlinestyle(0,1,1);

   //встановлює тип зафарбовування та його колір
     setfillstyle(1,8);

   //виводить лінію (частина ноти)
       line(20,5,20,25);

   //виводить зафарбований еліпс(частина ноти)
     fillellipse(17,26,3,2);

   //присвоєння змінній sizecircle значення числа байтів,
   //необхідних для зберігання прямокутної області екрана
     sizecircle=imagesize(0,0,30,30);

   //виділення пам'яті розміром sizecircle з динамічної облас-
ті
     map=malloc(sizecircle);

   //зберігає в буфері прямокутну область екрана
     getimage(0,0,30,30,map);

   //скидає стан поточного графічного екрана
```

```
    cleardevice();

    S_M(x,Y);       //виклик функції, яка рисує літеру «M»
    S_U(x+30,Y-7); );   //виклик функції, яка рисує літеру «U»
    S_S(x+60,Y-7); );   //виклик функції, яка рисує літеру «S»
    S_Y(x+95,Y-7); );   //виклик функції, яка рисує літеру «Y»
    S_A(x+123,Y-7); ); //виклик функції, яка рисує літеру «A»
    S_N(x+165,Y-7); ); //виклик функції, яка рисує літеру «N»
    S_Y(x+200,Y-7); ); //виклик функції, яка рисує літеру «Y»
    S_A(x+228,Y-7); ); //виклик функції, яка рисує літеру «A»

    setcolor(8);
    setfillstyle(1,14);
    setlinestyle(0,1,1);
    fillellipse(170,120,60,30);   //рисує зафарбований еліпс -
голову

    setcolor(8);
    setfillstyle(1,8);
    setlinestyle(0,1,1);
    fillellipse(110,120,10,20); //рисує лівий навушник
    fillellipse(230,120,10,20); //рисує правий навушник

  //рисує дуги, які є границями
  //з'єднання двох навушників
    arc(170,165,40,140,85);
    arc(170,170,40,140,85);
    floodfill(170,82,8);     //зафарбовує відокремлену дугами
область
    setfillstyle(1,15);

  //рисує зафарбований білий еліпс - ліве око
    fillellipse(160,90,10,15);

  //рисує зафарбований білий еліпс - праве око
    fillellipse(180,90,10,15);
    setfillstyle(1,1);

  //рисує зафарбований синій еліпс - ліве око
    fillellipse(165,95,4,6); );

  //рисує зафарбований синій еліпс - праве око
    fillellipse(185,95,4,6); );
    setcolor(8);
    setlinestyle(0,1,3);
    arc(170,75,220,320,60);  //рисує дугу - рот
    setlinestyle(0,1,1);
    arc(260,105,120,170,60);
    arc(260,105,120,170,50);
    arc(260,105,120,170,40);
    arc(80,105,10,60,60);
    arc(80,105,10,60,50);
```

```
  arc(80,105,10,60,40);     //дуги - волосся

  setcolor(8);
  setfillstyle(1,14);
  setlinestyle(0,1,1);
//рисує зафарбований еліпс - тулуб
  fillellipse(170,205,35,50);
//рисує зафарбований паралелепіпед - шия
  bar3d(165,150,175,155,0,1);
  setlinestyle(0,1,1);
  setfillstyle(1,1);
//рисує зафарбований паралелепіпед - спідниця
  bar3d(140,230,200,260,0,1); );

//рисує 2 дуги та лінію - контури майки
  arc(140,160,270,5,20);
  arc(200,160,173,275,20);
  line(135,200,205,200);
  setfillstyle(1,12);

//зафарбовує відокремлену дугами область - майка
  floodfill(145,190,8);
  setcolor(8);
  setfillstyle(1,7);
  setlinestyle(0,1,1);

//рисує зафарбований еліпс - музичний CD-плеєр
  fillellipse(170,215,20,20);
  setcolor(BLACK);

//виводить рядок тексту - напис на плеєрі
  outtextxy(155,205,"SONY");
  outtextxy(160,225,"MP3");
  setcolor(8);

//лінії - пальці
  line(160,227,165,220);
  line(167,228,165,220);
  line(172,225,165,220);
  line(165,216,175,220);
  line(175,210,175,220);
  line(170,213,175,220);

//дуги - руки
  setlinestyle(0,1,3);
  arc(170,190,280,50,30);
  arc(170,190,140,260,30);

//виводить на екран зафарбовані сектори кола - ноги
  setfillstyle(1,14);
  pieslice(155,340,83,90,80);
  pieslice(195,340,101,95,81);
```

```
//зафарбовується область, яка обмежена лінією
//та дугою - нерухома ступня
  line(155,340,125,340);
  arc(140,360,52,120,25);
  setfillstyle(1,8);
  floodfill(145,339,8);
  setcolor(8);
  setlinestyle(0,1,1);
  arc(160,75,50,130,10);
  arc(160,80,50,130,10);
  arc(180,75,50,130,10);
  arc(180,80,50,130,10);   //дуги - брови (фаза 1)
  sizeoffas1=imagesize(140,40,200,73);
  fas1=malloc(sizeoffas1);

//зберігає в буфері прямокутну
//область екрана, в якій попередньо виведені брови (фаза 1)
  getimage(140,40,200,73,fas1);
  putimage(140,40,fas1,1);
  line(195,340,225,340);
  arc(210,360,52,120,25);
  setfillstyle(1,8);

//зафарбовується область, яка обмежена лінією
//та дугою - рухома ступня  (фаза 1)
  floodfill(200,339,8);
  sizeoffas3=imagesize(195,320,225,340);
  fas3=malloc(sizeoffas3);

//зберігає в буфері прямокутну
//область екрана, в якій попередньо виведена ступня(фаза 1)
  getimage(195,320,225,340,fas3);
  putimage(195,320,fas3,1);
  setcolor(8);
  setlinestyle(0,1,1);
  arc(160,55,50,130,10);
  arc(160,60,50,130,10);
  arc(180,55,50,130,10);
  arc(180,60,50,130,10);   //дуги - брови (фаза 2)
  sizeoffas2=imagesize(140,40,200,60);
  fas2=malloc(sizeoffas2);

//зберігає в буфері прямокутну
//область екрана, в якій попередньо виведені брови (фаза 2)
  getimage(140,40,200,60,fas2);
  putimage(140,40,fas2,1);
  line(195,340,220,325);
  arc(218,349,85,155,25);
  setfillstyle(1,8);
  floodfill(207,332,8);
  sizeoffas4=imagesize(195,320,225,340);
  fas4=malloc(sizeoffas4);
```

```
   //зафарбовується область, яка
   //обмежена лінією та дугою - рухома ступня      (фаза 2)
     getimage(195,320,225,340,fas4);
     putimage(195,320,fas4,1);
     for (i=0;i<n;i++)
      {

   //розрахунок початкових координат
   //кожної з нот
        currentx[i]=230-30*i;
        currenty[i]=50+20*sin((currentx[i]+40)*pi/100);
      }
      putimage(currentx[0],currenty[0],map,1);

   //виводяться брови в стані фаза1
      putimage(140,40,fas1,1);
      putimage(195,320,fas4,1);    //виводиться ступня в стані
фаза1

        while (!kbhit()) //поки не натиснута будь-яка клавіша
        {
          putimage(140,40,fas1,1);
          putimage(195,320,fas3,1);
          putimage(140,40,fas2,1);
          putimage(195,320,fas4,1);
          delay(150);
     for (i=0;i<n;i++)
      {
        if (currentx[i]>229)
        putimage(currentx[i],currenty[i],map,1);
        currentx[i]+=3;
        currenty[i]=50+20*sin((currentx[i]+40)*pi/100);
        if (currentx[i]>620)

     //розрахунок нових координат кожної з нот
          {
          currentx[i]=230;
          currenty[i]=50+20*sin((currentx[i]+40)*pi/100);
          }
        if (currentx[i]>229)
        putimage(currentx[i],currenty[i],map,1);

     //виводяться ноти з новими координатами
      }
      putimage(140,40,fas1,1);
      putimage(195,320,fas4,1);
      putimage(140,40,fas2,1);
      putimage(195,320,fas3,1);
      delay(150);
     }
     getch();
        free(map);
```

```
      free(fas1);
      free(fas2);
      free(fas3);

//вивільнення пам'яті
      free(fas4);
  }

 void initgr()
  {
   int grdriver=DETECT,grmode,errorcode,color;
   initgraph(&grdriver,&grmode,"../bgi");
    errorcode=graphresult();
   if (errorcode != grOk)
     {
      printf("Error!");
       getch();
      exit(1);
     }
   setbkcolor(9);
  }

void S_M(int x1,int y1)     //функція, яка рисує літеру «М»
                            // за заданими координатами
 {
  setcolor(8);
  setfillstyle(1,8);
  setlinestyle(0,1,3);
  line(x1,y1,x1,y1-80);
  fillellipse(x1-7,y1,7,5);
  line(x1+25,y1,x1+25,y1-60);
  fillellipse(x1+18,y1,7,5);
  arc(x1,y1-45,313,0,25);
  arc(x1-26,y1-25,360,50,40);
  arc(x1+26,y1-75,180,230,25);
  arc(x1-10,y1-40,340,50,25);
  setfillstyle(1,8);
  setfillstyle(1,8);
  floodfill(x1+2,y1-59,8);
 }

//функція, яка рисує літеру «U» за заданими координатами
void S_U(int x2,int y2)
 {
  setcolor(8);
  setfillstyle(1,8);
  setlinestyle(0,1,3);
  line(x2,y2,x2,y2-30);
  fillellipse(x2+6,y2-30,5,3);
  line(x2+20,y2,x2+20,y2-30);
  fillellipse(x2+27,y2-30,5,3);
  line(x2,y2,x2+20,y2);
```

```
 }

//функція, яка рисує літеру «Y» за заданими координатами
void S_Y(int x4,int y4)
 {
  setcolor(8);
  setfillstyle(1,8);
  setlinestyle(0,1,3);
  line(x4,y4,x4,y4-30);
  fillellipse(x4-6,y4-30,5,3);
  line(x4+20,y4+10,x4+20,y4-30);
  line(x4,y4,x4+20,y4);
  arc(x4+6,y4+10,260,360,15);
  fillellipse(x4+5,y4+25,5,3);
 }

//функція, яка рисує літеру «A» за заданими координатами
void S_A(int x5,int y5)
 {
  setcolor(8);
  setfillstyle(1,8);
  setlinestyle(0,1,3);
  arc(x5+17,y5-17,325,270,18);
  arc(x5+17,y5-17,0,360,10);
  line(x5+26,y5-5,x5+26,y5-28);
  line(x5+26,y5-5,x5+32,y5-5);
 }

//функція, яка рисує літеру «N» за заданими координатами
void S_N(int x6,int y6)    {
  setcolor(8);
  setfillstyle(1,8);
  setlinestyle(0,1,3);
  line(x6,y6,x6,y6-35);
  line(x6+20,y6,x6+20,y6-30);
  arc(x6+11,y6-25,0,180,10);
  fillellipse(x6+25,y6+2,5,3);
 }

//функція, яка рисує літеру «S» за заданими координатами
void S_S(int x7,int y7)
 {
  setcolor(8);
  setfillstyle(1,8);
  setlinestyle(0,1,3);
  line(x7+10,y7-40,x7+10,y7+20);
  arc(x7+10,y7-30,270,90,10);
  arc(x7+10,y7-5,90,270,15);
  arc(x7+10,y7,270,130,10);
  arc(x7+3,y7+20,180,360,7);
 }
```

2.4 Контрольні запитання

1. Яку бібліотеку необхідно підключити для роботи в графічному режимі?
2. Які функції використовуються для рисування найпростіших геометричних фігур?
3. Яким чином здійснюється заливання деякої області певним кольором?
4. За допомогою якої команди ми можемо змінювати стиль заливання?
5. Які параметри має функція `arc`?
6. Для чого використовується функція `pieslice`?
7. За допомогою яких функцій ми створюємо рухоме зображення?
8. Як здійснюється виділення пам'яті певного розміру з динамічної області?
9. Як забезпечити неперервний рух зображення до натиснення клавіші?
10. Як звільнити пам'ять, що була виділена під збереження фаз руху?

2.5 Практикум з програмування

1. Напишіть фрагмент програми для побудови складної фігури з декількох ліній.
2. Напишіть фрагмент програми для побудови олімпійських кілець.
3. Напишіть фрагмент програми для побудови концентричних кіл.
4. Напишіть фрагмент програми для побудови будь-яких фігур з різнокольоровими квадратами.
5. Напишіть фрагмент програми для побудови трикутників за допомогою ліній.
6. Напишіть фрагмент програми для побудови фігури з кіл різного радіусу.
7. Напишіть фрагмент програми для побудови трикутників, вписаних в прямокутники.
8. Напишіть фрагмент програми, що зафарбовує область, яка обмежена лініями.
9. Напишіть фрагмент програми для побудови будь-якого рухомого зображення.
10. Напишіть фрагмент програми для побудови руху кола по горизонталі і одночасного руху квадрата по діагоналі.

3 ОСНОВНІ ПРИНЦИПИ СТВОРЕННЯ БАГАТОІЄРАРХІЧНОГО МЕНЮ МОВОЮ С

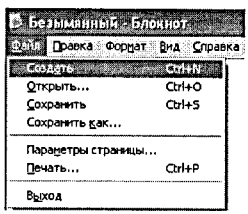
3.1 Загальні відомості

Меню – це інтерфейсний елемент або список об'єктів, з якого необхідно зробити вибір. Об'єкти меню називаються пунктами або командами. Меню можуть мати багаторівневу структуру, при цьому зберігається принцип послідовного переходу. В кожному меню наявні пункти, які є тимчасово недоступними, тобто обрати їх неможливо. Для програми, що підтримує багатодокументний інтерфейс, присутність визначеного меню є обов'язковою.

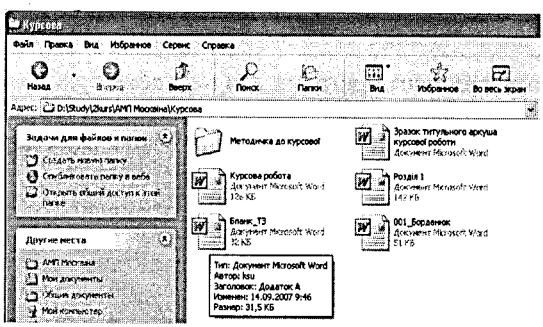
Види меню

1) Горизонтальне – menu bar:

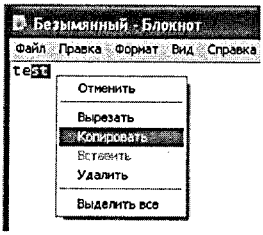
2) Низхідне (підпункт горизонтального меню) – pull down:



3) Висхідне меню – pop-up:



4) Контекстне меню



Існує ряд функцій, які дозволяють прикладній програмі працювати з меню. Ці функції призначені для:

- дозволу/відміни ряду команд меню;
- вставлення/зняття спеціальних відміток для команд меню;
- додання/змінення та видалення команд меню;
- використання зображень як команд меню;
- заміщення елементів меню;
- створення та ініціалізація меню.

3.2 Способи створення робочого вікна

Підключаємо стандартні бібліотеки:

```
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
#include<stdlib.h>
```

Гарний інтерфейс передбачає роботу з вікнами. Вікна меню будемо рисувати за допомогою функції `Window (int x1,int y1,int x2,int y2,int bg,int rect,char ch)`, де `int x1, y1` – відповідно, початкові координати x, y; `int x2, y2` – відповідно, кінцеві координати x, y; `int bg` – колір тексту; `int rect` – колір фону; `char ch` – це змінна, яка вказує є рамочка чи ні ('y' – є, 'n' – немає). Для цього використаємо такі оператори:

```
void Window(int x1,int y1,int x2,int y2,int bg,int
rect,char ch)
    {
     textcolor(bg);  textbackground(rect);
     for(int i=x1; i<=x2; i++)
     for(int j=y1; j<=y2; j++)
     {
```

За допомогою коду 219 (чи лінії ▌) рисуємо широку лінію. Це:

```
        gotoxy(i,j); cprintf("%c",219);
        // або cprintf("Ы");
     }
     if (ch=='y')
     {
        textcolor(rect); textbackground(bg);
```

```
for(int i=x1; i<=x2; i++)
gotoxy(i,y1); cprintf("%c",205); //abo cprintf("H");
gotoxy(i,y2); cprintf("%c",205); //abo cprintf("H");
```

За допомогою коду 205 (чи лінії ═) рисуємо горизонтальні поля рамочки. Ось результат:

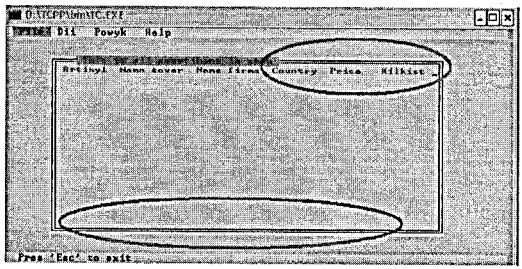


Рисунок 3.1 – Рисування горизонтальних ліній прямокутника

```
for(int j=y1; j<=y2; j++)
        gotoxy(x1,j);          cprintf("%c",186);//abo
cprintf("e");
        gotoxy(x2,j);          cprintf("%c",186);//abo
cprintf("e");
```

За допомогою коду 186 (чи лінії ‖) ми рисуємо вертикальні поля рамочки:

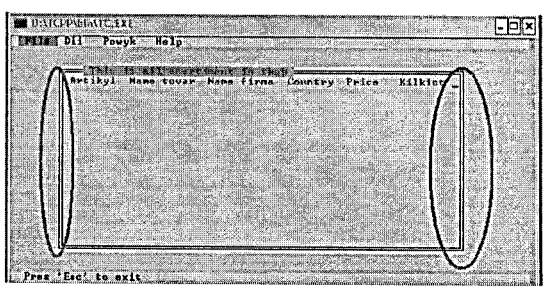


Рисунок 3.2 – Рисування вертикальних ліній прямокутника

`gotoxy(x2,y2); cprintf("%c",188);//abo cout<<'j';//188`
За допомогою коду 188 рисуємо кути типу: ╝ (рис. 3.3).
`gotoxy(x2,y1); cprintf("%c",187);//abo cout<<'»';//187`
За допомогою коду 187 рисуємо кути типу: ╗ (рис. 3.4).
`gotoxy(x1,y2); cprintf("%c",200);//abo cout<<'И';//200`
За допомогою коду 200 рисуємо кути типу: ╚ (рис. 3.5).

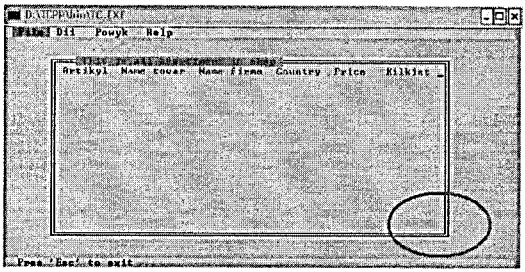


Рисунок 3.3 – Рисування нижнього правого кута прямокутника

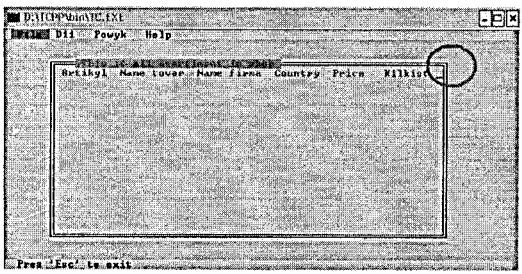


Рисунок 3.4 – Рисування верхнього правого кута прямокутника

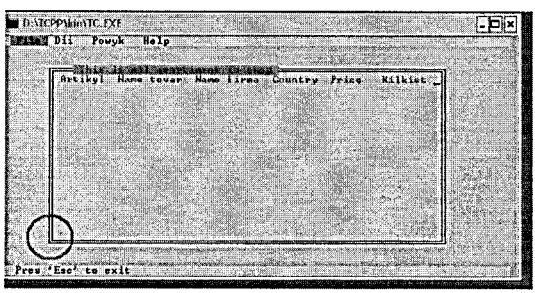


Рисунок 3.5 – Рисування нижнього лівого кута прямокутника

`gotoxy(x1,y1);  cprintf("%c",201);`//abo cout<<'Й';//201
За допомогою коду 201 рисуємо кути типу: ╔ (рис. 3.6).

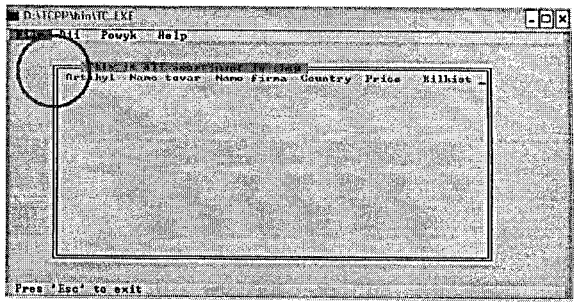


Рисунок 3.6 – Рисування верхнього лівого кута прямокутника

3.3 Алгоритм створення пунктів та підпунктів меню

1. При плануванні структури меню та визначенні кількості пунктів меню використовують такі оператори:

```
const MinMIndex = 1;
const MaxMIndex = 4;
int MIndex,PMIndex;
```

`const MinMIndex=1` – це мінімальна кількість пунктів меню; `const MaxMIndex = 4` – це максимальна кількість пунктів меню; `PMIndex` – це змінна, яка вказує на «підменю».

```
void MainMenu()
{ textbackground(1);
```

2. Встановимо колір основного фону меню – синій.

```
clrscr();
Window(0,1,78,1,7,1,'n');                    (1)
Window(0,25,79,25,7,1,'n');                  (2)
```

3. За допомогою функцій (1) і (2) рисуємо верхнє вікно, де розміщені написи File, Dii, Powyk, Help, та нижнє вікно, де знаходиться надпис Pres 'Esc' to exit відповідно.

4. Встановлюємо колір тексту чорний `(textcolor(0))` та колір фону яскраво-сірий `(textbackground(7))`

```
textcolor(0); textbackground(7);
```

5. Переходимо до координат (3,25)

```
gotoxy(3,25);
```

та пишемо Pres 'Esc' to exit

```
cprintf("Pres 'Esc' to exit ");
```

6. Встановлюємо чорний колір тексту `(textcolor(0))` та зелений фон `(textbackground(2))`

```
textcolor(0); textbackground(2);
```

7. Переходимо до мітки (2,1)

```
gotoxy(2,1);
```

і пишемо назву пункту меню File

```
cprintf(" File ");
```

8. Встановлюємо чорний колір тексту `(textcolor(0))` та яскраво-сірий фон `(textbackground(7))`

```
textcolor(0); textbackground(7);
```

і організуємо вивід на екран назви пунктів меню Dii, Powyk, Help

```
cprintf(" Dii  ");
cprintf(" Powyk  ");
cprintf(" Help ");
MIndex = MinMIndex;
}
```

//End draw main menu

```
void podmenu1()
{
int pch;
```

9. У першому підменю, підменю File, рисуємо вікно функцією

```
Window(2,2,16,5,7,0,'y');
```

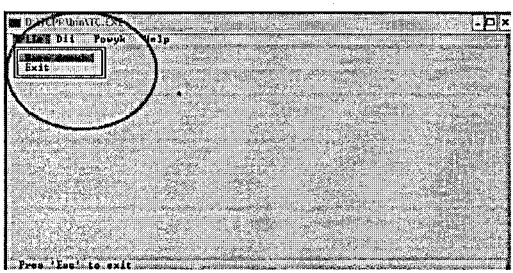


Рисунок 3.7 – Рисування підменю File

10. У функції `Window(2,2,16,5,7,0,'y')` – координати 2,2 – це початкові координати x, y відповідно; 16,5 – це кінцеві координати x, y відповідно; 7 – це яскраво-синій колір фону; 0 – це колір тексту; 'y' – рамочка є.

11. Переходимо до мітки (3,4) та (3,3), де пишемо Exit, Basa danuh відповідно.

```
gotoxy(3,4); cprintf(" Exit        ");
textbackground(2); – встановлюємо зелений колір фону
gotoxy(3,3); cprintf(" Basa danuh ");
do
{ pch=getch();
 switch (pch)
{
```

12. Для організації руху по підменю за допомогою клавіші ↓ (код 80) необхідно використати такі оператори:

```
case 80:
                PMIndex++;
                switch(PMIndex){
                case 2:
```

Встановлюємо яскраво-сірий колір фону

```
                textbackground(7);
```

Переходимо до мітки (3,3) та пишемо Basa danuh

```
                gotoxy(3,3); cprintf(" Basa danuh  ");
```

Встановлюємо зелений колір фону:

```
                textbackground(2);
```

Переходимо до мітки (3,4), де пишемо Exit

```
                    gotoxy(3,4); cprintf(" Exit       ");
                break;
                default:
```

13. Для обробки процесу натискання кнопки Basa danuh необхідно використовувати такі оператори:

```
                PMIndex=1;
```

Встановлюємо яскраво-сірий колір фону

```
                    textbackground(7);
```

На метці (3,4), де написано Exit

```
                gotoxy(3,4); cprintf(" Exit       ");
```

При цьому змінюється фон Basa danuh та стає зеленим

```
                textbackground(2);
                gotoxy(3,3); cprintf(" Basa danuh ");
                break;
                }// end switch(PMIndex)
                break; //end 80 Down
```

14. Для організації руху по підменю за допомогою клавіші 72, тобто ↑, працюємо за тим самим принципом. Основне, що потрібно пам'ятати: `textbackground(7);` – фон яскраво-сірий; `gotoxy(x,y);` – мітка, до якої потрібно перейти.

```
case 72:
                PMIndex--;
                switch(PMIndex)
                {
                case 1:
                textbackground(7);
                gotoxy(3,4); cprintf(" Exit       ");
                textbackground(2);
                gotoxy(3,3); cprintf(" Basa danuh ");
                break;
                 default:
                PMIndex=2;
                textbackground(7);
                gotoxy(3,3); cprintf(" Basa danuh  ");
                textbackground(2);
```

```
                    gotoxy(3,4);
                    cprintf(" Exit        ");
                    break;
                    }// end switch(PMIndex)
                    break; //end 72  Up
                     case 13:     // if enter press
```

15. При натисканні клавіші Enter, повинне з'явитися вікно, яке можна нарисувати за допомогою:

```
switch (PMIndex)
  {        case 1:
                    Window(2,2,16,6,1,0,'n');
```

- потрібне для того, щоб попереднє вікно зникло

```
                    Window(8,4,69,22,7,0,'y');
```

- потрібне для того, щоб нарисувати вікно з певними координатами та з рамочкою

Переходимо до координат `gotoxy(12,4)`

```
                    gotoxy(12,4);
```

та встановлюємо білий колір шрифта

```
                    textcolor(15);
```

і коричневий фон напису This is all asortiment in shop

```
                    textbackground(6);
                    cprintf(" This is all asortiment in shop
");
```

встановлюємо яскраво-сірий фон

```
                    textbackground(7);
```

та переходимо до мітки:

```
                     gotoxy(9,5);
                     textcolor(15);
```

і білим кольором пишемо складових підменю:

```
          cprintf(" Artikyl ");
          cprintf(" Name tovar ");
          cprintf(" Name firma ");
          cprintf(" Country ");
          cprintf(" Price  ");
          cprintf(" Kilkist ");
                    //OpenRec();
                    getch();
                    pch = 27;
                    break;
              case 2:
                    exit(1);
                    pch = 27;
                    break;      }
                     break; //end 13
                     } // end switch(pch)
                    }while(pch!=27); //Esc
                    Window(1,2,75,24,1,0,'n');
                    }
```

Отже, вікно буде мати вигляд, наведений на рисунку 3.8.

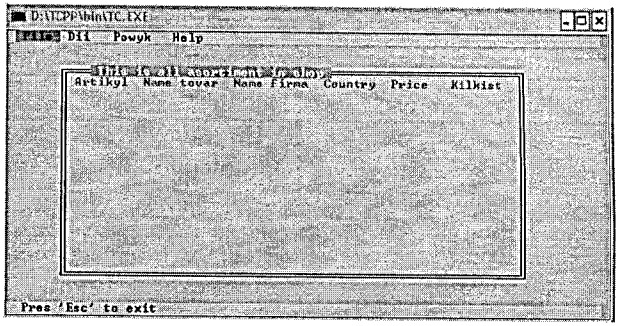


Рисунок 3.8 – Рисування меню Baza danuh

3.4 Правила реалізації пунктів меню

Заходимо в підменю Dii.

```
void podmenu2()
{ int pch;
Window(8,2,16,5,7,0,'y');
```

- рисуємо вікно, де розміщені написи Zminutu, Dodatu на відповідних мітках

```
gotoxy(9,4);
cprintf("Zminutu");
textbackground(2);
gotoxy(9,3);
cprintf("Dodatu");
do
```

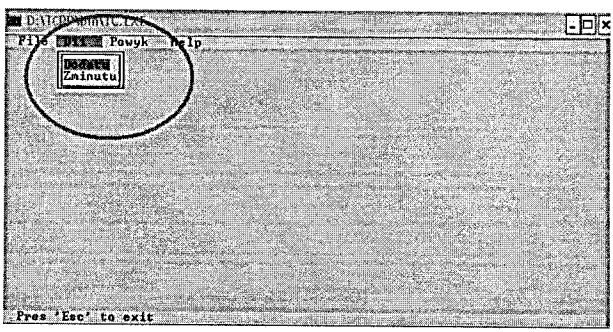


Рисунок 3.9 – Рисування підменю Дії

```
{
     pch=getch();
      switch (pch)
     {
      case 80: – стрілочкою рухаємося вниз
```

```
PMIndex++;
if (PMIndex==2)
```

Для обробки дій при натисканні на пункт підменю Zminutu необхідно використати нижченаведені оператори:

```
{
textbackground(7);
```

- встановлюємо яскраво-сірий колір фону

```
gotoxy(9,3);
cprintf("Dodatu");
```

- переходимо до мітки з координатами (9,3) та пишемо ім'я підменю – Dodatu

```
textbackground(2);
```

- встановлюємо зелений колір фону

```
gotoxy(9,4);
cprintf("Zminutu");
```

- переходимо до мітки з координатами (9,4) та пишемо ім'я підменю – Zminutu

```
}
else {
PMIndex=1;
```

Якщо натиснуто Dodatu:

```
textbackground(2);
```

- встановлюємо зелений колір фону

```
gotoxy(9,3);
cprintf("Dodatu");
```

- переходимо до мітки з координатами (9,3) та пишемо ім'я підменю – Dodatu

```
textbackground(7);
```

- встановлюємо яскраво-сірий колір фону

```
gotoxy(9,4);
cprintf("Zminutu");
```

- переходимо до мітки з координатами (9,4) та пишемо ім'я підменю – Zminutu

```
 }
break; //end 80  Down
case 72:
```

За допомогою цієї клавіші рухаємося вверх, принцип дії той самий

```
PMIndex--;
if (PMIndex==1)
{
textbackground(7);
gotoxy(9,4);
cprintf("Zminutu");
textbackground(2);
gotoxy(9,3);
cprintf("Dodatu");
}
else {
 PMIndex=2;
```

```
                    textbackground(7);
                    gotoxy(9,3);
                    cprintf("Dodatu");
                    textbackground(2);
                    gotoxy(9,4); cprintf("Zminutu");
                    }
                   break; //end 72  Up
  case 13:
```

Для обробки дій при натисканні клавіші Enter на пункті підменю Dodatu, необхідно використати нижченаведені оператори:

```
if (PMIndex==1)
{
Window(2,2,16,5,1,0,'n');
```

При цьому потрібно, щоб попереднє вікно зникло

```
Window(8,4,38,18,7,0,'y');
```

- та нарисувати вікно з певними координатами та з рамкою

```
gotoxy(10,4);
```

- переходимо до мітки (10,4)

```
textbackground(6);
```

- встановлюємо коричневий колір фону

```
textcolor(15);
```

- встановлюємо білий колір тексту

```
cprintf(" Please input new record ");
```

- пишемо напис Please input new record

```
gotoxy(10,6);
```

- переходимо до мітки (10,6)

```
textcolor(15);
```

- встановлюємо білий колір тексту

```
textbackground(7);
```

- встановлюємо яскраво-сірий колір фону.

Далі на відповідних мітках пишемо відповідні назви складових підменю:

```
cprintf("Artikyl        : ");
gotoxy(10,8);
cprintf("Name tovar     : ");
gotoxy(10,10);
cprintf("Name firma     : ");
gotoxy(10,12);
cprintf("Country        : ");
gotoxy(10,14);
cprintf("Price          : ");
gotoxy(10,16);
cprintf("Kilkist tovary : ");
gotoxy(4,1);
getch();
pch=27;
}
```

В результаті на екрані з'явиться вікно, вигляд якого наведено на рисунку 3.10.

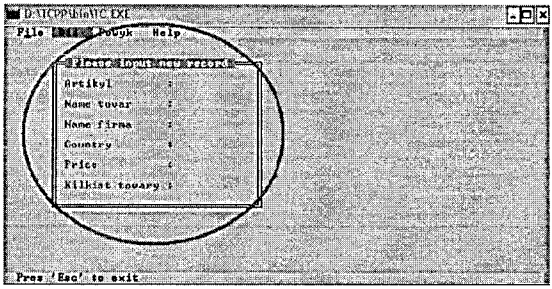


Рисунок 3.10 – Рисування меню Dodatu

```
else
```

Якщо ж натиснуто пункт підменю Zminutu, то з'являються вікна, вигляд яких наведений на рисунку 3.11. Для цього необхідно написати такий код:

```
 {
Window(1,2,16,5,1,0,'n');
```

- щоб зникло попереднє вікно

```
Window(3,3,45,6,7,0,'y');
```

- щоб з'явилось нове вікно

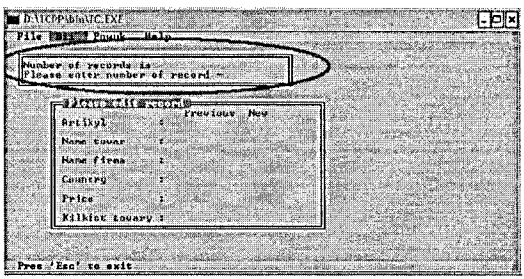


Рисунок 3.11 – Рисування меню Zminutu

```
textcolor(15);
```

- встановлюємо білий колір тексту

```
gotoxy(4,4);
```

- переходимо до мітки (4,4)

```
cprintf("Number of records is ");
```

- пишемо

```
gotoxy(4,5);
```

- переходимо до мітки (4,5)

```
cprintf("Please enter number of record -
");
Window(8,8,50,21,7,0,'y');
```

- щоб з'явилось вікно 1 (рис. 3.12) з:

```
gotoxy(10,8);
```

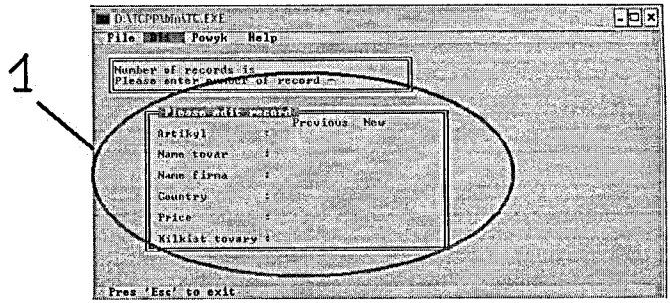


Рисунок 3.12 – Рисування меню Zminutu

- переходимо до мітки (4,4)

```
textcolor(15);
```

- встановлюємо білий колір тексту

```
textbackground(6);
```

- встановлюємо коричневий колір фону

```
cprintf(" Please edit record ");
```

- пишемо Please edit record

Далі переходимо до міток з певними координатами та формування відповідних пунктів підменю, можна використати такий код:

```
gotoxy(10,10);
textcolor(15);
textbackground(7);
cprintf("Artikyl         : ");
gotoxy(10,12);
cprintf("Name tovar      : ");
gotoxy(10,14);
cprintf("Name firma      : ");
gotoxy(10,16);
cprintf("Country         : ");
gotoxy(10,18);
cprintf("Price           : ");
gotoxy(10,20);
cprintf("Kilkist tovary : ");
gotoxy(29,9);
textbackground(7);
cprintf("Previous  New");
getch();
 pch=27;
 }
break; //13
} // end switch(pch)
```

```
}while(pch!=27); //Esc
Window(1,2,75,24,1,0,'n');
```

Такий код необхідний для того, щоб попереднє вікно зникло при натисканні клавіші Esc.

При натисканні пункту меню Powyk

```
void podmenu3()
{ int pch;
Window(14,2,25,6,7,0,'y');
```

- з'являється вікно.

На певних мітках відповідні підпункти меню можна вказати так:

```
gotoxy(15,4); cprintf(" Artikyl ");
gotoxy(15,5); cprintf(" Price   ");
textbackground(2);
gotoxy(15,3); cprintf(" Firma   ");
do
```

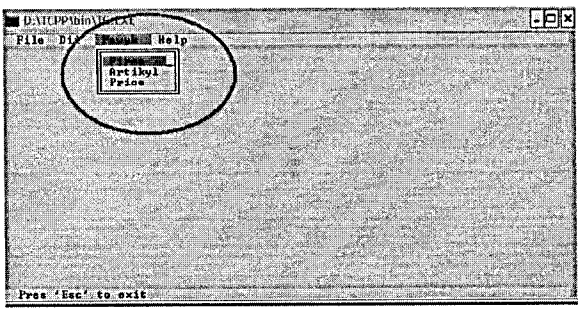


Рисунок 3.13 – Рисування підменю Powyk

```
{
 pch=getch();
 switch (pch)
 {
case 80:
```

При натисканні стрілочки «вниз»:

```
PMIndex++;
switch(PMIndex)
{
 case 2:
```

Якщо натиснути на пункт меню Artikyl:

```
textbackground(7);
```

- встановлюємо яскраво-сірий фон

```
gotoxy(15,3); cprintf(" Firma   ");
```

- переходимо до мітки з координатами (15,3) та пишемо назву підменю – Firma

```
textbackground(2);
```

- встановлюємо зелений фон

```
gotoxy(15,4); cprintf(" Artikyl ");
```

- переходимо до мітки, з координатами (15,4) та пишемо назву підменю – Artikyl

```
break;
case 3:
```

Якщо натиснути на пункт меню Price:

```
textbackground(7);
```

- встановлюємо яскраво-сірий фон

```
gotoxy(15,4); cprintf(" Artikyl ");
```

- переходимо до мітки з координатами (15,4) та пишемо назву підменю – Artikyl

```
textbackground(2);
```

- встановлюємо зелений фон

```
gotoxy(15,5);
cprintf(" Price   ");
```

- переходимо до мітки з координатами (15,4) та пишемо назву підменю – Price

```
break;
default:
PMIndex=1;
```

Якщо натиснути на пункт меню Firma:

```
textbackground(7);
```

- встановлюємо яскраво-сірий фон

```
gotoxy(15,4);
cprintf(" Artikyl ");
```

- переходимо до мітки з координатами (15,4) та пишемо назву підменю – Artikyl

```
gotoxy(15,5);
cprintf(" Price   ");
```

- переходимо до мітки з координатами (15,4) та пишемо назву підменю – Price

```
textbackground(2);
```

- встановлюємо зелений фон

```
gotoxy(15,3);
cprintf(" Firma   ");
```

- переходимо до мітки з координатами (15,3) та пишемо ім'я підменю – Firma

```
break;
}// end switch(PMIndex)
break; //end 80  Down
case 72:
```

Аналогічно формується програмний код для обробки дій, що відбуваються при натисканні стрілочки «вгору»

```
PMIndex--;
switch(PMIndex)
{
 case 1:
textbackground(7);
gotoxy(15,4);
cprintf(" Artikyl ");
textbackground(2);
gotoxy(15,3);
cprintf(" Firma   ");
break;
 case 2:
textbackground(7);
gotoxy(15,5);
cprintf(" Price   ");
textbackground(2);
gotoxy(15,4);
cprintf(" Artikyl ");
break;
default:
PMIndex=3;
textbackground(7);
gotoxy(15,3);
cprintf(" Firma   ");
gotoxy(15,4);
cprintf(" Artikyl ");
textbackground(2);
gotoxy(15,5);
cprintf(" Price   ");
break;
}// end switch(PMIndex)
break; //end 72  Up
case 13:
```

Якщо натиснути на Enter

```
 switch (PMIndex){
```

на першому підпункті меню, тобто на Firma, на екрані з'являється вікно,

```
case 1:
Window(1,2,25,7,1,15,'n');
```

яке потрібне для того, щоб попереднє вікно зникло.

```
Window(3,3,45,5,7,0,'y');
```

Щоб нарисувати вікно з певними координатами та з рамочкою

```
gotoxy(5,4);
```

переходимо до координат (5,4)

```
textcolor(15);
```

та встановлюємо білий колір тексту

```
cprintf("Please enter name of firma - ");
```

На рисунку 3.14 показане нарисоване вищенаведеними операторами вікно.

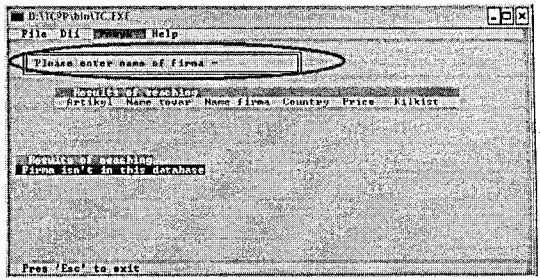


Рисунок 3.14 – Рисування підменю Firma

```
Window(8,8,69,8,7,7,'n');
```

Для того, щоб нарисувати вікно з певними координатами та без рамочки

```
gotoxy(9,8);
textcolor(15);
```

встановлюємо білий колір тексту.

Переходимо до міток з певними координатами та пишемо назви складових

```
cprintf(" Artikyl ");
cprintf(" Name tovar ");
cprintf(" Name firma ");
cprintf(" Country ");
cprintf(" Price  ");
cprintf(" Kilkist ");
Window(8,7,69,7,6,6,'n');
gotoxy(11,7);  textcolor(15);
cprintf("Results of seaching");
```

На рисунку 3.15 наведене нарисоване вищенаведеними операторами вікно.

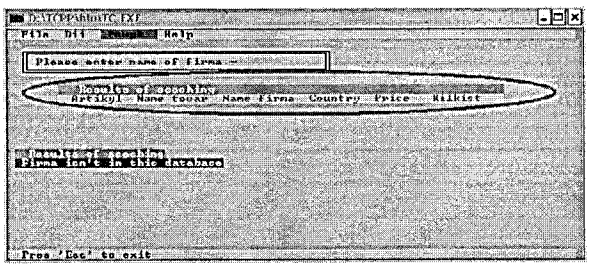


Рисунок 3.15 – Рисування підменю Firma

Якщо запитуваної фірми немає, то потрібно створити віконечко, яке про це інформує

```
gotoxy(2,14);
```

Для цього переходимо до координат (2,14)

```
textbackground(6);
```

- встановлюємо коричневий колір фону і пишемо:

```
cprintf(" Results of seaching");
gotoxy(2,15);
```

- переходимо до координат (2,15)

```
textbackground(RED);
```

- встановлюємо коричневий колір фону і пишемо:

```
cprintf(" Firma isn't in this database");
```

На рисунку 3.16 наведене нарисоване вищенаведеними операторами вікно.

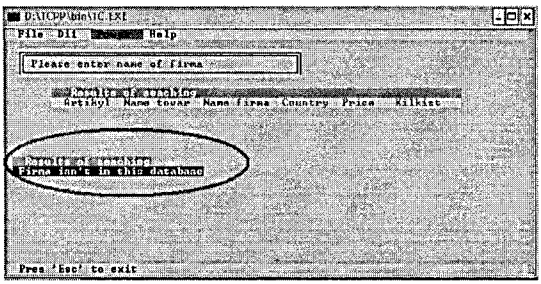


Рисунок 3.16 – Рисування підменю Firma

```
            getch();
            pch = 27;
            break;
case 2:
```

Якщо натиснути на Enter на першому підпункті меню, тобто на пункті Artikyl, з'являється вікно

```
Window(1,2,35,7,1,15,'n');
```

Для того, щоб попереднє вікно зникло

```
Window(3,3,45,5,7,0,'y');
```

Для того, щоб нарисувати вікно з певними координатами та з рамочкою, необхідно

```
gotoxy(5,4);
```

- переходимо до координат (5,4)

```
textcolor(15);
```

- встановлюємо білий колір тексту

```
cprintf("Please enter artikyl - ");
Window(8,8,69,8,7,7,'n');
```

Для того, щоб нарисувати вікно з певними координатами та без рамки необхідно використати такі оператори:

```
gotoxy(9,8);
```

- переходимо до координат (9,8)

```
textcolor(15);
```

- встановлюємо білий колір тексту

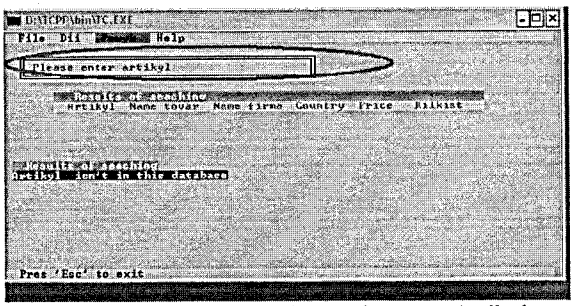


Рисунок 3.17 – Рисування підменю Artikyl

Переходимо до міток з певними координатами та пишемо назви складових

```
cprintf(" Artikyl ");
cprintf(" Name tovar ");
cprintf(" Name firma ");
cprintf(" Country ");
cprintf(" Price  ");
cprintf(" Kilkist ");
Window(8,7,69,7,6,6,'n');
gotoxy(11,7);
textcolor(15);
cprintf("Results of seaching");
```

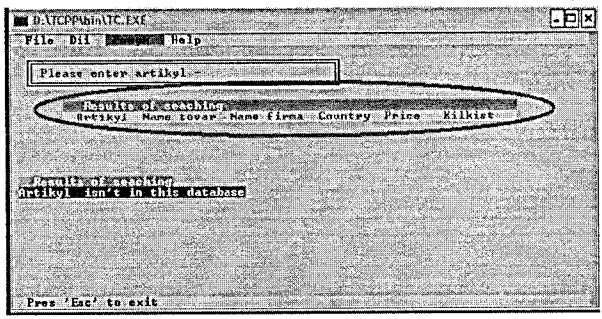


Рисунок 3.18 – Рисування підменю Artikyl

Якщо такого артикулу немає, то потрібно створити вікно, яке про це інформує, для цього

```
gotoxy(2,14);
```

- переходимо до координат (2,14)

```
textbackground(6);
```

- встановлюємо коричневий колір фону і пишемо:

```
cprintf("  Results of seaching");
gotoxy(2,15);
```

- переходимо до координат (2,15)

```
textbackground(RED);
```

- встановлюємо коричневий колір фону і пишемо:

```
cprintf("Artikyl  isn't in this database");
```

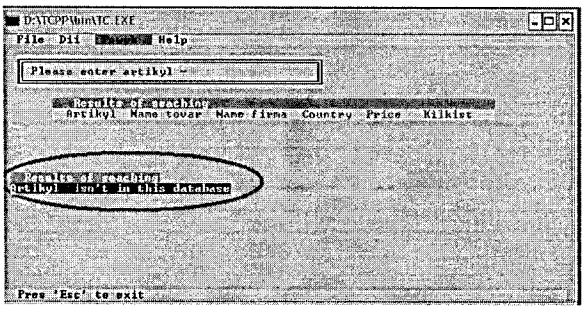


Рисунок 3.19 – Рисування підменю Artikyl

```
getch();
pch = 27;
break;
case 3:
```

Якщо натиснути на Enter на першому підпункті, тобто на Price, з'являється вікно:

```
Window(1,2,35,7,1,15,'n');
```

Для того, щоб попереднє вікно зникло, необхідно використати оператор

```
Window(3,3,45,7,7,0,'y');
```

Для того, щоб нарисувати вікно з певними координатами та з рамкою, необхідно використати такі оператори

```
gotoxy(5,4);
```

- переходимо до координат (5,4)

```
textcolor(15);
```

- встановлюємо білий колір шрифту і пишемо:

```
cprintf("Please enter artikyl - ");
gotoxy(5,5);
cprintf("Please enter min price - ");
gotoxy(5,6);
cprintf("Please enter max price - ");
 Window(8,10,69,10,7,7,'n');
```

Для того, щоб нарисувати вікно з певними координатами та без рамки, необхідно використати такі оператори:

```
gotoxy(9,10);
```

- переходимо до мітки з координатами (9,10)

```
textcolor(15);
```

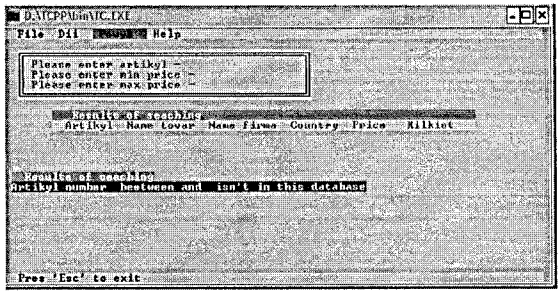


Рисунок 3.20 – Рисування підменю Price

- встановлюємо білий колір шрифту та пишемо:

```
cprintf(" Artikyl ");
cprintf(" Name tovar ");
cprintf(" Name firma ");
cprintf(" Country ");
cprintf(" Price  ");
cprintf(" Kilkist ");
Window(8,9,69,9,6,6,'n');
```

- рисуємо віконечко коричневого кольору

```
gotoxy(11,9);
```

- переходимо до мітки з координатами (11,9)

```
textcolor(15);
cprintf("Results of seaching");    }
```

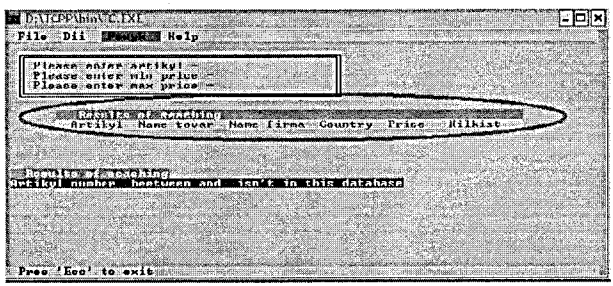


Рисунок 3.21 – Рисування підменю Price

Якщо такого артикулу з заданою ціною немає, то потрібно створити вікно, яке про це інформує

```
{ gotoxy(2,15);
```

```
                    textbackground(6);
                    cprintf("  Results of seaching");
                    gotoxy(2,16);
                     textbackground(RED);
                    cprintf("Artikyl number beetween and isn't
in this database");  }
```

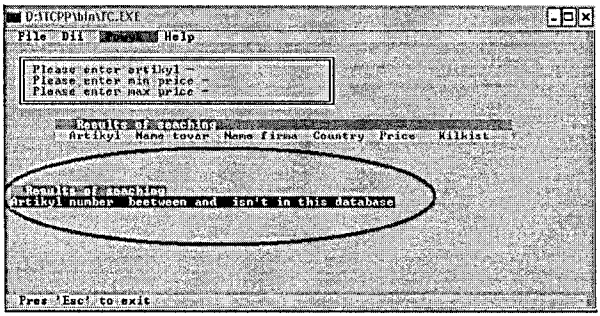


Рисунок 3.22 – Рисування підменю Price

```
                    getch();
                    pch = 27;
                    break;
                     }
                    break; // end 13
                     } // end switch(pch)
                    }while(pch!=27); //Esc
                    Window(1,2,75,24,1,0,'n');
                    }
  void podmenu4()
```

У четвертому підменю, підменю Help, рисуємо вікно

```
                    { int pch;
                    Window(22,2,30,5,7,0,'y');
```

Це вікно має рамку, яскраво-сірий фон та чорний колір тексту. Перейшовши до координат (23,4) та (23,3) пишемо назву пунктів підменю Avtor і Help

```
                    gotoxy(23,4); cprintf(" Avtor ");
                    textbackground(2);
                    gotoxy(23,3); cprintf(" Help  ");
                    do
                    {
                     pch=getch();
                     switch (pch)}
```

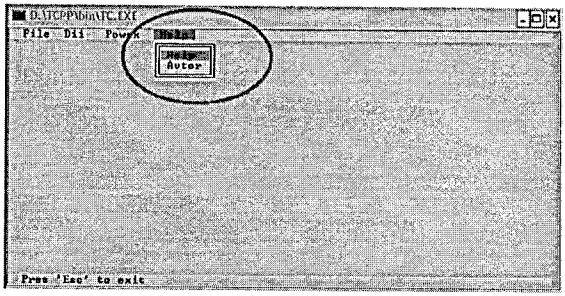


Рисунок 3.23 – Рисування підменю Help

```
case 80:
```

За допомогою клавіші ↓ переміщуємося по пунктах підменю

```
PMIndex++;
if (PMIndex==2)
```

Якщо натиснути на 2-му підпункті (Avtor), то фон яскраво-сірий на надпису Help перетворюється в фон зелений на активованому підпункті

```
{
textbackground(7); -
gotoxy(23,3); cprintf(" Help  "); -
textbackground(2);
gotoxy(23,4); cprintf(" Avtor ");
}
```

Якщо натиснути на першому підпункті (Help), то фон яскраво-сірий на надпису Avtor перетворюється в фон зелений на активованому підпункті

```
else {
 PMIndex=1;
 textbackground(2);
 gotoxy(23,3); cprintf(" Help  ");
 textbackground(7);
 gotoxy(23,4); cprintf(" Avtor ");
 }
break; //end 80 Down
```

```
case 72:
```

Аналогічні дії відбуваються, коли натиснути на клавішу зі стрілкою «вгору»

```
PMIndex--;
if (PMIndex==1)
{
textbackground(7);
gotoxy(23,4); cprintf(" Avtor ");
textbackground(2);
gotoxy(23,3); cprintf(" Help  ");
}
else {
PMIndex=2;
 textbackground(7);
```

```
                gotoxy(23,3);
                cprintf(" Help  ");
                textbackground(2);
                gotoxy(23,4);
                cprintf(" Avtor ");
                }
               break; //end 72  Up
  case 13:
```

Якщо натиснути на Enter на першому підпункті, тобто на Help, з'являється вікно

```
               if (PMIndex==1)
               {
               Window(1,2,35,7,1,15,'n');
```

Для того, щоб попереднє вікно зникло, необхідні такі оператори

```
               Window(8,4,55,12,7,0,'y');
```

Для того, щоб нарисувати вікно з певними координатами, з рамкою та з яскраво-сірим фоном і чорним кольором шрифта

```
               gotoxy(10,4);
```

- переходимо до координат (10,4)

```
               textbackground(6);
```

- встановлюємо коричневий фон

```
               textcolor(15);
```

- білий колір шрифту і пишемо:

```
               cprintf(" Help ");
               getch();
               pch = 27; }
```

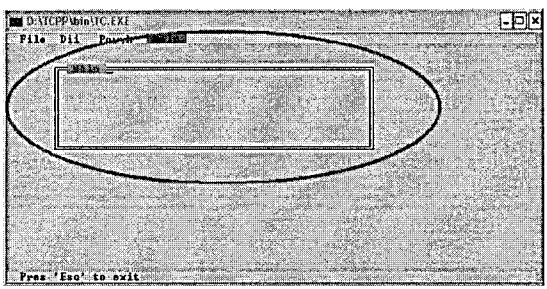


Рисунок 3.24 – Рисування меню Help

Якщо натиснути на другому підпункті, то з'явиться інше вікно:

```
               else
                { Window(1,2,35,7,1,15,'n');
                Window(8,4,38,11,7,0,'y');
```

Для того, щоб нарисувати вікно з певними координатами, з рамкою та з яскраво-сірим фоном, чорним кольором шрифту необхідно використати такі оператори

```
gotoxy(10,4);
```
- переходимо до координат (10,4)
```
textbackground(6);
```
- встановлюємо коричневий фон
```
textcolor(15);
```
- білий колір шрифту і пишемо:
```
cprintf(" About ");
getch();
pch = 27;
}
```

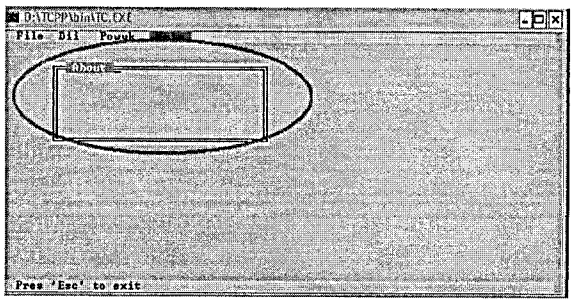


Рисунок 3.25 – Рисування меню About

```
break; //end 13
 } // end switch(pch)
}while(pch!=27);
Window(1,2,75,24,1,0,'n');
}
```

Дане вікно потрібне для того, щоб при натисканні кнопки Esc зникло попереднє вікно. Для цього вікно повинно мати такі ж самі координати, як і попереднє вікно, але фон синього кольору (як і фон вікна основного меню).

```
void main()
{
int ch;
MainMenu();
while(1)
{
ch=getch();
switch (ch){
```

По основних пунктах меню можна рухатись за допомогою клавіш «вправо» та «вліво». Для цього необхідно використати такі оператори.

Для організації руху вліво (код клавіші 77), необхідно використати оператори:

```
case 77:
```

```
                MIndex++;
                switch (MIndex)
                {

        case 2: (активне Dii)
                textcolor(0);
```
- встановлюємо чорний колір шрифту
```
                textbackground(7);
```
- встановлюємо яскраво-сірий фон
```
                gotoxy(2,1);
```
- переходимо до мітки з координатами (2,1)
```
                cprintf(" File ");
```
- пишемо File
```
                textcolor(0);
```
- встановлюємо чорний колір шрифту
```
                textbackground(2);
```
- встановлюємо зелений фон (зелений фон завжди при активації)
```
                cprintf(" Dii  ");
```
- пишемо Dii
```
                break;

        case 3: (активне Poshyk)
                textcolor(0);
```
- встановлюємо чорний колір шрифту
```
                textbackground(7);
```
- встановлюємо яскраво-сірий фон
```
                gotoxy(8,1);
```
- переходимо до мітки з координатами (8,1)
```
                cprintf(" Dii  ");
```
- пишемо Dii
```
                textcolor(0);
```
- встановлюємо чорний колір шрифту
```
                textbackground(2);
```
- встановлюємо зелений фон
```
                cprintf(" Powyk  ");
```
- пишемо Poshyk
```
                break;

        case 4: (активне Help )
                textcolor(0);
```
- встановлюємо чорний колір шрифту
```
                textbackground(7);
```
- встановлюємо яскраво-сірий фон
```
                gotoxy(14,1);
```
- переходимо до мітки з координатами (14,1)
```
                cprintf(" Powyk  ");
```
- пишемо Poshyk
```
                textcolor(0);
```

- встановлюємо чорний колір шрифту
```
                textbackground(2);
```
- встановлюємо зелений фон
```
                cprintf(" Help ");
```
- пишемо Help
```
                break;
        default: (активне File)
                MIndex=MinMIndex;
                textcolor(0);
```
- встановлюємо чорний колір шрифту
```
                textbackground(2);
```
- встановлюємо зелений фон
```
                gotoxy(2,1);
```
- переходимо до мітки з координатами (2,1)
```
                cprintf(" File ");
```
- пишемо File
```
                gotoxy(22,1);
```
- переходимо до мітки з координатами (22,1)
```
                        textcolor(0);
```
- встановлюємо чорний колір шрифту

```
                textbackground(7);
```
- встановлюємо яскраво-сірий фон
```
                cprintf(" Help ");
```
- пишемо Help

```
                break;//default
                }// end switch(MIndex)
                break;//end ch = 77
```

Для організації руху вправо (код клавіші 75) використовується програмний код, аналогічний описаному вище:

```
        case 75:
                        MIndex--;
                        switch (MIndex)
                        {
                case 1:
                        textcolor(0);
                        textbackground(7);
                        gotoxy(8,1);
                        cprintf(" Dii  ");

/*Edit*/
                        textcolor(0);
                        textbackground(2);
                        gotoxy(2,1);
                        cprintf(" File ");
```

```
                    break;

          case 2:
                    textcolor(0);
                    textbackground(7);
                    gotoxy(14,1);
                    cprintf(" Powyk  ");

/*Search*/
                    textcolor(0);
                    textbackground(2);
                    gotoxy(8,1);
                    cprintf(" Dii  ");

/*Edit*/
                    break;

          case 3:
                    textcolor(0);
                    textbackground(7);
                    gotoxy(22,1);
                    cprintf(" Help ");
                    textcolor(0);
                    textbackground(2);
                    gotoxy(14,1);
                    cprintf(" Powyk  ");

/*Search*/
                    break;
                    default:
                    MIndex=MaxMIndex;
                    textcolor(0);
                    textbackground(7);
                    gotoxy(2,1);
                    cprintf(" File ");
                    textcolor(0);
                    textbackground(2);
                    gotoxy(22,1);
                    cprintf(" Help ");
                    break;//default
                    }// end switch(MIndex)
                    break;//end 75

          case 13:  // if enter press
                    PMIndex = 1;
                    switch (MIndex)
                    {
                    case 1:  podmenu1(); break;
                     case 2:  podmenu2(); break;
                     case 3:  podmenu3(); break;
                     case 4:  podmenu4(); break;
                    }// end switch(MIndex) for Podmenu
```

```
break; // 80    Down
 case 27:  //   Esc
exit(1);
} //end switch(ch)
} //end while(1==1)
}// end main()
```

3.5 Контрольні запитання

1. Які стандартні бібліотеки необхідно підключити при створенні багатоієрархічного меню?
2. Які параметри повинна мати функція, що рисує вікна у меню?
3. Як нарисувати вікно з подвійною рамкою?
4. Як в поточному вікні здійснити відкриття ще одного вікна?
5. За допомогою яких функцій встановлюється колір тексту та колір фону?
6. Як створити рядок головного меню програми?
7. Як організувати переміщення клавішами по основних пунктах меню?
8. Як створити підменю?
9. Як реалізувати рух по підменю?
10. Як здійснити вихід з підменю?
11. Як здійснити вихід з головного меню?

3.6 Практикум з програмування

1. Написати фрагмент програми для створення вікна за заданими координатами.
2. Написати фрагмент програми для створення вікна з подвійною рамкою.
3. Написати фрагмент програми, в якому б задавалася кількість пунктів меню.
4. Розробити фрагмент програми, яка б створювала рядок горизонтального меню.
5. Написати фрагмент програми, який би дозволяв здійснювати рух по пунктах головного меню.
6. Написати фрагмент програми для створення вікна висхідного меню.
7. Написати фрагмент програми для здійснення руху по пунктах підменю.

4 ОСОБЛИВОСТІ ПОБУДОВИ ГРАФІКА ФУНКЦІЇ АЛГОРИТМІЧНОЮ МОВОЮ С

4.1 Загальні відомості

Побудова на екрані дисплея графіка функції – це цікава алгоритмічна задача, розв'язання якої починається з табулювання функції (тобто представлення функції за заданими точками). Метою такого подання даних є отримання рисунка потрібного розміру, що відображає тенденції зміни функції. Такий рисунок зручно використовувати у публікаціях, технічній документації, презентаціях тощо.

Професійний програмний продукт повинен не просто відображати графік функції $f(x)$ на відрізку $[a, b]$, а також дозволяти користувачу виконувати різні перетворення над графіком. До операцій перетворення можна віднести:

1) паралельний перенос осей координат;
2) зміна масштабів по осях координат;
3) зміна орієнтації осей координат;
4) перетворення абсолютних величин на графіку.

В цьому розділі посібника розглядається приклад програми мовою С для побудови графіка функції $f(x)$ заданого вигляду.

Програма починається з підключення стандартних бібліотек:

```
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
#include<math.h>
#include<graphics.h>
```

`void graf(float*,float*,int)` – це функція, за допомогою якої будемо рисувати графік функції двох змінних. В заголовку функції float* – вказівник на масив x, float* – вказівник на масив y, int – кількість точок в цьому масиві.

```
void graf(float*,float*,int);
```

`void Tablycia(float*,float*,int,int)` – це функція, за допомогою якої будуємо таблицю залежності функції x від y (float* – вказівник на масив x, float* – вказівник на масив y, int, int – номери елементів x і y, перший і останній).

```
void Tablycia(float*,float*,int,int);
```

`int MAX(float*,int)` – це функція для знаходження максимального числа x або y.

`int MIN(float*,int)` – це функція для знаходження мінімального числа x або y.

Головна програма:

```
 void main()
{
 clrscr();
```

`kol` – кількість точок масиву

```
int kol=1;
```

*x – вказівник на масив x, *y – вказівник на масив y, *xo* – початкова точка, *xn* – кінцева точка, *kr* – крок

```
float *x,*y, xo, xn, kr;
```

Вводимо початкові, кінцеві значення та крок:

```
printf("Xo="); scanf("%f",&xo);
printf("Xn="); scanf("%f",&xn);
printf("Krok="); scanf("%f",&kr);
```

Це виглядає так:

$$x_0 = -10$$
$$x_n = 10$$
$$Krok = 0.2$$

Умова `if(fmod(xn-xo,kr)==0)` перевіряє чи `(xn-xo)` з кроком kr=0

```
if(fmod(xn-xo,kr)==0)
```

якщо так, то `kol=(xn-xo)/kr;`

якщо ні, то `else kol=(xn-xo)/kr+1;`

Виділяємо пам'ять під два масиви: x і y

```
x=(float*)calloc(kol,sizeof(float));
y=(float*)calloc(kol,sizeof(float));
```

i – номер елемента масива

```
int i=0;
```

Вираз `while(i<kol)` означає: поки *i* менше за кількість точок

```
while(i<kol)
```

Вираз `x[i]=xo+i*kr;` означає: формуємо масив x

```
{
x[i]=xo+i*kr;
//y[i]=cos(x[i]);
//y[i]=sin(x[i])
//y[i]=tan(x[i]); функції, які необхідно побудувати
//y[i]=log(x[i]);
//y[i]=cos(x[i])*exp((-1)*x[i]);
```

`i++` – додаємо крок

```
i++;
}
```

`graf(x,y,kol);` – викликаємо функцію рисування графіка

```
graf(x,y,kol);
```

Звільняємо пам'ять x і y:

```
free(x);
free(y);
 closegraph();
}
```

Нижче на рисунках 4.1 – 4.3 показано, як за допомогою запропонованої програми будуть виглядати певні функції. Функція y[i] = cos(x[i]) буде мати вигляд, наведений на рисунку 4.1.

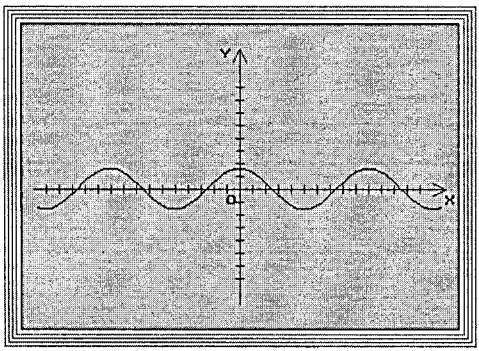


Рисунок 4.1 – Рисування графіка cos(x)

Функція y[i]=sin(x[i]) буде мати вигляд, наведений на рисунку 4.2.

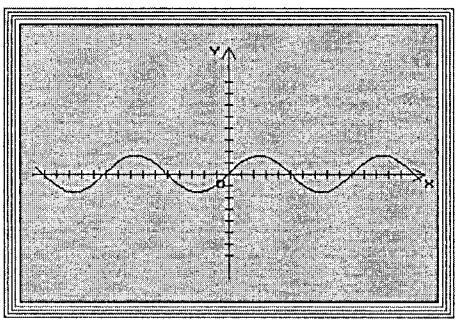


Рисунок 4.2 – Рисування графіка sin(x)

Функція y[i] = tan(x[i]) буде мати вигляд, наведений на рисунку 4.3.

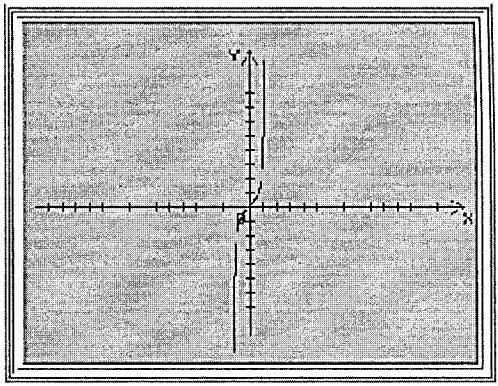


Рисунок 4.3 – Рисування графіка *tan*(x)

Заголовок функції `void graf()`, де `*xmas` – масив значень точок по осі x, `*ymas` – масив значень точок по осі y, `kill` – кількість точок, має вигляд:

```
void graf(float *xmas,float *ymas,int kill)
```

4.2 Ініціалізація графіки

Для ініціалізації графіки можна використати стандартний набір операторів:

```
{
   int gdriver = DETECT, gmode, errorcode;
   initgraph(&gdriver, &gmode, "");
   errorcode = graphresult();
   if (errorcode != grOk)
   {
       printf("Graphics error: %s\n",
              grapherrormsg(errorcode));
       printf("Press any key to halt:");
       getch();
       exit(1);    }
      cleardevice(); – очищуємо екран
      cleardevice();
      setcolor(4); – задаємо колір вікна (4 – червоний)
      setcolor(4);
      outtextxy(270,2,"Grafik funkcii"); – виводимо з координатами x=270 та y=2 напис "Grafik funkcii"
      outtextxy(270,2,"Grafik funkcii");
```

4.3 Введення даних

Для організації введення даних можна використати такі змінні:
q – змінна, якій присвоюється індекс масиву;
char **xmax* – максимальне x ,
**xmin* – мінімальне x,
**ymax* – максимальне y,
**ymin* – мінімальне y,
**kiln* – кількість точок;
int q;

```
float xmin, ymin, xmax, ymax;
```

Вираз `q=MIN(xmas,kill)` означає повернення номера найменшого елемента x.

```
xmin=xmas[q]; - присвоєння xmin значення найменшого x.
xmin=xmas[q];
```

Вираз `q=MIN(ymas,kill);` означає повернення номера найменшого елемента y.

`ymin=ymas[q];` – присвоєння `ymin` значення найменшого y.

```
ymin=ymas[q];
```

Вираз `q=MAX(xmas,kill);` означає повернення номера найбільшого елемента x.

`xmax=xmas[q];` – присвоєння `xmax` значення найбільшого x.

Вираз `q=MAX(ymas,kill);` означає повернення номера найбільшого елемента y.

`ymax=ymas[q];` – присвоєння `ymax` значення найбільшого y.

`setcolor(4);` – задаємо колір поточного вікна.

```
setcolor(4);
```

`setfillstyle (1,8);` – задаємо тип та колір вікна відповідно.

`bar (80,380,330,460);` – рисуємо прямокутник з координатами по x: (80-380) та по y (380-460).

`setcolor(14);` - задаємо колір поточного вікна

`setfillstyle (1,7);` - задаємо тип та колір поточного вікна відповідно.

Рисуємо прямокутник:

```
rectangle(76,376,334,464);
```

Далі аналогічно будуємо ще 3 прямокутники:

```
setcolor(10);
rectangle(73,373,337,467);
setcolor(11);
rectangle(70,370,340,470);
```

```
setcolor(13);
rectangle(67,367,343,473);
setcolor(14);
setfillstyle(1,7);
rectangle(80,380,330,460);
```

В результаті отримуємо вікно для рисування графіка у вигляді, наведеному на рисунку 4.4.

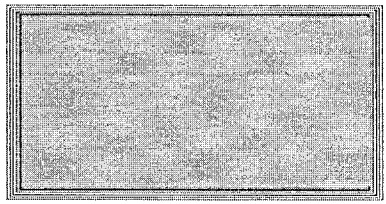


Рисунок 4.4 – Рисування вікна

Вираз `sprintf(x_min,"%4.1f",xmin);` означає, що перетворюємо змінну xmin типу float у змінну x_min типу char.

За допомогою функції `outtextxy(210,400,"X_min=");` виводимо по координатах поточного вікна (210,400) надпис *X_min=*.

За допомогою `outtextxy(260,400,"x_min=");` виводимо по координатах поточного вікна (260,400) значення мінімального x (x_*min=).*

В результаті отримуємо вікно для виведення на екран інформації про фунцію, графік якої будується, у вигляді, наведеному на рисунку 4.5.

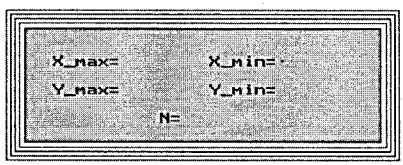


Рисунок 4.5 – Рисування вікна min-max без значень

Далі виконуємо аналогічні дії для визначення мінімальних значень x та y, а також для максимального y та визначимо N – кількості точок, необхідних для побудови графіка. Фрагмент програми, що дозволяє отримати таку інформацію, має вигляд:

```
sprintf(y_min,"%4.1f",ymin);
outtextxy(210,420,"Y_min=");
outtextxy(260,420,y_min);
sprintf(x_max,"%4.1f",xmax);
outtextxy(100,400,"X_max=");
```

```
outtextxy(150,400,x_max);
sprintf(y_max,"%4.1f",ymax);
outtextxy(100,420,"Y_max=");
outtextxy(150,420,y_max);
sprintf(kiln,"%5d",kill);
outtextxy(175,440,"N=");
outtextxy(175,440,kiln);
```

В результаті отримуємо вікно з інформацією про мінімальні та максимальні значення функції, що досліджується, у вигляді, показаному на рисунку 4.6.

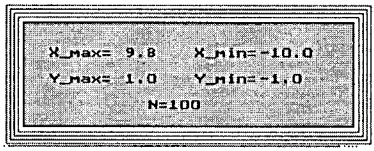


Рисунок 4.6 – Рисування вікна min-max зі значеннями

4.4 Алгоритм побудови графіка табличної функції

*//**************VIKNO GRAFIKA********************

1. Задаємо стиль функцією `setfillstyle (1,8)`, тобто поточний тип ліній (1 – суцільний) та колір (8 – темно-синій).

2. Рисуємо вікно з координатами по x (40,380) та y (80,320)

```
bar (40,80,380,320);
```

3. Задаємо поточний колір прямокутника (рамки) – 14 (жовтий) оператором setcolor(14);

4. Задаємо стиль функцією `setfillstyle (1,7);`, тобто поточний тип ліній (1 – суцільний) та колір (7 – сірий).

5. Рисуємо прямокутник (рамку) з координатами по x (40,380) та y(80,320) за допомогою функції rectangle (40,80,380,320).

6. Виконуємо аналогічні операції, тільки з іншими координатами та різними кольорами (12 – яскраво-червоний, 10 – яскраво-зелений, 11 – яскраво-блакитний) за допомогою операторів:

```
setcolor(12);
rectangle(27,67,393,333);
setcolor(10);
rectangle(30,70,390,330);
setcolor(11);
rectangle(33,73,387,327);
setcolor(13);
rectangle(36,76,384,324);
}
```

7. Встановлюємо поточний колір (в даному випадку – це колір осей та розмітки осей) оператором setcolor (YELLOW).

8. За допомогою функції `line(x1,y1,x2,y2);` будуємо осі, стрілочки, та розмітку осей, де x1, y1 – початкові координати по x і y відповідно; x2, y2 – кінцеві координати по x і y відповідно. Для цього можна використати такий набір операторів:

```
//вертикальна ось
line(210,100,210,300);

//стрілочка
line(210,100,205,110);
line(215,110,210,100);

//горизонтальна ось
line(50,210,370,210);

//стрілочка
line(370,210,360,205);
line(360,215,370,210);
```

В результаті отримуємо вікно для рисування графіка з осями *x* та *y* у вигляді, наведеному на рисунку 4.7.

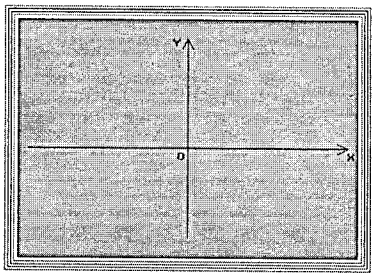


Рисунок 4.7 – Рисування осей x і y

9. Для розмітки осей можна використати такий набір операторів:

```
//вертикальна розмітка осей
{setcolor(14);
line(207,130,213,130);
line(207,140,213,140);
line(207,150,213,150);
line(207,160,213,160);
line(207,170,213,170);
line(207,180,213,180);
```

```
line(207,190,213,190);
line(207,200,213,200);
line(207,220,213,220);
line(207,220,213,220);
line(207,230,213,230);
line(207,240,213,240);
line(207,250,213,250);
line(207,260,213,260);
line(207,270,213,270);
line(207,280,213,280);
```

//вертикальна розмітка осей

```
setcolor(14);
line(60,207,60,213);
line(70,207,70,213);
line(80,207,80,213);
line(90,207,90,213);
line(100,207,100,213);
line(110,207,110,213);
line(120,207,120,213);
line(130,207,130,213);
line(140,207,140,213);
line(150,207,150,213);
line(160,207,160,213);
line(170,207,170,213);
line(180,207,180,213);
line(190,207,190,213);
line(200,207,200,213);
line(210,207,210,213);
line(220,207,220,213);
line(230,207,230,213);
line(240,207,240,213);
line(250,207,250,213);
line(260,207,260,213);
line(270,207,270,213);
line(280,207,280,213);
line(290,207,290,213);
line(300,207,300,213);
line(310,207,310,213);
line(320,207,320,213);
line(330,207,330,213);
line(340,207,340,213);
line(350,207,350,213);
}
```

10. Встановлюємо жовтий колір напису оператором setcolor(14); та виводимо "X" з координатами (370,215) функцією outtextxy(370,215,"X"); виводимо "Y" з координатами (195,100) функцією outtextxy(195,100,"Y"). Визначаємо точку перетину осей функціями

```
outtextxy(200,215,"0");
setfillstyle(1,4);
```

В результаті отримуємо вікно для рисування графіка з розміткою осей координат у вигляді, показаному на рисунку 4.8.

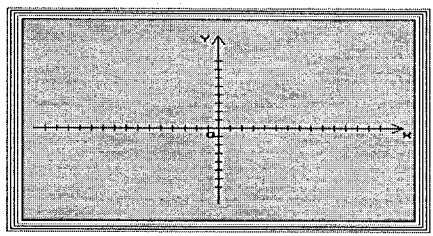


Рисунок 4.8 – Рисування осей координат з розміткою

11. Встановлюємо поточний білий колір функцією setcolor(15).

12. Описуємо змінні: *kx, ky* – коефіцієнти, з якими стискується графік по x і y відповідно

```
float kx,ky,mx,my;
if (fabs(xmin)>xmax)  mx=fabs(xmin); else mx=xmax;
if (fabs(ymin)>ymax)  my=fabs(ymin); else my=ymax;
```

(400-80) означає: рисуємо графік з координати 80 до координати 400 по x

```
kx=(400-80)/(2*mx); //коефіцієнти 20
```

(400-180) означає: рисуємо графік з координати 180 до координати 400 по y

```
ky=(400-180)/(2*my);
```

13. Визначення коефіцієнту стиску. Умову `if (kx<ky) ky=kx;` потрібно прочитати так: якщо коефіцієнт стиску по x менший за коефіцієнт стиску по y, то коефіцієнту стиску по y присвоюємо значення коефіцієнта стиску по x.

`if (kx<ky) ky=kx;` якщо ні, то коефіцієнту стиску по x присвоюємо коефіцієнт стиску по y

```
else kx=ky;
```

14. Рисування графіку. Задаємо цикл:

```
for(int j=1;j<kill-1;j++) //рисування графіка
```

для побудови графіка за допомогою line

```
line(kx*xmas[j]+210,210-ky*ymas[j],kx*xmas[j+1]+210,
210-ky*ymas[j+1]);
```

В результаті отримуємо вікно для рисування графіка у вигляді, наведеному на рисунку 4.9.

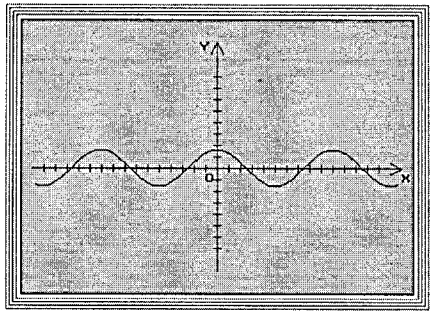


Рисунок 4.9 – Рисування косинусоїди

4.5 Створення вікна таблиці для виведення даних

*//**********VIKNO TABLICI**********

1. Ініціалізуємо ще одну змінну `int ch` для роботи з клавіатурою

```
int ch;
```

2. Присвоюємо змінній *i* значення 35 – це кількість пар точок x і y

```
int i=35;
```

`if(kill<35)` означає: якщо кількість точок менша за 35, то виводимо на екран таблицю залежності y від x; `Tablycia(xmas,ymas,0,kill)`: xmas – масив x-ів, ymas – масив y-ів, 0 – це початкова точка, kill – всі пари точок.

```
if(kill<35)
  Tablycia(xmas,ymas,0,kill);
```

якщо кількість таких пар більша за 35, то виводимо табуляцію, але тільки з кількістю точок, що дорівнює 35.

```
else {
  Tablycia(xmas,ymas,0,35);
```

3. Нескінченний цикл необхідний для того, щоб можна було рухатись по таблиці табуляції нескінченно):

```
while(1)
 {
```

Вираз `ch=getch()` означає, що йде зчитування коду клавіші.

Вираз `if(ch==27)` перевіряє чи дорівнює змінна коду 27, а це код клавіші Esc. Якщо так, то `exit(0)`, тобто виконується вихід з програми.

```
if(ch==27) exit(0);
```

Вираз if(ch==80) перевіряє чи дорівнює змінна коду 80, а це код клавіші ↓.

```
if(ch==80) {  //донизу
```

4. Використовуємо умову

`if(kill-i<=35)`, тобто якщо кількість точок менша за 35, то виводимо на екран

```
  Tablycia(xmas,ymas,i,kill);
```

якщо не менша, то виводимо табуляцію з кількістю точок i+35. Інакше використовуємо такі оператори:

```
else
{
  Tablycia(xmas,ymas,i,i+35); i+=35;
}
}
```

5. Далі виконуємо такі ж операції, але уже для того, щоб рухатись по таблиці вгору*/

6. Вираз `if(ch==72)` перевіряє чи дорівнює змінна коду 80, а це код ↑.

```
if(ch==72) {     //Vverh
if(i<35&&i>=0)
{Tablycia(xmas,ymas,0,35); i=0;}
else
{Tablycia(xmas,ymas,i-35,i); i-=35; }
}
}
 }
 }
```

Функція Tablycia() має такий заголовок:

```
void Tablycia(float *x,float *y,int p,int k),
```

де

`float *x` – масив значень точок по осі x,

`float *y` – масив значень точок по осі y,

`int p` – перша точка, з якої починаємо таблицю табуляції,

`int k` – кінцева точка табуляції.

7. За аналогією як будувалося вікно для графіка, встановлюємо колір фону, тексту, стиль, зокрема тип ліній, будуємо вікна, за допомогою таких операторів:

```
setcolor(15);
setfillstyle(1,8);
rectangle(440,50,600,430);
floodfill(455,70,15);
{
setcolor(14);
rectangle(440,50,600,430);
setcolor(12);
rectangle(436,46,604,434);
setcolor(10);
rectangle(433,43,607,437);
setcolor(11);
rectangle(430,40,610,440);
setcolor(13);
rectangle(427,37,613,443);
}
```

В результаті отримуємо вікно для таблиці значень функції у вигляді, наведеному на рисунку 4.10.

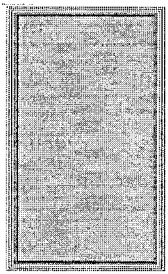


Рисунок 4.10 – Рисування вікна, де буде розміщена табуляція

8. Для рисування таблиці в цьому вікні використовуємо такі оператори:

```
setcolor(14);
line(520,50,520,430);
line(440,70,600,70);
setcolor(14);
outtextxy(480,60,"X");
outtextxy(550,60,"Y");
```

В результаті отримуємо вікно для виведення таблиці значень функції у вигляді, наведеному на рисунку 4.11.

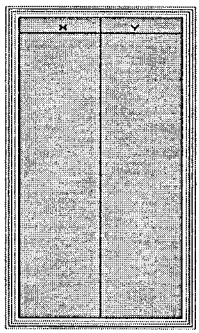


Рисунок 4.11 – Рисування вікна, буде розміщена табуляція

9. Для виведення числових значень функції у вигляді таблиці в графічне вікно виконуємо перетворення їх в рядок. Для цього описуємо змінні char *vr1,*vr2, де *vr1 – тимчасова змінна, призначена для перетворення змінних (по x) з float в int; *vr2 – тимчасова змінна, призначена для перетворення змінних (по y) з float в int та використовуємо оператори:

```
char *vr1,*vr2;
int ytab,j;
```

```
for(ytab=75,j=p;j<k;ytab+=10,j++)
  {
  sprintf(vr1,"%4.1f",x[j]);
  outtextxy(465,ytab,vr1);
```

перетворюємо змінні float в int

```
  sprintf(vr2,"%4.1f",y[j]);
  outtextxy(540,ytab,vr2);
  }
}
```

В результаті отримуємо вікно з таблицею значень функції у вигляді, наведеному на рисунку 4.12.

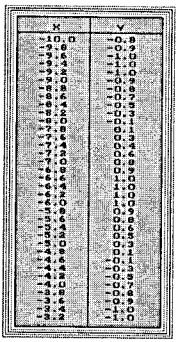


Рисунок 4.12 – Рисування вікна табуляції функції

10. Наступна функція `int MAX()` призначена для пошуку максимального значення в масивах `x` і `y`

```
int MAX(float *m,int n)
```

де*m – масив x-ів або y-ів, int n – кількість елементів в масиві.

```
{ float z; int imax;
```

Тимчасовій змінній `z` присвоюємо початковий номер елемента у масиві, а змінній `imax` – мінімальне значення.

```
z=m[0]; imax=0;
```

11. Задаємо цикл для пошуку максимального значення в масиві:

```
for(int i=1;i<n;i++)
{
```

Якщо поточний номер елемента більший за початковий, тобто

```
if(m[i]>=z)
{
```

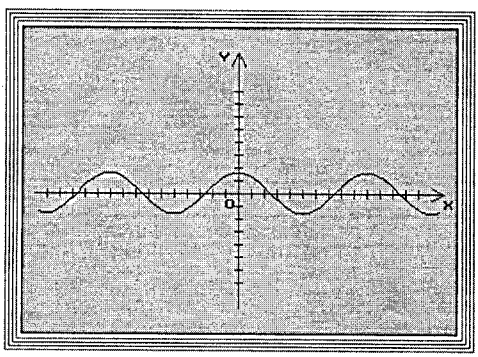
то початковому значенню присвоюємо нове

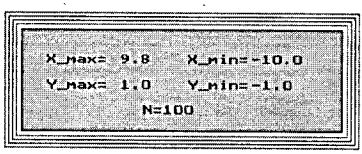
```
z=m[i];
```
номеру максимального елемента присвоюємо теж новий індекс
```
imax=i;
}
}
```
Повертаємо `imax:` `return imax;`
```
}
```

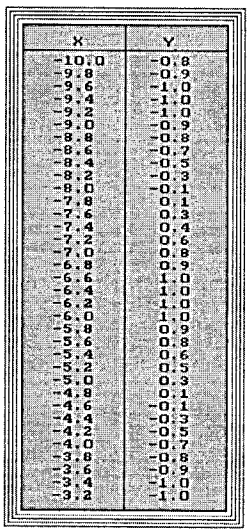
12. Створюємо функцію `int MIN(float *m,int n)` для знаходження мінімального значення. Принцип його знаходження такий самий, як для максимуму. Для цього використовуємо оператори:

```
int MIN(float *m,int n)
{ float z; int imin;
z=m[0]; imin=0;
for(int i=1;i<n;i++)
{
if(m[i]<=z)
 {
z=m[i]; imin=i;
}
}
return imin;
}
```

Отже, для функції y[i] = cos(x[i]) вікно графіка має вигляд:







X	Y
-10.0	-0.8
-9.8	-0.9
-9.6	-1.0
-9.4	-1.0
-9.2	-1.0
-9.0	-0.9
-8.8	-0.8
-8.6	-0.7
-8.4	-0.5
-8.2	-0.3
-8.0	-0.1
-7.8	0.1
-7.6	0.3
-7.4	0.4
-7.2	0.6
-7.0	0.8
-6.8	0.9
-6.6	1.0
-6.4	1.0
-6.2	1.0
-6.0	1.0
-5.8	0.9
-5.6	0.8
-5.4	0.6
-5.2	0.5
-5.0	0.3
-4.8	0.1
-4.6	-0.1
-4.4	-0.3
-4.2	-0.5
-4.0	-0.7
-3.8	-0.8
-3.6	-0.9
-3.4	-1.0
-3.2	-1.0

Рисунок 4.13 – Загальний вигляд вікна для побудови графіка функції

4.6 Контрольні запитання

1. Які стандартні бібліотеки необхідно підключити при побудові графіка функції в С?
2. Як реалізувати функцію, що рисує графік?
3. Як реалізувати функцію, що будує таблицю залежності x від y?
4. Як перевірити, чи крок табулювання функції більше нуля?
5. Графіки яких функцій можна побудувати засобами мови програмування С?
6. Як реалізувати цикл рисування графіка функції?
7. Як перетворити змінні одного типу даних в інший? Для чого це може бути потрібно?
8. Як здійснити виведення таблиці значень?
9. Як побудувати осі графіка?
10. Як на осях графіка відобразити шкалу значень?
11. Як вивести графік табличної функції в окремому вікні?

4.7 Практикум з програмування

1. Розробити функцію, що перетворювала б дані типу float в дані типу int.
2. Написати фрагмент програми для виведення таблиці значень масивів X та Y в окремому вікні.
3. Написати фрагмент програми для виведення max_x, max_y, min_x, min_y в окремому вікні.
4. Написати фрагмент програми для побудови осей графіка табличної функції та розбиття його на поділки.
5. Як промасштабувати табличну функцію Y(x)?
6. Написати фрагмент програми для масштабування функції y=cos(x).
7. Написати фрагмент програми для масштабування функції y=sin(x).
8. Написати фрагмент програми для масштабування функції y=tan(x).
9. Написати фрагмент програми для масштабування функції y=log(x).

5 ОСНОВНІ ПРИНЦИПИ РОБОТИ З ПОТОКАМИ ДАНИХ

5.1 Загальні відомості

Бібліотека мови С підтримує три рівня введення – виведення:

- потоковий;
- введення – виведення низького рівня;
- консольне введення – виведення (спеціалізований обмін даними з дисплеєм та портами введення – виведення).

На рівні потокового введення – виведення обмін даними виконується побайтно, що зручно як для файлів на диску так і для пристроїв побайтного обміну (принтер, дисплей).

Функції бібліотеки введення – виведення мови С підтримують обмін даними з файлами на рівні потоку та дозволяють обробляти дані різних розмірів та форматів за допомогою буфера, тому потік – це файл з засобами буферизації. Файл – це іменована область на диску, призначена для зберігання текстових, символьних, аудіо-, відеоданих. Буфер – це частина оперативної пам'яті для тимчасового зберігання даних в процесі обміну.

В бібліотеці введення – виведення мови С існує ряд функцій, що підтримують роботу з потоками, такі як створення, введення-виведення, відриття-закриття, управління буферізацією, для використання яких необхідно підключення файлу stdio.h (#include<stdio.h>). Однак використання цих функцій залежить від типу файлу (текстовий або бінарний), зв'язаного з потоком. В мові С розглядають два типи файлів: текстові і бінарні. Текстовий файл – це набір рядків заданого розміру, кожен з яких закінчується кодом #13 (CR) і кодом #10 (LF). Бінарний файл – це набір компонент заданого розміру, кожна з яких є набором байтів.

5.2 Читання і запис текстових файлів

Введення – виведення окремих символів

Одним з найбільш ефективних способів здійснення введення – виведення в потік одного символу є використання бібліотечних функцій `getchar()` і `putchar()`.

`getchar()` – функція, що здійснює введення в потік одного символу. При звертанні вона повертає у функцію, яка її викликала, один введений символ. При читанні з потоку функцією `getchar()` може бути досягнуто кінця файлу. В цьому випадку операційна система у відповідь на спробу читання символу передає функції `getchar ()` значення `EOF (End of File)`.

`putchar()` – функція, що виводить в потік один символ, при цьому також повертає у функцію, яка її викликала, щойно введений символ.

Введення – виведення рядків

Бібліотека мови С для обміну рядками (масиву символів) через стандартні потоки містить функції `gets()` і `puts()`, які зручно використовувати при створенні діалогових систем. Обидві функції мають тільки один аргумент – вказівник s на масив символів. Якщо рядок прочитаний нормально, функція `gets()` повертає адресу того масиву *s*, в який відбувається введення рядка. В разі помилки повертається `NULL`.

Функція `puts()` у випадку успішного завершення повертає останній символ, який завжди є символом `'\n'`. В разі помилки повертається `EOF`.

Приклад 5.1. Використання функцій `gets()` і `puts()`

```
#include<stdio.h>
char str1[ ] = " ";
int main ()
  {
    char name[80];
    puts(str1);
    gets(name);
    return 0;
  }
```

Будь-який рядок символів в мові С повинен закінчуватись нуль-символом `'\0'`. В останній елемент масиву `str1` нуль-символ буде записаний автоматично під час трансляції при ініціалізації масиву.

Форматне введення – виведення

Для роботи з стандартними потоками в режимі форматного введення – виведення визначені дві функції:

`printf()` – форматне виведення, перетворює дані з внутрішнього подання в символьний вигляд у відповідності з форматним рядком і виводить їх в потік. Дані, які перетворюються і виводяться, задаються як аргументи функції `printf()`;

`scanf()` – форматне введення з вхідного потоку. Читає послідовності кодів символів з вхідного потоку та інтерпретує їх як цілі числа, дійсні числа, одиничні символи, рядки.

Для роботи з файлами задіяні такі функції:

`fgetc(), getc()` – введення (читання) одного символу з файлу;

`fputc(), putc()` – запис одного символу в файл;

`fprintf()` – форматне виведення у файл;

`fscanf()` – форматне введення (читання) з файлу;

`fgets()` – введення (читання) рядка з файлу;

`fputs()` – запис рядка в файл.

Функція `fputs(const char *s, FILE *stream)` записує обмежений символом `'\0'` рядок у потік, визначений вказівником `stream`, і повертає невід'ємне число. Символ `'\0'` в файл не переноситься, і символ `'\n'` не записується в кінці рядка замість `'\0'`.

Функція `fgets(char *s, int n, FILE *stream)` – читає з визначеного вказівником `stream` потоку не більше (n-1) символів і записує їх в рядок, на який вказує вказівник `s`. Функція закінчує читання, як тільки прочитає (n-1) символів чи зустріне символ нового рядка `'\n'`, який переноситься в рядок `s`. Додатково в кінець кожного рядка записується ознака кінця рядка `'\0'`.

У випадку успішного завершення функція повертає вказівник `s`. При помилці чи при досягненні кінця файлу, при умові, що із файлу не прочитаний жоден символ, повертається значення `NULL`. В цьому випадку вміст масиву, який адресується вказівником `s`, залишається без змін. На відміну від `fgets()` функція `gets()` відкидає символ `'\n'`.

Перед тим, як працювати з потоком, його необхідно ініціалізувати. При цьому потік зв'язується у виконуваній програмі зі структурою визначеного типу FILE. Визначення структурного типу FILE знаходиться в заголовному файлі **stdio.h**. В структурі FILE вміщуються компоненти, за допомогою яких ведеться робота з потоком: вказівник на буфер, вказівник поточної позиції в потоці та ін.

При відкритті потоку в програму повертається вказівник на потік, що є вказівником на об'єкт структурного типу FILE. Цей вказівник ідентифікує потік у всіх наступних операціях.

Вказівник на потік, наприклад `fp`, повинен бути описаний в програмі таким чином:

```
#include<stdio.h>
FILE *fp;
```

Вказівник на потік набуває значення в результаті виконання функції відкриття потоку:

```
fp = fopen (ім'я_файлу, режим_відкриття);
```

Параметри функції `fopen()` є вказівниками на масиви символів, які вміщують відповідно ім'я файлу, зв'язаного з потоком, і рядок режимів відкриття. Однак ці параметри можуть задаватись і безпосередньо у вигляді рядка при виклику функції відкриття файлу:

```
fp = fopen ("file.txt","r");
```

де `file.txt` – ім'я деякого файлу, зв'язаного з потоком;

`r` – позначення одного з режимів роботи з файлом (тип доступу до потоку).

Після закінчення роботи з файлами рекомендується закрити їх явно. Для цього використовується функція `fclose` (вказівник на потік).

Відкритий файл можна відкрити повторно (наприклад, для зміни режиму роботи з ним) тільки після того, як файл буде закритий з допомогою функції `fclose()`.

5.3 Читання і запис двійкових файлів

Текстові файли, незважаючи на своє широке поширення, є тільки одним з видів файлів, які можна зберігати на диску. Двійкові файли знайшли більш широке застосування в професійних програмах, тому що:

1) одна з переваг збереження даних у двійкових форматах – швидкість (тобто немає необхідності перетворювати дані при передачі з диска в пам'ять і навпаки);

2) інша перевага – пам'ять. Так, наприклад, змінна типу double займає 4 байти, а в текстовій формі набагато більше. Звичайно двійкові файли за розміром менші текстових і в більшості випадків програми обробляють їх швидше, ніж текстові.

Відкриття двійкових фалів

Для відкриття двійкового файлу необхідно:

1) визначити файлову змінну `File *fp`;

2) безпосереднє відкриття виконується функцією `fopen()` по всіх режимах якої додається буква "b" або "rb" – читання, "wb" – записування, "ab" – дозаписування, "r+b" – читання та записування в існуючому файлі, "w+b" – читання та записування в існуючому або новому файлі, "a+b" – дозаписування і читання з існуючого файлу.

Наприклад, оператор `fp=fopen("test.txt","r+b")` відкриває файл для читання і записування в двійковому режимі. При відкритті можливі помилки (тобто програмні переривання), які можна передбачити за допомогою фрагмента програми:

```
File *out;
out = fopen (filename, "wb");
if ( ! out)
  {
     puts ("Can't create file");
     exit (1);
  }
```

Існує два способи читання і записування двійкових файлів: послідовний і прямий (довільний).

5.4 Файли з послідовним доступом

Послідовна обробка корисна для швидкого запам'ятовування значень у файлах і для роботи з файловими даними, як з потоком байтів (аналогічно запису на магнітну стрічку).

Запис даних у файл з послідовним доступом

Приклад 5.2. Написати програму записування даних у послідовний файл

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
File *outf;
int i;

outf = fopen ("int.dat", "wb");

if ( ! outf)
{
 puts ("Can't create file");
 exit (1);
}

puts ("Zapis into file");
for (i=0; i<100; i++)
fwrite(&i, sizeof(int),1, outf);
fclose (outf);
return 0;
}
```

Для записування даних у двійковий файл використовується функція `fwrite()`, що містить чотири параметри:

1) адресу змінної чи масиву змінних, з якого байти копіюються на диск;

2) число байтів в одній змінній;

3) число записуваних елементів: 1 для одного значення, або інше позитивне ціле число записуваних елементів масиву;

4) змінна файлового типу `File *`, відкритого в двійковому режимі.

Для записування у відкритий файл на диску масиву у 100 значень можна скористатися фрагментом програми:

```
int array[100];
fwrite(&array, sizeof(int),100,outf);
```

Читання даних з файлу послідовного доступу за допомогою функції fread()

Приклад 5.3. Написати програму для послідовного читання двійкових значень з файлу **int.dat**

```
#include <stdio.h>
#include <stdlib.h>
 int main()
{
File *inpf;
int I, val;
inpf = fopen ("int.dat", "rb");

if ( ! inpf)
{
  puts ("Can't create file");
  exit (1);
}

for (i=0; i<100; i++)
{
  fread(&val, sizeof(int),1, inpf);
  printf ("%d",val);
}

return 0;
}
```

Функція `fread()` вимагає тих же аргументів, що і функція `fwrite()`. Перший аргумент функції є адресою приймача, у який ця функція повинна скопіювати байти з диска, тому розмір цієї змінної достатній для запам'ятовування необхідної кількості байтів.

Для переміщення всього масиву зі 100 цілочислових значень з диска у файл можна використовувати фрагмент програми:

```
int array[100];
fread(&array, sizeof(int),1, inpf);
```

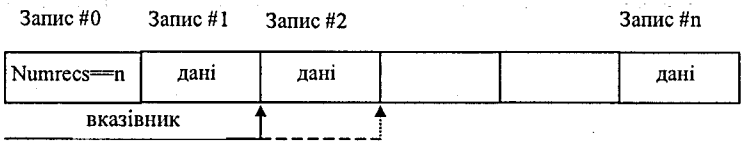
це буде самим швидким способом читання значень з диска в пам'ять.

5.5 Файли з довільним доступом

Особливістю таких файлів є те, що:

1) файл складається з елементів (записів) однакової довжини (розміру). Кожен запис у файлі має номер;

2) уздовж файлу переміщається вказівник, положення якого можна позиціонувати програмно за допомогою функції `fseek()`.



3) функція `fseek()` вимагає три аргументи:

- змінна потоку типу `File*`, відкритого для двійкового доступу;
- значення зміщення вказівника щодо заданої (чи поточної) позиції;
- один із трьох компонентів:

`SEEK_SET` – зсув вказівника на число байтів, на які внутрішній вказівник пересувається вперед від початку файлу;

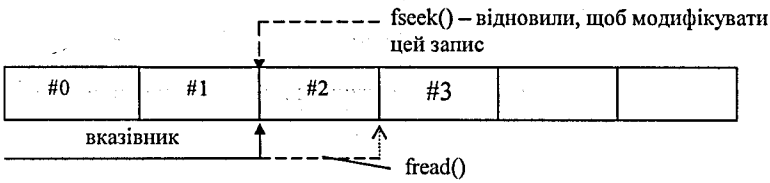
`SEEK_CUR` – зсув на число байтів вперед чи назад від поточної позиції вказівника, в залежності від знака константи;

`SEEK_END` – зсув на число байтів від кінця файлу в напрямку до початку.

Приклад 5.4. Написати програму для читання 11-го запису у файлі int.dat

```
#include <stdio.h>
#include <stdlib.h>
 int main()
{
File *inpf;
int val;
inpf = fopen ("int.dat", "rb");
if ( ! inpf)
{
  puts ("Can't create file");
  exit (1);
}
fseek (inpf, 10*sizeof(int),SEEK_SET);
fread(&val, sizeof(int),1, inpf);
printf (" Record #10 == %d",val);
fclose(inpf);
return 0;
}
```

Другий аргумент у `fseek()` – добуток розміру в байтах змінної типу int на номер запису (10) шуканого значення.



Приклад 5.5. За допомогою функції fseek() і fwrite() модифікувати 11-й запис у файлі **int.dat**

```
#include <stdio.h>
#include <stdlib.h>
 int main()
{
File *outf;
int val=99;              /*нове значення запису */
outf = fopen ("int.dat", "r+b");

if ( ! outf)
{
  puts ("Can't create file");
  exit (1);
}

printf("Writing %d to record #10",val);

/* модифікація 10-го запису */
fseek (outf, 10*sizeof(int),SEEK_SET);
fwrite(&val, sizeof(int),1, outf);
fclose(outf);
return 0;
}
```

5.6 Передача файлів між комп'ютерами

Сьогодні багато організацій мають у своєму розпорядженні кілька комп'ютерів, причому часто ці комп'ютери виявляються різних типів або різних моделей, а також мають несумісні формати дисків. При використанні різних комп'ютерів більша перевага може бути досягнута при з'єднанні комп'ютерів через їхні послідовні порти з метою спільного використання ними інформації і/або програм. У багатьох випадках створення програм, що забезпечують обмін файлами для таких комп'ютерів через їхні послідовні порти, є проблематичним.

Однак існує досить швидкодійна й ефективна програма передачі файлів. Ця програма має ряд значних переваг: вона працює з будь-якими типами файлів на всіх типах комп'ютерів, що природно відрізняються один від одного своєю продуктивністю і, найголовніше, не використовують апаратного підтвердження зв'язку. У додаток до всього, програма може працювати навіть тоді, коли апаратне підтвердження зв'язку в принципі неможливе і непотрібне.

Підпрограми передачі файлів виконують свої функції, використовуючи програмне підтвердження зв'язку, і функціонують фактично в різних середовищах. Однак для рішення глобальної проблеми краще пожертвувати продуктивністю, збільшивши надійність системи.

Програмне підтвердження зв'язку

Коли апаратне підтвердження зв'язку неможливе або непотрібне, єдиним способом, що дозволяє уникнути помилок переповнення регістра, що не можуть бути зареєстрованими безпосередньо під час передачі даних по каналу зв'язку, є введення програмного підтвердження зв'язку. Програмне підтвердження зв'язку працює в такий спосіб: комп'ютер-джерело посилає перший байт і переходить у стан чекання повернення від комп'ютера-приймача звітувального байта (байта, що підтверджує прийняття попереднього повідомлення). При одержанні звітувального байта, комп'ютер-джерело посилає наступний байт і знову переходить у стан чекання звітувального байта від комп'ютера-приймача.

Цей процес продовжується доти, доки весь файл повністю не буде переданий. Нижче наведені в термінах псевдо-С процедури передачі і прийому даних.

```
send()
{
  while ( є байти для передачі ){
      send( байт );
       wait();
  }
}

receive()
{
  do {
     receive_byte();
     send( звітувальний байт );
  } while( поки всі байти не зчитані );
}
```

При цьому підході передача даних не викликає ніколи переповнення регістра в порту-приймачі незалежно від того, наскільки велика різниця у швидкості виконання операцій комп'ютерів, між якими встановлений зв'язок.

При цьому типі підтвердження зв'язку є лише один недолік – швидкість передачі даних падає вдвічі в порівнянні з теоретично можливою. Це пояснюється тим, що при передачі одного байта інформації фактично відбувається передача двох байтів (згадайте про звітуючий байт).

Перекачування файлу

Першою необхідною підпрограмою є функція, що забезпечує передачу файлу через послідовний порт. У загальному випадку ця функція повинна відкрити файл, що буде переданий на інший комп'ютер, підрахувати його довжину, передати в порт-приймач довжину переданого файлу і, зрештою,

перекачати сам файл. Функція send_file(), представлена нижче, призначена для розв'язання саме таких задач.

```
/* перекачка специфікованого файлу */
void send_file(fname)
char *fname;
{
  FILE *fp;
  char ch;
  union {
    char c[2];
    unsigned int count;
  } cnt;

  if(!(fp=fopen(fname,"rb"))) {
    printf("Вхідний файл не може бути відкритим\n");
    exit(1);
  }
  send_file_name(fname);   /* передача імені файлу */
  wait(PORT); /* очікування звітувального байта */

 /* обчислення розміру вихідного файлу */
  cnt.count = filesize(fp);

 /* розмір посилання */
  sport(PORT, cnt.c[0]);
  wait(PORT);
  sport(PORT, cnt.c[1]);

  do {
    ch = getc(fp);
    if(ferror(fp)) {
      printf(" помилка читання вихідного файлу\n");
      break;
    }

/* очікування готовності порта-приймача */
    if(!feof(fp)) {
      wait(PORT);
      sport(PORT, ch);
    }
  } while(!feof(fp));
  wait(PORT);
/* очікування підтвердження отримання останнього байта*/
  fclose(fp);
}
```

Функція send_file_name(), наведена нижче, встановлює відповідність між ім'ям прийнятого і переданого файлів.

```
/* Перекачка імені файлу */
void send_file_name(f)
char *f;
{
  printf(" Очікування передачі... \n");
  do {
    sport(PORT, '?');
   } while(!kbhit() && !(check_stat(PORT)&256));
   if(kbhit()) {
     getch();
    exit(1);
  }
  wait(PORT);/* очікування отримання звітувального байта */
  printf("Передано %s\n\n",f);

/* фактична передача імені файлу */
  while(*f) {
    sport(PORT, *f++);
    wait(PORT); /* очікування отримання звітувального байта
*/
  }
  sport(PORT,'\0'); /* символ кінця рядка */
}
```

Функція `send_file_name()` призначена для розв'язання двох основних задач: по-перше, вона встановлює зв'язок з комп'ютером-приймачем шляхом передачі йому маркера запитання ('?') і чекає відповіді від нього у вигляді звітувального байта (як звітувальний символ використовується крапка. Однак можна за своїм розсудом використовувати інший символ). Після того, як зв'язок буде встановлено, здійснюється передача імені файлу. Ця функція завершує аварійно свою роботу при надходженні переривання від клавіатури.

Функція `wait()`, наведена нижче, очікує звіт від комп'ютера-приймача, що реалізує програмне підтвердження зв'язку.

```
/* очікування відповіді */
void wait(port)
int port;
{
  if(rport(port)!='.') {
 printf("помилка встановлення зв'язку \n");
    exit(1);
  }
}
```

Таким чином, при виявленні помилки ця функція припиняє свою роботу. Однак можна передбачити обробку даної ситуації.

Функція `filesize()` повертає розмір файлу в байтах. Її використання можливо, якщо компілятор С підтримує функцію обчислення довжини файлу, у протилежному випадку потрібно замінити цю функцію розробленою вами, але таку, що виконує аналогічні дії. Змінна ***cnt***, що входить до складу структури `union`, служить для збереження двобайтової довжини файлу, але потрібно пам'ятати, що за одиницю часу можна переслати через послідовний порт тільки один байт.

Прийом файлу

Прийом файлу є прямо протилежною операцією передачі файлу. По-перше, функція прийому очікує маркер запиту на одержання даних (символ '?'). На одержання маркера функція відповідає крапкою (символом звітування). Після одержання імені файлу функція очікує одержання його розміру в байтах. В остаточному підсумку функція починає читання файлу. Після одержання і читання кожного байта функція посилає комп'ютеровіджерелу звітувальний байт. У такий спосіб вона реалізує програмне підтвердження зв'язку. Функція `rec_file()` наведена нижче.

Приклад 5.6. Прийом файлу

```
void rec_file()
{
  FILE *fp;
  char ch;
  char fname[14];
  union {
    char c[2];
    unsigned int count;
  } cnt;

  get_file_name(fname); /* одержання імені файлу */
  printf(" Отриманий файл %s\n",fname);
  remove(fname);
  if(!(fp=fopen(fname, "wb"))) {
    printf(" Неможливо відкрити вихідний файл \n");
    exit(1);
  }

 /* Одержання довжини файлу */
  sport(PORT, '.'); /* звітування */
  cnt.c[0] = rport(PORT);
 sport(PORT, '.'); /* звітування */
  cnt.c[1] = rport(PORT);
  sport(PORT, '.'); /* звітування */
  for(; cnt.count; cnt.count--) {
    ch = rport(PORT);
    putc(ch, fp);
```

```
    if(ferror(fp)) {
      printf(" помилка запису у файл ");
      exit(1);
    }
    sport(PORT, '.'); /* звітування */
  }
  fclose(fp);
}
```

Функція `get_file_name()` наведена нижче.

```
/* Одержання імені файлу */
void get_file_name(f)
char *f;
{
  printf("Чекання одержання...\n");
  while(rport(PORT)!='?') ;
  sport(PORT, '.'); /* звітування */
  while((*f=rport(PORT))) {
    if(*f!='?') {
      f++;
      sport(PORT, '.'); /* звітування */
    }
  }
}
```

5.7 Контрольні запитання

1. За допомогою яких функцій здійснюється введення-виведення одного символу?
2. Для чого призначені функції `gets()` і `puts()`?
3. Що являє собою текстовий файл?
4. Які функції призначені для роботи з текстовими файлами?
5. Напишіть функції, які здійснюють відкриття файлу, та вкажіть їх параметри?
6. Які ви знаєте режими відкриття текстового файлу?
7. Що таке двійковий файл?
8. Які переваги двійкових файлів ви знаєте?
9. Як здійснюється відкриття двійкових файлів?
10. Які є способи читання і запису даних у двійкові файли? Наведіть приклад.
11. Як здійснюється запис даних у файл з послідовним доступом? Наведіть приклад
12. Як відбувається читання даних з файлу послідовного доступу? Наведіть приклад.
13. Які особливості файлу з довільним доступом? Наведіть приклад.

14. Як здійснюється перекачування файлів?
15. Як відбувається прийом файлів з порту?

5.8 Практикум з програмування

1. Написати фрагмент програми для підрахування кількості рядків в текстовому файлі.
2. Написати фрагмент програми для підрахування кількості слів в текстовому файлі.
3. Написати фрагмент програми для порівняння двох текстових файлів та виведення номера рядка та позиції символу, де вони відрізняються.
4. Написати фрагмент програми для підрахування скільки разів з'являється дане слово в текстовому файлі.
5. Написати фрагмент програми для дозапису рядка в початок текстового файлу.
6. Написати фрагмент програми для дозапису даних у початок існуючого двійкового файлу.
7. Написати фрагмент програми для дозапису даних у середину існуючого двійкового файлу з заданої позиції.
8. Написати фрагмент програми для видалення заданої кількості даних з початку двійкового файлу.
9. Написати фрагмент програми для видалення заданої кількості даних з заданої позиції заданого двійкового файлу і вставляння їх з заданої позиції у новий файл.
10. Написати фрагмент програми для перезапису змісту файлу з кінця в початок у зворотному порядку.

6 ОСОБЛИВОСТІ РОБОТИ З ТЕКСТОВИМИ ДАНИМИ МОВОЮ С

6.1 Загальні відомості

Для подання текстової інформації мовою С використовуються символи (константи), символьні змінні і рядки (рядкові константи), для яких у стандарті мови С не введено окремого типу на відміну від деяких інших мов програмування.

Для символьних даних введений базовий тип `char`. Опис символьних змінних має вигляд:

```
char список_імен_змінних;
```

Наприклад:

```
char a,z;
```

Введення-виведення символьних даних

Для введення і виведення символьних значень у форматних рядках бібліотечних функцій `printf()` і `scanf()` використовується специфікація перетворення `%c`.

Розглянемо таку задачу.

Ввести речення, слова в якому розділені пропусками і наприкінці якого стоїть крапка. Видалити пропуски, які повторюються між окремими словами (залишити по одному пропуску), вивести відредаговане речення на екран.

Текст програми:

```
/* Видалення повторюваних пропусків */
#include "stdafx.h"

void main(   )
{
  char  z,s; /* z  - поточний символ, що вводиться */
  printf("\n Напишіть речення з крапкою в кінці:\n");
   for (z=s=' ';z!='.';s=z)
       {/*   s  -  попередній символ  */
       scanf("%c",&z);
       if   (z==' '&&  s==' ') continue;
       printf ("%c",z) ;
       } /* Кінець циклу обробки */
 }  /* Кінець програми */
```

У програмі дві символьні змінні: `z` – для читання попереднього символу і `s` – для збереження попереднього. В заголовку циклу змінні `z` і `s` одержують значення «пропуск». Черговий символ вводиться як значення змінної `z`, і пара `z`, `s` аналізується. Якщо хоча б один із символів значень пари

відмінний від пропуску, то значення z друкується. В заголовку циклу z порівнюється із символом «крапка» і при розбіжності запам'ятовується як значення s. Далі цикл повторюється до появи на вході крапки, причому поява двох пропусків (z і s) приводить до пропускання оператора друку.

Приклад роботи програми.

Напишіть речення з крапкою в кінці:

```
YYYYY      yyyyy     hhhhh   ttttt.
YYYYY yyyyy hhhhh ttttt.
```

Крім `scanf()` і `printf()` для введення і виведення символів в бібліотеці передбачені спеціальні функції обміну:

`getchar()` – функція без параметрів. Дозволяє читати з вхідного потоку (звичайно з клавіатури) по одному символу за одне звертання. Як і при використанні функції `scanf()`, читання даних, що вводяться, починається після натискання клавіші **<Enter>**. Це дає можливість виправляти інформацію, що вводиться (стираючи останні введені символи за допомогою клавіші **<Backspace>**) до початку її читання програмою;

`putchar(X)` – виводить символьне значення `X` в стандартний потік (звичайно на екран дисплея).

Внутрішні коди й упорядкованість символів

У мові прийнята угода, що скрізь, де синтаксис дозволяє використовувати цілі числа, можна використовувати і символи, тобто дані типу `char`, що при цьому подаються числовими значеннями своїх внутрішніх кодів. Така угода дозволяє порівняно просто упорядковувати символи, працюючи з ними як з цілочисловими величинами. Наприклад, внутрішні коди десяткових цифр в таблицях кодів ASCII упорядковані за числовим значенням, тому нескладно перебирати символи десяткових цифр у потрібному порядку.

Рядки чи рядкові константи

У програмі рядки або рядкові константи подаються послідовністю зображень символів, які поміщені в лапки (не в апострофи), наприклад "будь-які символи". Серед символів рядка можуть бути ескейп-послідовності, що відповідають кодам не зображуваних (спеціальних) символьних констант.

Приклади рядків:

"1234567890"

"\t Склад президії"

"Початок рядка \n і кінець рядка".

Однак у рядків є деякі особливості. Транслятор відводить кожному рядку окреме місце в пам'яті ЕОМ навіть у тих випадках, коли кілька рядків повністю збігаються (стандарт мови С припускає, що в конкретних реалізаціях це правило може не виконуватися). Розміщаючи рядок у пам'яті, транслятор автоматично додає в його кінці символ `'\0'`, тобто нульовий

байт. У записі рядка може бути й один символ: "A", однак на відміну від символьної константи 'A' (використані апострофи) довжина рядка "A" дорівнює двом байтам. На відміну від інших мов (наприклад, від Паскаля) у мові С немає окремого типу для рядків. Прийнято, що рядок – це масив символів, тобто він завжди має тип char[ ]. Таким чином, рядок вважається значенням типу "масив символів". Кількість елементів у такому масиві на 1 більша, ніж у зображенні відповідної рядкової константи, тому що в кінець рядка доданий нульовий байт '\0'.

Присвоїти значення масиву символів (тобто рядку) за допомогою звичайного оператора присвоювання не можна. Помістити рядок у масив можна або за допомогою ініціалізації (при визначенні символьного масиву), або за допомогою функцій введення. У функції scanf() чи printf() для символьних рядків використовується специфікація перетворення %s.

При визначенні масиву типу char з одночасною ініціалізацією можна не вказувати межі зміни індексу. Сказане ілюструє нижченаведена програма:

```
/* Друкування символьного рядка */
#include <stdio.h>
void main( )
   {
char B[]="Сезам, відкрийся!";
printf("%s",B);
    } /* Кінець програми */
```

Результат виконання програми:

```
Сезам, відкрийся!
```

У програмі довжина масиву B – 17 елементів, тобто довжина рядка, що поміщається в масив (16 символів), плюс нульовий байт закінчення рядка. Саме 17 байтів виділяється при ініціалізації масиву в наведеному прикладі. Ініціалізація масиву символів за допомогою рядкової константи являє собою скорочений варіант ініціалізації масиву і введена в мову для спрощення. Можна скористатися звичайною ініціалізацією, помістивши початкові значення елементів масиву у фігурні дужки і не забувши при цьому помістити в кінці списку початкових значень спеціальний символ закінчення рядка '\0'. Таким чином, у програмі була б припустима така ініціалізація масиву B:

```
   char     B[     ]={'С','е','з','а','м',',',' ','в','і',
'д','к','р','и','й','с','я','!','\0'    };
```

Домовленість про обов'язкову наявність ознаки закінчення рядка потрібно дотримуватись, формуючи в програмах рядки з окремих символів. Як приклад розглянемо таку задачу: «Ввести речення, що закінчується крапкою, слова в якому відділені пропусками. Надрукувати останнє слово речення».

Аналіз умови задачі дозволяє виявити такі помилкові ситуації й особливі випадки: відсутність крапки в кінці речення; пропуск чи пропуски перед першим словом речення; кілька пропусків між словами; пропуски перед крапкою, що завершує речення; відсутність слів у реченні (тільки крапка). Щоб не ускладнювати розв'язання задачі, приймемо умову про те, що речення, яке вводиться, завжди поміщається на одному рядку дисплея, тобто довжина його не перевищує 80 символів. Це дозволить легко виявляти відсутність крапки в кінці речення й обмежує довжину будь-якого слова речення. Щоб врахувати особливі ситуації з пропусками, необхідно при аналізі чергового введеного символу (змінна s) розглядати і попередній символ (змінна ss). Для виявлення відсутності слів у реченні будемо обчислювати довжину k кожного чергового слова, що вводиться. Якщо k==0, а вводиться символ '.', то це ознака порожнього речення.

Текст програми:

```
/* Надрукувати останнє слово в реченні */
#include <stdio.h>
void main(   )
{
  char s,ss;        /* s – символ, що вводиться*/
                    /* ss – попередній уведений символ */
  char A[80]; /* масив для слова */
  int i,k;      /* k – довжина слова */
  printf("Напишіть речення з крапкою в "
                    "кінці:\n");
  for (i=0,s=' ',k=0; ³<=79; i++)
       {
            ss=s; s=getchar( );
            if (s==' ') continue;
            if (s=='.') break.;
            if (ss==' ') k=0;
            A[k]=s; k++;
       }
/*Вихід за крапкою чи після закінчення введення рядка*/
  if (i==80 || k=0)
       printf(" Неправильне речення \n");
  else
  {
       A[k]='\0'; /* Кінець рядка */
       printf("Останнє слово: %s",A) ;
  }
} /* Кінець програми  */
```

Читання даних, що вводяться, виконується посимвольно в циклі з параметром. Якщо вводиться пропуск, то оператор continue викликає перехід до наступної ітерації. Якщо введена крапка, то цикл переривається.

При цьому в перших k елементах масиву A[] запам'ятовується останнє слово пропозиції. Якщо введений символ відмінний від крапки і пропуску, то аналізується попередній символ. Якщо це пропуск, то починається введення наступного слова і встановлюється нульове значення k. У наступних операторах введений символ записується в k-й елемент масиву A і k збільшується на 1. Вихід з циклу можливий з появою крапки або після введення 80 символів. Останнє виконується при відсутності в реченні крапки. У цьому помилковому випадку i==80. Коли в реченні немає слів, k залишається рівним нулю. Якщо i<80 і k не дорівнює 0, то в k-й елемент масиву A записується ознака кінця рядка (останнього слова речення), і вона як єдине ціле виводиться на друк.

Результат виконання програми:

```
Напишіть речення з крапкою в кінці:
Орфографічний словник. Останнє слово: словник
```

Як ще раз ілюструє наведена програма, робота з символьними рядками – це робота з масивами типу char. При цьому варто звертати увагу на обов'язкову присутність в кінці рядка (в останньому зайнятому елементі масиву) символу '\0' і не допускати його випадкового знищення при обробці.

6.2 Рядки та вказівники

Розглянемо такі два визначення:

```
char A[20];
/* Масив, у який можна записати рядок */
char *B;
/* Вказівник, з яким можна зв'язати рядок */
```

Масив A одержує пам'ять автоматично при обробці визначення. Рядку, з яким хочемо зв'язати вказівник B, пам'ять при обробці визначення вказівника не виділяється. Тому якщо далі слідують оператори

```
scanf("%s",A);   /* Оператор правильний */
scanf("%s",B);   /* Оператор некоректний */,
```

то перший з них допустимий, а другий приводить до помилки під час виконання програми – спроба введення в невизначену ділянку пам'яті. До виконання другого з цих операторів введення з вказівником B потрібно пов'язати деяку ділянку пам'яті. Для цього існує кілька можливостей. По-перше, змінній B можна присвоїти адрес вже визначеного символьного масиву. По-друге, вказівник B можна «налаштувати» на ділянку пам'яті, що виділяються за допомогою засобів динамічного розподілу пам'яті.

Наприклад, оператор

```
B=(char  *)   malloc(80);
```

виділяє 80 байтів і пов'язує цей блок пам'яті з вказівником B. Тільки тепер застосування наведеного вище оператора введення припустимо. Прототи-

пи функції `malloc()` й інших функцій розподілу пам'яті знаходяться у файлі **stdlib.h**

За допомогою вказівників типу `char*` зручно одержувати доступ до рядків у вигляді масивів символів. Типова задача – обробка слів чи речень, кожне з яких подано в масиві типу `char` у вигляді рядка (тобто, в кінці подання речення чи слова знаходиться нуль-символ `'\0'`). Використання вказівників, а не масивів з фіксованими розмірами, особливо доцільно, коли речення чи слова повинні бути різної довжини. У наступній програмі визначений і при ініціалізації пов'язаний з набором рядків одновимірний масив `point[]` вказівників типу `char*`. Функція `printf()` і специфікатор перетворення `%s` допускають використання як параметр вказівника на рядок. При цьому на дисплей (у вихідний потік) виводиться не значення вказівника `point[i]`, а вміст адресованого йому рядка.

Текст програми:

```
#include    <stdio.h>
void main(   )
{
       char * point[ ]={"thirteen","fourteen",
       "fifteen","sixteen","seventeen","eighteen",
       "ninteen"};
       int i, n;
       n=sizeof(point)/sizeof(point[0]);
       for (i=0 ; i<n; i++)
       printf("\n%s",point[i]);
}
```

Результат виконання програми:

```
Thirteen fourteen fifteen sixteen seventeen eighteen
Ninetee
```

Рядки як параметри функцій

Рядки як фактичні параметри можуть бути специфіковані або як одновимірні масиви типу `char[]`, або як вказівники типу `char*`. В обох випадках за допомогою параметра в функцію передається адрес початку символьного масиву, що містить рядок. На відміну від звичайних масивів для параметрів рядка немає необхідності явно вказувати їх довжину. Ознака `'\0'`, що розміщується в кінці кожного рядка, дозволяє завжди визначити його довжину, точніше, дозволяє перебирати рядки і не вийти за його межі. Як приклад використання рядка як параметра функцій розглянемо декілька функцій, що розв'язують типові задачі обробки рядків. Аналогії більшості з наведених далі функцій є в бібліотеці стандартних функцій компілятора. Їх прототипи та інші засоби зв'язку з цими функціями знаходяться в заголовних файлах **string.h** і **stdlib.h**.

6.3 Стандартні функції для роботи з рядками (бібліотека string.h)

В мові С існують стандартні функції для обробки рядків, які спрощують розробку програм. Ці Функції знаходяться в бібліотеці string.h. Розглянемо основні з них.

Таблиця 6.1 – Основні стандартні функції обробки рядків

Назва функції	Призначення	Синтаксис	Опис
stpcpy	Копіювати рядок у рядок	`char *stpcpy(char *dest,const char *src);`	Копіює вихідний рядок `src` і нуль, що його завершує, у рядок результату `dest`. Повертає `dest+strlen(src)`.

strcat, fstrcat	Конкатенувати рядок	`char *strcat(char *dest, const char *src);`
		
		`char far *far _fstrcat(char far *dest, const char far *sfc);`
			Конкатенує (об'єднує) вихідний рядок `src` і ініціалізований у підсумковий рядок `dest`, приєднуючи останній до кінця першого. Повертає `dest`

strchr, fstrchr	Шукати в рядку символ	`char *strchr(const char *s, int c);`
		
		`char far * far _fstrchr(const char far *s, int c);`
			Шукає в рядку `s` перше входження символу `c`, починаючи з початку рядка. У випадку успіху повертає вказівник на знайдений символ, інакше повертає нуль

strcmp, fstrcmp	Порівняти рядки	`int strcmp(const char *s1, const char *s2);`
		
		`int far _fstrcmp(const char far *s1, const char far *s2);`
			Порівнює два рядка. Повертає від'ємне значення, якщо `s1 < s2;` нуль, якщо `s1 == s2;` додатнє значення, якщо `s1 > s2`

Продовження таблиці 6.1

strcmpi	Порівняти рядки не розрізняючи малі та великі літери	int strcmpi(const char *s1, const char *s2);	Ця функція аналогічна функції strcmp(), але ігнорує різницю між малими та великими літерами. Для сумісності з іншими компіляторами мови С функція strcmpi() реалізована у вигляді макросу, що безпосередньо викликає функцію strcmpi().

strcpy	Копіювати рядок у рядок	char *strcpy(char *dest,const char *src);
		
		char far * far_fstrcpy(char far*dest, const char far *src);
			Копіює вихідний рядок src і нульовий символ, який завершує його, у рядок результату dst, перезаписуючи символи підсумкового рядка, розташовані в місці копіювання. Повертає dest.

strftime	Запам'ятати дату і час у рядку	size_t strftime(char *s, size_t maxsize, const char *fmt const struct tm *t);,	Форматує дату і час у вигляді текстового рядка, використовуючи систему правил перетворення, аналогічну системі printf().
strncpy, _fstrncpy	Копіювати частини рядка	char *strncpy(char *dest, const char *src, size_t maxlen); char far *far _fstrncpy(char far *dest, const char far *src, size _t maxlen);	Копіює максимум maxlen символів із вихідного рядка src у підсумковий рядок dest, перезаписуючи символи підсумкового рядка.

Продовження таблиці 6.1

strstr, _fstrstr	Шукати в рядку підрядок	char *strstr (const char *s1, const char *s2); char far *far _ fstrstr (const char far *s1, const char far *s2);	Шукає рядок (s2) в іншому рядку (s1). Повертає адресу першого символу входження рядка або, якщо рядок s2 не знайдений, в рядок s1 повертає нуль.
strtod	Перетворити рядок у число типу double	double strtod (const char *s, char **endptr); long double _strtold(const char *s, char ** endptr);	Перетворює символьне подання числа з плаваючою крапкою у його двійкове подання типу double чи long double. У випадку успіху повертає отриманий при перетворенні результат, а у випадку помилки повертає HUGE_VAL (strtod ())
strtol	Перетворити рядок у довге ціле число	long strtol (const char *s, char **endptr, int radix);	Перетворює символьне подання значення типу long у його двійкове подання. Значення в рядку може бути подано у десятковому, вісімковому чи шістнадцятковому вигляді, що використовує стандартні правила форматування мови С (що діють для printf( ), scanf() і інших аналогічних функцій).

6.4 Приклади використання стандартних функцій для роботи з рядками

Розглянемо приклади використання основних стандартних функцій для роботи з рядками.

Функція `stpcpy()` – копіює вихідний рядок `src` і нуль, що його завершує, у рядок результату `dest`. Повертає `dest + strlen(src)`.

Синтаксис: `char *stpcpy(char *dest,const char *src);`

Параметри:

- `char *dest` – вказівник на рядок результату достатньо великого розміру, щоб містити результат.
- `const char *src` – вказівник на вихідний рядок, що завершується нульовим символом.

Функція аналогічна `strcpy(), strncpy()`

Приклад 6.1.

```
#include<stdio.h>
#include<string.h>
main()
{
  char src[80] = "abcdefghij";
  char dst[80] = "1234567890";
  printf("До: джерело==%s\n",src, dst);
  puts("Викликаємо stpcpy(приймач, джерело) ");
  stpcpy(dst, src);
  printf("Після: джерело==%s приймач==%s\n",src, dst);
  return 0;
 }
```

Функція `strcat(), fstrcat()` – конкатенує (об'єднує) вихідний рядок `src` і ініціалізований підсумковий рядок `dest`, приєднуючи останній до кінця першого. Повертає `dest.( string.h)`

Синтаксис: `char *strcat(char *dest, const char *src);`

`char far *far _fstrcat(char far *dest, const char far *sfc);`

Параметри:

- `char *dest` – вказівник на ініціалізований рядок результату.
- `const char *dest` – вказівник на вихідний рядок, що приєднується до кінця підсумкового.

Функція аналогічна `stpcpy(), strcpy(), strncat()`

Приклад 6.2.

```
#include<stdio.h>
#include<string.h>
main()
{
char src[80] = "abcdefghij";
```

```
char dst[80] = "1234567890";
printf("До: джерело ==%s        приймач==%s\n", src, dst);
puts("Викликаємо strcat(приймач, джерело)");
strcat(dst, src);
printf("Після: джерело==%s  приймач==%s\n", src, dst);
return 0;
}
```

Функція `strchr(), fstrchr()` – шукає в рядку `s` перше входження символу `c`, починаючи з початку рядка. У випадку успіху повертає вказівник на знайдений символ, інакше повертає нуль **(string.h).**

Синтаксис: `char *strchr(const char *s, int c);`
`char far * far _fstrchr(const char far *s, int c);`

Параметри:

- `const char *s` – вказівник на рядок.
- `int c` - шуканий символ. Для пошуку нульового символу, що завершує рядок, вкажіть нульове значення.

Функція аналогічна `strcspn(), strrchr(), trspn(), strstr()`

Приклад 6.3.

```
#include<stdio.h>
#include<string.h>
main()
{
char src[80] = "abcdefghij";
printf("Рядок у src==%s\n", src);
puts("Викликаємо char *p = strchr(src, 'd')");
char *p = strchr(src, 'd');
printf("Рядок у p  == %s\n", p);
return 0;
}
```

Функція `strcmp(), fstrcmp()` – порівнює два рядки. Повертає від'ємне значення, якщо `s1 < s2`; нуль, якщо `s1 == s2`; додатнє значення, якщо `s1 > s2` **(string.h)**

Синтаксис: `int strcmp(const char *s1, const char *s2);`
`int far _fstrcmp(const char far *s1, const char far *s2);`

Параметри:

- `const char *s1` – вказівник на перший порівнюваний рядок.
- `const char *s2` – вказівник на другий порівнюваний рядок.

Функція аналогічна `strchr(), strcmpi(), strcol()l, stricmp(), strncmp(), strnicmp()`

Приклад 6.4.

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
main(int argc, char *argv[])
{
   char *p;
```

```
    if (argv <= 2)  {
    puts("Введіть два рядки");
    puts("Наприклад, STRCMP  рядокА   рядокБ");
   exit(1);
  }
   int result = strcmp(argv[1], argv[2]);
   if (result < 0)
   p = "менше";
  else if (result > 0)
  p = "більше";
 else
  p = "дорівнює";
  printf("%s  %s  %s", argv[1], p, argv[2]);
  return 0;
  }
```

Функція `strcmpi()` – ця функція аналогічна функції `strcmp()`, але вона ігнорує різницю між малими та великими літерами. Для сумісності з іншими компіляторами мови С функція `strcmpi()` реалізована у вигляді макросу, що безпосередньо викликає функцію `strcmpi()`.

Синтаксис: `int strcmpi(const char *s1, const char*s2)`

Параметри:

- `const char *s1` – вказівник на перший порівнюваний рядок;
- `const char *s2` – вказівник на другий порівнюваний рядок.

Функція аналогічна `strchr()`, `strcmp()`, `strcoll()`, `stricmp()`, `strncmp()`, `strnicmp()`

Приклад 6.5.

```
/*strcmpi.cpp*/
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
main(int argc, char *argv[])
 {
  char *p;
 if (argv <= 2)
 {
 puts("Введіть два рядки, що порівнюються без урахування різниці між малими та великими літерами");
puts("Наприклад, STRCMPI рядока рядокб");
 exit(1);
 }
 int result = strcmpi(argv[1], argv[2]);
 if (result < 0)
 p = "менше";
 else if (result > 0)
 p = "більше";
 else
 p = "дорівнює";
 printf("%s  %s  %s", argv[1], p, argv[2]);
 return 0;
```

}

Функція strcpy() – копіює вихідний рядок src і нульовий символ, який його завершує, у рядок результату dst, перезаписуючи символи підсумкового рядка, розташовані в місці копіювання. Повертає dest. **(string.h)**

Синтаксис:
```
char *strcpy(char *dest,const char *src);
char far * far _fstrcpy(char far *dest, const
char far *src);
```

Параметри:

- char *dest – вказівник на рядок результату, що перезаписується вихідним рядком. Підсумковий рядок не обов'язково повинен бути ініціалізованим;
- const char *src – вказівник на вихідний рядок, що завершується нульовим символом.

Функція аналогічна stpcpy(), strncpy()

Приклад 6.6.

```
    /*strcpy.cpp*/
#include<stdio.h>
#include<string.h>
 main()
{
char src[80] = "abcdefghij";
char dst[80] = "1234567890";
puts("Викликаємо strcpy(приймач, джерело)");
strcpy(dst, src);
printf("Після: джерело==%s  приймач==%s\n", src, dst);
return 0;
}
```

Функція strdate() – запам'ятовує поточну дату у вигляді рядка у форматі mm/dd/yy (mm – місяць, dd – день, yy – рік. Повертає datestr, **(time.h)**

Синтаксис: `char *_strdate(cha *datestr);`

Параметри:

- char *datestr – вказівник на підсумковий рядок довжиною, принаймні, 9 байтів.

Функція аналогічна asctime(), ctime(), strtime(), time()

Приклад 6.7.

```
      /*_strdate.cpp*/
#include<stdio.h>
#include<time.h>
 main()
{
  char sdate[9];
  _strdate(sdate);
```

```
    printf(": %s\n", sdate);
    return 0;
}
```

Функція `strerror()` – створює з рядка `s` рядок повідомлення про помилку, додаючи двокрапку, пропуск і опис поточної системної помилки **(string.h** чи **stdio.h)**.

Синтаксис: `char *_strerror(const char *s);`

Параметри:

- `const char *s` – повідомлення користувача про помилку довжиною не більше 94 символів, до якого додається повідомлення про системну помилку. Якщо цей вказівник дорівнює нулю, `_strerror()` повертає вказівник на рядок повідомлення про помилку з найостаннішим кодом помилки.

Функція аналогічна `perror(), strerror()`

Приклад 6.8.

```
#include<stdio.h>
#include<string.h>
 main()
{
char *p = _strerror("Перевірне повідомлення про помилку");
printf("Повідомлення про помилку == %s\n", p);
p = _strerror(NULL);
printf("Найостанніша помилка == %s\n", p);
return 0;
}
```

Функція `strftime()` – форматує дату і час у вигляді текстового рядка, використовуючи систему правил перетворення, аналогічну системі `printf()`. Форматний рядок `fmt` містить одне чи декілька правил, що заміняються компонентами дати і часу. Крім того, форматний рядок може також містити й інші символи, що не належать до правил перетворення. Повертає число символів, записаних у підсумковий рядок **(time.h).** Правило перетворення складається зі знака відсотка (%) і символу.

Синтаксис: `size_t strftime(char *s, size_t maxsize, const char *fmt, const struct tm *t);`

Параметри:

- `char *s` – вказівник на підсумковий рядок, в який запам'ятовується результат функції;
- `size_t maxsize` – максимальне число символів, записуваних у підсумковий рядок. Звичайно встановлюється рівним розміру рядка мінус одиниця;
- `const char *fmt` – вказівник на форматний рядок, що містить звичайний текст і правила перетворення;
- `const struct tm *t` – вказівник на структуру типу `tm`, що містить дату і час, форматуємо у рядок.

Функція аналогічна `asctime()`, `ctime()`, `localtime()`, `mktime()`, `time()`.

Таблиця 6.2 – Правила перетворення для `strftime()`

Правила перетворення	Компоненти дати і часу
%	Вставити знак відсотка (%)
A	День тижня (абревіатура: Sun, Mon і т. д.)
A	Повна назва дня тижня
B	Місяць (абревіатура: Jan, Feb і т. д.)
B	Повна назва місяця
C	Дата і час у форматі asctime()
D	День місяця (від 01 до 31)
H	Час в 24-годинному форматі (від 00 до 23)
I	Час в 12-годинному форматі (від 00 до 12)
J	День року (від 001 до 366)
M	Номер місяця (від 1 до 12)
M	Хвилина (від 00 до 59)
P	Букви AM чи PM (що позначають час до чи після півдня відповідно)
S	Секунда (від 00 до 59)
U	Номер тижня (від 00 до 53) (тиждень починається з неділі)
w	День тижня (від 0 до 6) (неділя == 0)
W	Номер тижня (від 00 до 53) (тиждень починається з понеділка)
X	Дата
X	Час
Y	Рік мінус сторіччя (наприклад, 68 для 1968)
Y	Повний рік (наприклад, 1968)
Z	Найменування часової зони (EST чи EDT)

Приклад 6.9.

```
#include<stdio.h>
#include<time.h>
#define SIZE 80
main()
{
time_t t;
struct tm *tp;
char s[SIZE];
time(&t);
tp = localtime(&t);
puts("");
strftime(s, SIZE, "Дата: %x\n", tp);
puts(s);
strftime(s, SIZE, "Час: %x\n", tp);
puts(s);
strftime(s, SIZE, "Сьогодні %A\n", tp);
puts(s);
```

```
.
return 0;
}
```

Функція `strncpy(), fstrncpy()` – копіює максимум `maxlen` символів із вихідного рядка `src` у підсумковий рядок `dest`, перезаписуючи символи підсумкового рядка. Якщо `maxlen` дорівнює розміру в байтах підсумкового рядка (і вихідний рядок має не меншу довжину), то підсумковий рядок не завершується нульовим символом. Якщо `maxlen` і довжина вихідного рядка перевищують розмір підсумкового рядка, кінець підсумкового рядка перезаписується, що, можливо, зруйнує інші дані чи код у цьому місці пам'яті. Щоб уникнути подібних помилок, ніколи не встановлюйте `maxlen` більше максимального числа символів, що їх може містити підсумковий рядок **(string.h)**

Функція аналогічна `strcat(), strcpy()`

Приклад 6.10.

```
#include<stdio.h>
#include<string.h>
main ( )
{
   char src[80] = "abcdefghij";
   char dst[80] = "1234567890";
   printf ("До: src = =%s  dst = =%s \n", src, dst);
   puts ("Викликаємо strncpy (dst, src, 5)");
   strncpy (dst, src, 5);
   printf ("Після: src == %s  dst = = %s \n", src, dst);
   return 0;
}
```

Функція `strstr(), fstrstr()` – шукає рядок `(s2)` в іншому рядку `(s1)`. Повертає адресу першого символу входження рядка або, якщо підрядок `s2` не знайдений в рядку `s1`, – повертає нуль **(string.h)**

Синтаксис:

```
char *strstr (const char *s1, const char *s2);
char far *far _ fstrstr (const char far *s1, const char far
*s2);
```

Параметри:

- `const char *s1` – вказівник на рядок, у якому відбувається пошук підрядка, адресуємо `s2`;
- `const char *s2` – вказівник на підрядок, що шукається в рядку, який адресується `s1`.

Функція аналогічна `strchr(), strcmp(), strspn()`

Приклад 6.11.

```
                        /*strstr.cpp*/
#include<stdio.h>
#include<string.h>
```

```
 main ()
{
   char *s1= "filename.cpp";
   char *s2=".cpp";
   printf ("s1= = %s \n", s1 );
   printf ("s2= = %s \n", s2);
puts ("Викликаємо char *s3 = strstr (s1, s2) ");
   char *s3= strstr (s1, s2);
   printf ("s3== %s \n", s3);
   return 0;
}
```

Функція `strtod()` – перетворить символьне подання числа з плаваючою крапкою, що міститься в рядку, у його двійкове подання типу `double` чи `long double`. У випадку успіху повертає отриманий при перетворенні результат, а у випадку помилки повертає `HUGE_VAL (strtod ())` чи `LHUGE _VAL` помилки повертає `HUGE_VAL (strtod ())` чи `LHUGE _VA` **(stdlib.h)**

Синтаксис:

```
double strtod (const char *s, char **endptr);
long double  strtold (const char *s, char ** endptr);
```

Параметри:

- `const char *s` – вказівник на рядок, що містить символьне подання числа з плаваючою крапкою або в десятковій нотації (наприклад, "123.45"), або в науковій нотації (наприклад, "4.5e-3");

- `char **endptr` – якщо цей параметр не дорівнює нулю, то він повинен бути рівним адресі символу, розташованого безпосередньо після останнього символу в рядку s, що бере участь у перетворенні. Цей необов'язковий вказівник використовується при аналізі рядків, що містять кілька значень з плаваючою комою, можливо, розділених символами пропуску, комами.

Функція аналогічна `atof()`, `printf()`, `sprintf()`, `strtol()`

Приклад 6.12.

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
main (int argc, char *argv [ ])
{
   if (argc <= 1)
   {
       puts ("Введіть значення типу double");
       puts ("наприклад, STRTOD 3.14159");
        exit (1);
   }

   char *endptr;
```

```
    double d = strtod (argv[1], &endptr);
    printf ("Значення у двійковому поданні = =  %lf \n", d);

    if (strlen (endptr)>0)
      printf ("Перегляд зупинено на: %s \n", endptr);
    return 0;
}
```

Функція `strtol()` – перетворить символьне подання значення типу `long` у його двійкове подання. Значення в рядку може бути подано в десятковому, вісімковому чи шістнадцятковому вигляді, що використовує стандартні правила форматування мови С (що діють для `printf( )`, `scanf( )` та інших аналогічних функцій). Розпізнаються також інші основи системи числення, від 2 до 36. У випадку успіху функція повертає перетворений результат і встановлює ненульовий вказівник `endptr`, рівним адресі, що слідує за останнім символом, який входить у символьне подання числа. У випадку помилки функція повертає нуль і встановлює ненульовий вказівник `endptr`, рівним `s`.

Синтаксис:

```
long strtol (const char *s, char **endptr, int radix);
```

Параметри:

- `const char *s` – вказівник на рядок, що містить ціле значення типу `long` у текстовому вигляді. Цій рядок може також містити й інші символи;
- `char ** endptr` – якщо цей параметр не дорівнює нулю, то він повинен бути рівним адресі символу, розташованого безпосередньо після останнього символу в рядку `s`, що бере участь у перетворенні. Цей необов'язковий вказівник використовується при аналізі рядків, що містять кілька значень типу `long`, можливо, поділених символами пропуску, комами і т. п.;
- `int radix` – задає основу системи числення, яка використовується при перетворенні і може приймати значення від 2 до 36. Наприклад, 2 – для двійкових значень, 10 – для десяткових значень, 16 – для шістнадцяткових значень і т. д. Літери від A до Z розпізнаються як числові символи для основ, що перевищують 10. (При основі 16 для подання числа використовуються цифри від 0 до 9 і букви від A до F, при основі 17 – цифри від 0 до 9 і букви від A до G і т. п.) Якщо параметр radix встановлений рівним нулю, функція буде автоматично розпізнавати і перетворювати числа, використовуючи стандартні правила форматування мови С, тобто десяткові числа повинні починатися з цифри, вісімкові – з літери О, а шістнадцяткові – з префікса Ох чи ОХ.

Функція аналогічна `atoi()`, `atof()`, `atol()`, `printf()`, `sprintf()`, `strtoul()`.

Алгоритм розв'язання задачі наведено на рис. 6.1.

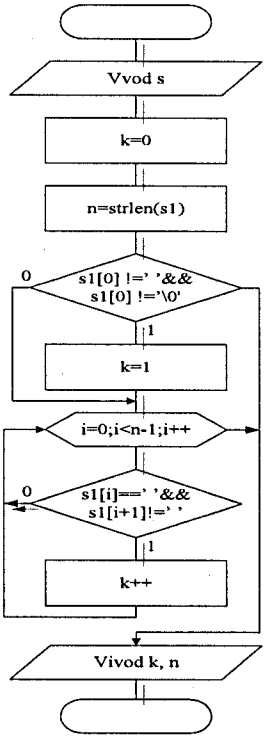


Рисунок 6.1 – Логічна схема програми визначення кількості слів у тексті

Розробка тексту програми

```
#include <stdio.h>
#include <ctype.h>
#include <conio.h>
#include <string.h>
#include <stdlib.h>
#include <dos.h>
int main()
{
   char dod[1],s1[80];
   int i,n,k=0;
   char choice,vihid;
   float bac;
   clrscr();bac=0;
   do
    {   clrscr();
    printf("  MENU\n");
    printf("1.- Vvedennya danih\n");
    printf("2.- Rishennya zadachi\n");
```

```
   printf("3.- vivedennya danih\n");
   printf("4.- Zavershennya roboti\n");
   scanf("%d",&choice);
   switch(choice)
   {
   case 1:
  {
   clrscr();
   printf("Vi vibrali punkt -vvedennya danih\n");
  printf("vveditye recennya:\n");
  gets(dod);
  gets(s1);
  bac++;
}
    break;
    case 2:
{   clrscr();
if (!(bac>0))
 {
  printf("viberity pershiy punkt i vvedit zminni\n");
  getch();
  break;
 }
 printf("znahodgennya kilkosti sliv\n");
if (s1[0] != ' '&&s1[0] != '\0') k=1;
n=strlen(s1);
for(i=0;i<n-1;i++)
if (s1[i]==' '&&s1[i+1]!=' ')
k++;
printf("Virishuyu zadachu →");
 for(i=0;i<10;i++)
 {
            printf(«O»);
   delay(500);
 }
 printf("\n zadachu uspishno virisheno!!!\n");
 getch();
 k=k;
   break;
   }
   case 3:
   {
     clrscr();
    printf("vivod danih\n");
    printf("string  s=%s\n",s1);     k=k;
    printf("dovgina l=%d\n",strlen(s1)); k=k;
    printf("kilkisty sliv k=%d\n",k);
   break;
   }
   case 4:
   printf(«vi vibrali vihid\n»);
   break;
```

```
    default:
     printf("\n\n Illegal choice!!!/n");
    }
    if (choice==1)
    printf("Chi bagayete rozvyazati zadachu?\n");
    else
    printf("chi bagayete dali prachuvati?\n");
    printf("1-Tak    2-Ni\n");
    scanf("%d",&vihid);
 }
    while(vihid==1);
  if (vihid==2)
      printf("Bagayemo uspihu!\n");
  else
      printf("\n\n Illegal choice!!!\n");
  getch();
  return choice;
}
```

На рис. 6.2 наведено алгоритм другого способу розв'язання задачі визначення кількості слів у тексті.

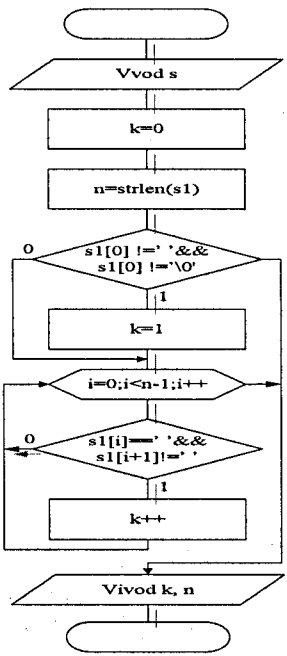


Рисунок 6.2 – Логічна схема програми визначення кількості слів у тексті

Текст такої програми має вигляд, наведений нижче.

```
#include <stdio.h>
#include <ctype.h>
#include <conio.h>
#include <string.h>
#include <stdlib.h>
#include <dos.h>
int main()
{
   char dod[1],s1[80];
   int i,n,k=0;
   char choice,vihid;
   float bac;
   clrscr();bac=0;
  do
     {   clrscr();
    printf("  MENU\n");
    printf("1.- Vvedennya danih\n");
    printf("2.- Rishennya zadachi\n");
    printf("3.- vivedennya danih\n");
    printf("4.- Zavershennya roboti\n");
    scanf("%d",&choice);
    switch(choice)
    {
    case 1:
       {
        clrscr();
       printf("Vi vibrali punkt -vvedennya danih\n");
  printf("vveditye recennya:\n");
  gets(dod);
  gets(s1);
  bac++;
      }
     break;
     case 2:
  {   clrscr();
  if (!(bac>0))
   {
    printf("viberity pershiy punkt i vvedit zminni\n");
    getch();
    break;
   }
   printf("znahodgennya kilkosti sliv\n");
    if (s1[0] != ' '&&s1[0] != '\0') k=1;
    n=strlen(s1);
    for(i=0;i<n-1;i++)
      if (s1[i]==' '&&s1[i+1]!=' ')
      k++;
      printf("Virishuyu zadachu -->");
   for(i=0;i<10;i++)
   { printf("O");
     delay(500);
```

```
    }
    printf("\n zadachu uspishno virisheno!!!\n");
    getch();
    k=k;
      break;
      }
      case 3:
      {
        clrscr();
       printf("vivod danih\n");
       printf("string  s=%s\n",s1);     k=k;
       printf("dovgina l=%d\n",strlen(s1)); k=k;
       printf("kilkisty sliv k=%d\n",k);
      break;
      }
      case 4:
     printf("vi vibrali vihid\n");
      break;
      default:
       printf("\n\n Illegal choice!!!/n");
    }
    if (choice==1)
    printf("Chi bagayete rozvyazati zadachu?\n");
    else
    printf("chi bagayete dali prachuvati?\n");
    printf("1-Tak    2-Ni\n");
    scanf("%d",&vihid);
 }
    while(vihid==1);
  if (vihid==2)
      printf("Bagayemo uspihu!\n");
      else
      printf("\n\n Illegal choice!!!\n");
 getch();
return choice;
}
```

6.5 Функції для обробки текстової інформації

Приклади розробки функцій для обробки текстової інформації

Приклад 6.13.

Функція обчислення довжини рядка. Наведена нижче функція має старомодну форму, що не рекомендується стандартом заголовка (у стандартній бібліотеці їй відповідає функція `strlen( )`):

```
int len(e)
char e[ ] ;
{
int m;
for    (m=0;    e[m]!='\0';    m++) ;
```

```
return m;
 }
```

Відповідно до ANSI-стандарту та стандарту ISO заголовок повинен мати, наприклад, такий вигляд:

```
int len   (char e[   ])
```

У цьому прикладі й у заголовку, і в тілі функції немає навіть згадувань про споріднення масивів і вказівників. Однак компілятор завжди сприймає масив як вказівник на його початок, а індекс – як зсув відносно початку масиву. Наступний варіант тієї ж функції (тепер заголовок відповідає стандартам мови) явно реалізує механізм роботи з вказівниками:

```
int len (char *s)
{
int m;
for (m=0; *s++! = '\0' ; m++)
return m;
 }
```

Для формального параметра-вказівника ***s*** всередині функції виділяється ділянка пам'яті, куди записується значення фактичного параметра. Оскільки s не константа, то значення цього вказівника може змінюватися. Саме тому припустимо вираз s++

Приклад 6.14.

Функція інвертування рядка-аргументу з параметром-масивом (заголовок відповідає стандарту):

```
void invert(char e[ ])
{
char s;
int i, j , m;
/*m – номер позиції символу '\0' у рядку e */
for (m=0; e[m]!='\0'; m++) ;
for (i=0, j=m-1; i<j; i++, j--)
{
                    s=e[i];
e[i]=e[j];
e[j]=s;
               }
  }
```

У визначенні функції `invert()` за допомогою ключового слова `void` зазначено, що функція не повертає значення.

Як вправу можна переписати функцію `invert()`, замінивши параметр-масив параметром-вказівником типу `char*`.

При виконанні функції `invert()` рядок – фактичний параметр функції, наприклад, ***«сирop»*** перетвориться в рядок «порис». При цьому символ `'\0'` залишається на своєму місці в кінці рядка. Приклад використання функції `invert()`:

```
#include <stdio.h>
void main( )
```

```
{
char ct[ ]="0123456789";
/* Прототип функції: */
void invert(char [ ]);
/* Виклик функції: */
invert(ct)/
printf ("\n%s",ct) ;
}
```

Результат виконання програми:

9876543210

Приклад 6.15.

Функція пошуку в рядку найближчого зліва входження іншого рядка (у стандартній бібліотеці є подібна функція `strstr()`)

```
/*Пошук рядка CT2 у рядку CT1 */
 int index (char * CT1, char * CT2)
   {
int i, j , m1, m2;
/* Обчислюються m1 і m2 – довжини рядків */
for (m1=0; CT1[m1] !='\0'; m1++);
for (m2=0; CT2[m2] !='\0'; m2++);
if (m2>m1)
return -1;
for (i=0; i<=m1-m2; i++)
 {
  for (j=0; j<m2; j++) /*Цикл порівняння */
  if (CT2[j] !=CT1[i+j])
 break;
 if (j==m2)
return i;
} /* Кінець циклу за i */
return -1;
}
```

Функція `index()` повертає номер позиції, починаючи з якої `CT2` цілком збігається з частиною рядка `CT1`. Якщо рядок `CT2` не входить у `CT1`, то повертається значення `-1`.

Приклад звертання до функції `index( )`:

```
#include <stdio.h>
void main( )
{
 char  C1[   ]="сума мас";
/*   Прототип  функції:   */
int  index(char [],   char []);
char C2[]="ма"/
char  C3[]="ам";
printf("\nДля  %s індекс=%d",C2,index(C1,C2) );
  printf("\nДля       %s       індекс=%d",C3,index(C1,C3));
     }
```

Результат виконання програми:
Для ма індекс=3 Для ам індекс=-1

У функції `index()` параметри специфіковані як вказівники на тип `char`, а в тілі функції звертання до символів рядків виконується за допомогою індексованих змінних. Замість параметрів-вказівників підставляють як фактичні параметри імена символьних масивів `C1[ ], C2[], C3[]`. Ніякої неточності тут немає: для масивів припустимі обидві форми звертання – і за допомогою індексованих змінних, і з використанням розіменованих вказівників.

Приклад 6.16.

Функція порівняння рядків. Для порівняння двох рядків можна написати нижченаведену функцію (у стандартній бібліотеці є близька до цієї функція `strcmp()`):

```
int  row(char C1[],   char  C2[])
{
int i,m1,m2; /* m1,m2 - довжини рядків C1,C2 */
for (m1=0;*(C1+m1)= '\0'; m1++) ;
for (m2=0;*(C2+m2)= '\0'; m2++);
if (m1!=m2) return -1;
for (i=0; i<m1; i++)
   if (*C1++ != *C2++) return (i+1);
return 0;
 }
```

У тілі функції звертання до елементів масивів-рядків реалізовано через розіменування вказівників. Функція `row()` повертає: значення -1, якщо довжини рядків-аргументів `C1, C2` різні; 0 – якщо всі символи рядків збігаються. Якщо довжини рядків однакові, але символи не збігаються, то повертається порядковий номер (ліворуч) перших незбіжних символів.

Особливість функції `row()` – специфікація параметрів як масивів і звертання до елементів масивів всередині тіла функції за допомогою розіменування. При цьому за рахунок операцій `C1++` і `C2++` змінюються початкові «настроювання» вказівників на масиви. Одночасно в тій же функції до тих же масивів-параметрів виконується звертання і за допомогою виразів `*(C1+m1)` і `*(C2+m2)`, при обчисленні яких значення `C1` і `C2` не змінюються.

Приклад 6.17.

Функція з'єднання рядків. Подана нижче функція дозволяє «приєднати» до першого рядка-аргументу другий рядок-аргумент (у стандартній бібліотеці є подібна функція : `strncat()`):

/* З'єднання (конкатенація) двох рядків: */

```
  void cone(char *C1, char *C2)
  {
int i,m;  /* m - довжина 1-го рядка  */
for (m=0; *(C1+m)!='\0'; m++);
for (i=0; *(C2+i)!='\0'; i++)
*(C1+m+i)=*(C2+i);
*(C1+m+i)='\0';
  }
```

Результат повертається як значення першого аргументу C1. Другий аргумент C2 не змінюється. Зверніть увагу на те, що при використанні функції conc() довжина рядка, що заміняє параметр C1, повинна бути достатньої для прийому підсумкового рядка.

Приклад 6.18.

Функція виділення підрядка. Для виділення з рядка C1 фрагмента заданої довжини (підрядка) можна запропонувати таку функцію:

```
void substr(char *C1, char *C2, int n, int k)
 /* C1 – вихідний рядок */
 /* C2 – підрядок, що виділяється */
 /* n – початок підрядка, що виділяється */
 /* k – довжина підрядка, що виділяється*/
{
   int i,m;      /*  m - довжина вихідного   рядка  */
   for (m=0; C1 [m] ! =' \0 ' ;  m++) ;
   if (n<0  || n>m || k<0 || k>m-n)
     {
      C2[0]='\0';
      return;
     }
   for (i=n; i<k+n; i++)
     C2[i-n]=C1[i-1];
   C2[i-n]='\0';
   return;
  }
```

Результат виконання функції – рядок C2[ ] з k символів, виділених з рядка C1[ ], починаючи з символу, що має номер n. При неправильному поєднанні значень параметрів повертається порожній рядок, використаний як параметр C2.

Приклад 6.19.

Функція копіювання вмісту рядка. Оскільки в мові С відсутній оператор присвоювання для рядків, то корисно мати спеціальну функцію, що дозволяє «переносити» вміст рядка в інший рядок (така функція strcpy() є в стандартній бібліотеці, але вона має інше значення, що повертається):

```
/*  Копіювання вмісту рядка  C2 в C1  */
void copy(char  *C1,   char  *C2) /* C2 – оригінал, C1 – копія  */
{
```

```
int  i;
for    (i=0;    C2[i]!='\0';   i++)
 C1[i]=C2[i];
C1[i]='\0';
}
```

Розробка алгоритму розв'язання

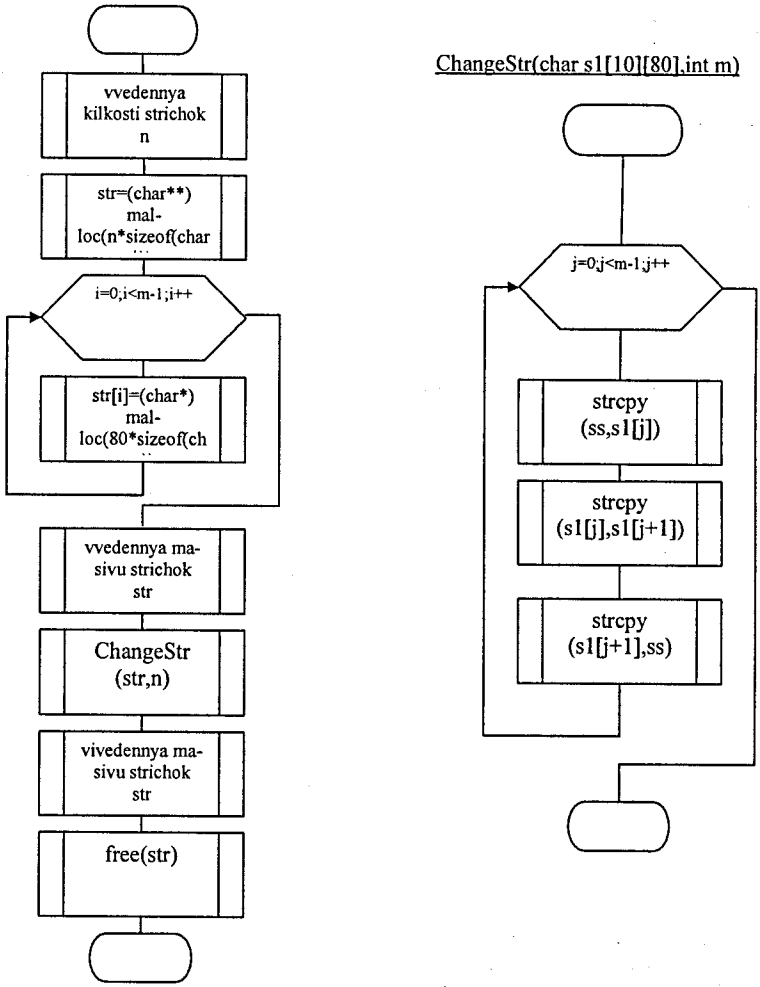


Рисунок 6.3 – Логічна схема програми, яка міняє і-й рядок з і+1

Текст такої програми має вигляд, наведений нижче.

```
#include <stdio.h>
```

```
#include <string.h>
#include <stdlib.h>
#include <conio.h>
#include <alloc.h>
void ChangeStr(char **,int);
int main()
{
 char s[1],**str;
 int n,i;
 clrscr();
 while (1)
 {
  printf("\n vveditye kilkisty strichok n=");
  scanf("%d",&n);
  if (n>=1&&n<=10)
  break;
  printf("\n ERRORS!!! 1<=n<=10 \n");
 }
 str=(char**)malloc(n*sizeof(char*));
 for(i=0;i<n;i++)
 str[i]=(char*)malloc(80*sizeof(char));
 printf("vvedity %d strok \n",n);
 gets(s);
 for(i=0;i<n;i++)
{printf("string[%d]=",i+1);
  gets(str[i]);
 }
 printf("\n-----------pislya  zminu  strichok  mayemo  takuy
masiv----------\n");
ChangeStr(str,n);
 for(i=0;i<n;i++)
            {
  printf("string[%d]=",i+1);
  puts(str[i]);
 }
free(str);
 getch();
return 0;
}
void ChangeStr(char **s1,int m)
 {
  char ss[80];
  int j;
  for (j=0;j<m-1;j++)
  {
   strcpy(ss,s1[j]);
//   puts(ss);
   strcpy(s1[j],s1[j+1]);
   strcpy(s1[j+1],ss); } }
```

6.6 Контрольні запитання

1. Які бібліотеки необхідно підключати для роботи з рядками?
2. Які є способи ініціалізації рядків?
3. За допомогою яких функцій відбувається введення символів?
4. Як відбувається введення рядків, які функції бібліотеки **string.h** при цьому використовуються?
5. За допомогою яких стандартних функцій відбувається введення рядків?
6. За допомогою яких стандартних функцій відбувається виведення рядків?
7. Які стандартні функції використовуються для визначення довжини рядків?
8. Як відбувається пошук в рядку входження підрядків?
9. Напишіть, які ви знаєте стандартні функції для роботи з рядками?
10. Яка стандартна функція з бібліотеки string.h призначена для копіювання рядків? Навести приклад.
11. За допомогою якої стандартної функції відбувається інвертування рядка?
12. Як використати рядок як параметри функцій? Навести приклад.
13. Яка різниця між функцією `strcmpi()` і функцією `strcmp()`?

6.7 Практикум з програмування

1. Написати фрагмент програми для визначення кількості цифр в рядку, введеному з клавіатури.
2. Написати фрагмент програми для інвертування заданого рядка.
3. Написати фрагмент програми для визначення кількості слів у тексті, що вводиться з клавіатури.
4. Написати фрагмент програми для поділу рядка на підрядки, довжиною 5 символів кожен, не враховуючи пропуск.
5. Написати фрагмент програми для копіювання з заданого тексту даної частини в рядок.
6. Написати фрагмент програми для злиття заданих рядків з тексту, що вводиться з клавіатури.
7. Написати фрагмент програми для визначення рядка максимальної довжини в тексті, що вводиться з клавіатури.
8. Написати фрагмент програми для пошуку в рядку `STR` кількості входжень підрядка `STR1`.
9. Написати фрагмент програми для знаходження кількості однакових символів у слові, що вводиться з клавіатури.
10. Написати фрагмент програми, яка визначає максимальну і мінімальну довжини рядків в масиві і міняє їх місцями.

7 РОБОТА З БАЗАМИ ДАНИХ

7.1 Загальні відомості

Відомі два підходи до організації інформаційних масивів: файлова організація та організація у вигляді бази даних. Файлова організація передбачає спеціалізацію та збереження інформації, орієнтованої, як правило, на одну прикладну задачу, та забезпечується прикладним програмістом. Така організація дозволяє досягнути високої швидкості обробки інформації, але характеризується рядом недоліків.

Характерна риса файлового підходу – вузька спеціалізація як обробних програм, так і файлів даних, що служить причиною великої надлишковості, тому що одні й ті самі елементи даних зберігаються в різних системах. Оскільки керування здійснюється різними особами (групами осіб), відсутня можливість виявлення порушення суперечливості збережувальної інформації. Розроблені файли для спеціалізованих прикладних програм не можна використовувати для задоволення запитів користувачів, які перекривають дві і більше ділянки. Крім того, файлова організація даних внаслідок відмінностей структури записів і форматів передання даних не забезпечує виконання багатьох інформаційних запитів навіть у тих випадках, коли всі необхідні елементи даних містяться в наявних файлах. Тому виникає необхідність відокремити дані від їхнього опису, визначити таку організацію збереження даних з урахуванням існуючих зв'язків між ними, яка б дозволила використовувати ці дані одночасно для багатьох застосувань. Вказані причини зумовили появу баз даних.

База даних може бути визначена як структурна сукупність даних, що підтримуються в активному стані та відображають властивості об'єктів зовнішнього (реального) світу. В базі даних містяться не тільки дані, але й описи даних, і тому інформація про форму зберігання вже не прихована в сполученні «файл-програма», вона явним чином декларується в базі.

База даних орієнтована на інтегровані запити, а не на одну програму, як у випадку файлового підходу, і використовується для інформаційних потреб багатьох користувачів. В зв'язку з цим бази даних дозволяють значною мірою скоротити надлишковість інформації. Перехід від структури БД до потрібної структури в програмі користувача відбувається автоматично за допомогою систем управління базами даних (СУБД).

Основними та невід'ємними властивостями БД є такі:

- для даних допускається така мінімальна надлишковість, яка сприяє їх оптимальному використанню в одному чи кількох застосуваннях;
- незалежність даних від програм;
- для пошуку та зміни даних використовуються спільні механізми;
- як правило, у складі БД існують засоби для підтримки її цілісності та захисту від неавторизованого доступу.

Основні принципи створення БД: цілісність, вірогідність, контроль, захист від несанкціонованого доступу, єдність і гнучкість, стандартизація та уніфікація, адаптивність, мінімізація введення і виведення інформації (однократність введення інформації, принцип введення – виведення тільки змін).

7.2 Особливості розробки програм для роботи з базами даних

Програми для роботи з базами даних (*DB – data base*) є частиною програмних комплексів автоматизованих систем керування технологічними процесами, підприємствами, деякими складними об'єктами. Однією з головних особливостей таких програм є організація збереження, відновлення, обробки великих обсягів інформації, тому при розробці таких програм необхідно розв'язати певне коло задач. До них належать:

1. Визначити кінцеву мету, тобто які задачі необхідно розв'язувати за допомогою цих програм, а так само – яку інформацію й у якому вигляді можна одержати з їх допомогою;
2. Визначити набір функцій, які повинна виконувати програма. Наприклад, за допомогою програми необхідно:
   - створити файли бази даних (тобто інформаційне сховище),
   - «накачати» даними файли бази даних,
   - дозаписувати дані в файли бази даних,
   - видаляти деякі дані з файлу бази даних,
   - редагувати дані в файлі бази даних (корегування даних),
   - сортувати дані в файлах бази даних за заданими критеріями;
3. Визначити набір вихідної інформації (вхідні дані), необхідний для розв'язання поставлених задач;
4. Визначити кількість файлів у базі даних, об'єднавши інформацію в групи за обраними критеріями і спроектувати базу даних;
5. Розробити структуру програми керування, помістивши до неї основний клас задач, що їх необхідно розв'язати;
6. Розробити ієрархічну структуру підзадач, деталізувавши її до рівня програмних функцій.

Наприклад, найпростіша програма для роботи з базою даних може працювати з такими задачами:

– Задача «Робота з базою даних»;

– Задача «Формування звітних документів за заданими критеріями».

Задача «Робота з базою даних» може містити такі підзадачі:

- створення бази даних;
- дозапис у базу даних;
- «накачування» даних в файли бази даних;
- редагування записів в файлах бази даних;
- видалення запису з файлу бази даних.

7.3 Приклад проектування бази даних

У мові С компоненти файлу бази даних зручно представляти у вигляді структур (struct). Тобто кожен запис є елементом структурного типу даних із членами, що є рядками полів фіксованого розміру, цілі, дійсні значення й ін. Усе, що можна записати в структурі, можна записати в двійковий файл із довільним доступом і прочитати з нього.

Проектування баз даних

Першим кроком у написанні програми створення, обробки, ведення бази даних є розробка структури запису файлу. Наприклад, для задачі «Телефонний абонент» структура запису основного файлу може мати вигляд:

```
Typedef struct
  { char name [NAMELEN];
    char addr [ADDRLEN];
    char csz [CSZLEN];
    char phone [PHONELEN];
    double balance;
  } Record;
```

Другим кроком є планування набору функцій з ведення бази даних і розробка їх прототипів. З прототипів функцій зручно створити заголовний файл, наприклад, **db.h (database)**, а потім у файлі db.c скомпонувати всі ці функції, що будуть викликатися функцією main()

```
/* Заголовний файл для роботи з базою даних  db.h  */
#include <stdio.h>
#include <limits.h>

#define FALSE 0
#define TRUE 1
#define NAMELEN 30
#define ADDRLEN 30
#define CSZLEN 30
#define PHONELEN 13

typedef struct record{
  union{
       long numrecs;    /* Заголовок==число записів  */
       long custnum;    /*Запис==номер клієнта  */
  }info;
  char name[NAMELEN];    /* ПІБ клієнта  */
  char addr[ADDRLEN];    /*Адреса клієнта  */
  char csz[CSZLEN];      /* Місто, штат клієнта  */
  char phone[PHONELEN]; /*Телефон клієнта  */
  double balance;        /* Поточний залишок  */
```

```
  }Record;
#define MAXREC (LONG_MAX / sizeof(Record))

// Створення нової бази даних з ім'ям path. Повертає FILE чи NULL і
// header
int CreateDB(const char *path);

// Відкриває базу даних з ім'ям path. Повертає FILE чи NULL і header
FILE *OpenDB(const char *path, Record *header);

// Читає запис з номером recnum у rec. Повертає TRUE/FALSE
int ReadRecord(FILE *f, long recnum, Record *recp);

// Записує запис з номером recnum з rec. Повертає TRUE/FALSE
int WriteRecord(FILE *f, long recnum, Record *recp);

// Додає запис у кінець файлу. Повертає TRUE/FALSE
int AppendRecord(FILE *f, long recnum, Record *recp);

// Відображає мітку і вводить символьний рядок
void InputLong(const char *label, long *lp);

// Відображає мітку і вводить значення типу double
void InputDouble(const char *label, double *dp);
void Inputchar(const char *label, char *cp, int len);
```

У структурі `Record` використовується об'єднання `union`, що допомагає організувати базу даних. Поля об'єднання запам'ятовуються одне поверх іншого, тобто не зважаючи на те, що оголошено в `union` два елементи типу `long`, в один і той же час може використовуватися тільки одне з них. Призначення об'єднання полягає в реєстрації номерів записів у файлі бази даних. Запис з нульовим номером запам'ятовує цю інформацію, використовуючи в структурі поле `numrecs` (і ніяке інше).

Записи з номерами 1 і вище містять значущу інформацію – у даному випадку, номер клієнта, його ім'я, адреса й інші дані.

Розглянемо організацію цього файлу.

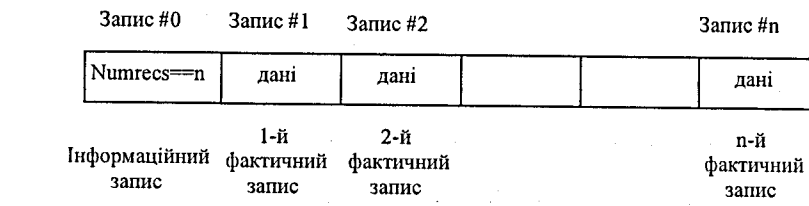


Рисунок 7.1 – Приклад структури запису файлу бази даних і запис інформаційного заголовка

//Файл **db.c**, що містить опис функцій, описаних у **db.h**

```
#include<stdio.h>
#include<string.h>
#include <stdlib.h>
#include "db.h"

//  **********************
//
int CreateDB(const char *path)
{
  FILE *f;
  Record rec;
  int result;
  f=fopen(path,"wb");     //створення нового файлу БД
  if (!f)
       return FALSE;
  memset(&rec,0,sizeof(Record));
  rec.info.numrecs=0; //початкове значення запису numrec=0, тому що поки що БД ще не містить ніяких записів
  result=WriteRecord(f,0,&rec);
  fclose(f);
 return result;
}
FILE *OpenDB(const char *path, Record *header)
{
  FILE *f=fopen(path,"r+b"); //відкриття файлу f
  if (f)
       ReadRecord(f,0, header); //читання інформаційного запису файлу, визначаючи кількість записів, що містяться у файлі
  return f;
}
//Читання запису з номером recnum  з файлу f у рядок Record, що адресується вказівником *recp.
int ReadRecord(FILE *f, long recnum, Record *recp) //
{
       if(recnum>MAXREC)
       return FALSE;
   if(fseek(f,recnum      *      sizeof(Record),SEEK_SET)!=0) //переміщення вказівника в позицію
       return FALSE;
  return(fread(recp,sizeof(Record),1,f)==1);
}
//передає на диск структуру  запису, що адресується вказівником recp.
{
int WriteRecord(FILE *f, long recnum, Record *recp)
  if (recnum > MAXREC)
       return FALSE;
 if(fseek(f,recnum*            sizeof(Record),SEEK_SET)!=0) //позиціювання вказівника
       return FALSE;
  return(fwrite(recp,sizeof(Record),1,f)==1);
```

```
}
int AppendRecord(FILE *f, long *recnum, Record *recp)
{
  if (fseek(f,0, SEEK_END)!=0) // встановлення файлового вказівника в кінець файлу
        return FALSE;
  *recnum = ftell(f) / sizeof(Record); // ftell(f) повертає значення вказівника у середині файлу
//визначення номера запису, що додається
  return WriteRecord(f, *recnum, recp); //дозапис у кінець файлу
}

void InputLong(const char *label, long *lp)
{
  char buffer[128];
  printf(label);
  gets(buffer);
  *lp=atol(buffer);
}

void InputChar(const char *label, char *cp, int len)
{
  char buffer[128];
  printf(label);
  gets(buffer);
  strncpy(cp,buffer,len-1);
}

void InputDouble(const char*label, double *dp)
{
  char buffer[128];
    printf(label);
  gets(buffer);
  *dp=atof(buffer);
}
```

Створення файлів обробки бази даних

Після розробки структури файлу бази даних і функцій запису даних у файл, необхідно розробити програму створення бази даних (тобто створити порожній файл бази даних).

Розглянемо приклад програми для розв'язання цієї задачі. Оформимо її в окремий файл **MAKEDB.C.** Така програма обов'язково повинна містити заголовний файл **db.h,** що повідомляє про структуру файлу бази даних і містить прототипи функцій, описаних у модулі **db.c.** Щоб створити закінчену програму, необхідно:

1) скомпілювати **MAKEDB.C.** і **db.c;**

2) скомпонувати об'єктні файли програм **MAKEDB.C** і **db.c** у закінчений виконуваний файл **MAKEDB.EXE**

// Програма **MAKEDB.C**

```
#include <stdlib.h>
#include <string.h>
#include "db.h"
// Створення порожнього файлу бази даних
int FileExists(const char *path);
int main()
 {
   char path[128];
   puts("Create new database file");
   printf("Filename= ");
   gets(path);
   if (strlen(path)==0)
        exit(0);
   if (FileExists(path))
   {
        printf("%s ****  already exists. \n", path);
        puts("Delete file and try again");
        exit(1);
       }
   if (!CreateDB(path))     //якщо файл існує, то програма не перезаписує його
              perror(path);
   else
        printf("%s ***  created. \n",path);
   return 0;
 }
```

`int FileExists(const char path)` //допоміжна функція, що повертає «істину», якщо заданий файл, обумовлений вказівником path типу char*, існує

```
 {
   FILE *f = fopen(path, "r");
   if (!f)
        return FALSE;
   else
   {
        fclose(f);
        return TRUE;
 }
 }
```

// Програма додавання записів у БД
// файл ADDREC.c

```
#include <stdlib.h>
#include <string.h>
#include <memory.h>
```

```
#include "db.h"

void GetNewRecord(Record *recp);
int main()
{
  char path[128];
  FILE *dbf;
  Record rec;
  long numrecs;
  //
  printf(" *********  Database name =  ");
  gets(path);
  dbf= OpenDB(path,&rec);
  if (!dbf)
  {
       printf(" ********* Can't open  %s  file \n", path);
       exit(1);
  }
  numrecs= rec.info.numrecs;
  printf("**********  Number of records ==%lu\n",numrecs);
    GetNewRecord(&rec);
  if (AppendRecord(dbf, &numrecs, &rec))
       printf(" ***********  Record  #%lu  added  no  database
\n", numrecs);
  memset(&rec,0,sizeof(Record));
  rec.info.numrecs = numrecs;
  WriteRecord(dbf,0,&rec);
  fclose(dbf);
  return 0;
}

void GetNewRecord(Record *recp);
{
  memset(recp,0,sizeof(Record));
  InputLong("Customer # : ", &recp->info.custnum);
  InputChar("Name :", recp->name, NAMELEN);
          InputChar("Address :", recp->addr, ADDRLEN);
  InputChar("Sity, State, Zip :", recp->csz, CSZLEN);
  InputChar("Telephone :", recp->phone, PHONELEN);
  InputDouble("ACCOUNT balance :", recp->balance);
}
```

Дана програма додає тільки 1 запис у файл. Якщо необхідно дозаписати певну кількість записів, то створюється цикл за кількістю записів, наприклад, після введення кожного запису запитувати користувача, чи не хоче він додати ще один запис.

Більшість програм роботи з базами даних охоплює крім введення нових записів, процедури редагування (коректування, змінювання) вже існуючих у файлі певних записів, не змінюючи інші. Алгоритм такої програми наведений на рисунку 7.2.

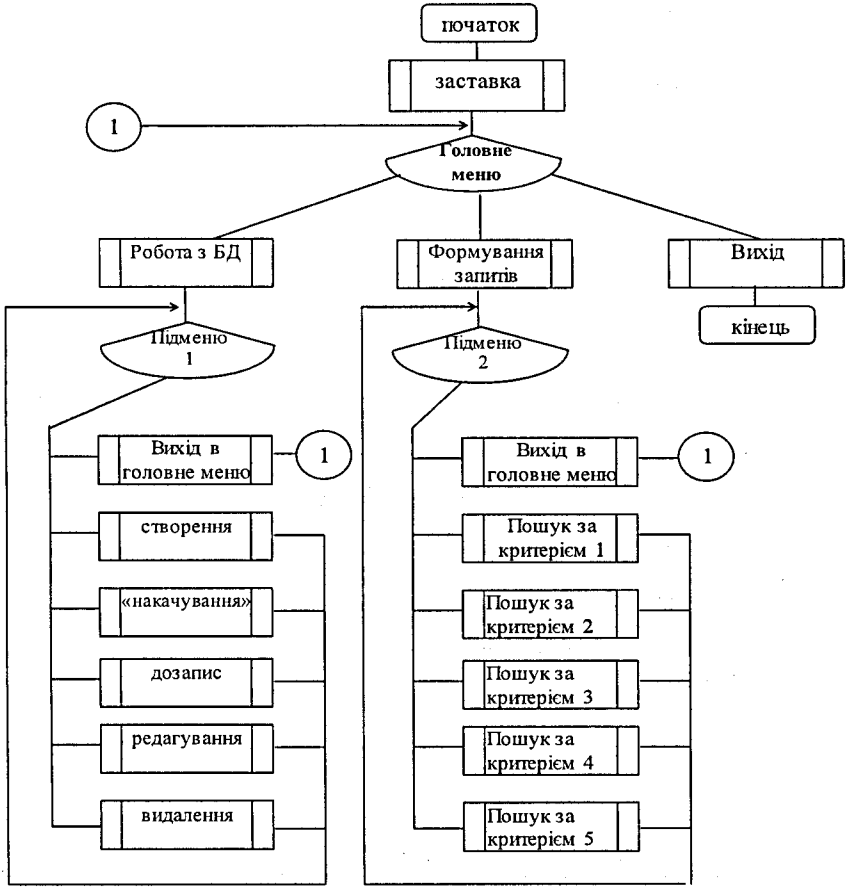


Рисунок 7.2 – Алгоритм програми роботи з базою даних

У запропонованому нижче прикладі програми використовуються методи довільного доступу після введення номера запису для редагування і після цього відбувається відновлення інформації, змушуючи користувача повторно вводити всі поля обраного запису. Більшість програмістів таке редагування називають недружелюбним.

```
// Приклад програми редагування записів у файлі БД
// файл EDIREC.c
#include <stdlib.h>
#include <memory.h>
#include "db.h"
void EditRecord(FILE *f,long recnum, Record *recp);
```

```
int main()
{
  char path[128];
  FILE *dbf;
  Record rec;
  long numrecs, recnum;
  printf("************ Database file name =");
  gets(path);
  dbf = OpenDB(path,&rec);
  if (!dbf)
  {
       printf(" ******** Can't open %s   file \n", path);
       exit(1);
  }
    numrecs = rec.info.numrecs;
  printf(" ****** Number of records  == %lu \n",numrecs);
  InputLong("*** Record number ?",&recnum);
  if ((recnum<=0) || (recnum>numrecs))
  {
       puts("**** Record number  out of range");
       fclose(dbf);
       exit(1);
  }
  EditRecord(dbf, recnum, &rec);
  if (WriteRecord(dbf, recnum, &rec))
       printf("**** Record namber #%lu   writted to &s\n",
recnum, path);
  else
       perror("Write record");
  fclose(dbf);
  return 0;
}

void EditRecord(FILE *f, long recnum, Record *recp)
{
  if (!ReadRecord(f,recnum, recp))
  {
       perror("Reading record");
       exit(1);
  }
  printf("***** Cstomer  # :  %lu\n",recp->info.custnum);
  InputLong("Customer  # : ",&recp->info.custnum);
  printf("**** Name : %s\n",recp->name);
  InputChar("Name : ", recp->name, NAMELEN);
  printf("**** Address : %s\n", recp->addr);
  InputChar("Adderss : ", recp->addr, ADDRLEN);
  printf("**** City, State, Zip :", recp->csz);
  InputChar(" City, State, Zip :", recp->csz, CSZLEN);
  printf("**** Telephone : %s\n", recp->phone);
  InputChar("Telephone :", recp->phone, PHONELEN);
  printf(" ** Account balance : %8.2f\n", recp->balance);
  InputDouble(" Account balance : " , &recp->balance);
```

Для роботи з базами даних можна використовувати програмне забезпечення, що характеризується певними структурами та функціями.

В першу чергу необхідно визначити типи файлів для збереження в них інформації та структури цих файлів, оскільки це спрощує та полегшує роботу з ними.

На прикладі розглянемо особливості створення структур файлів, що складають базу даних для програми автоматизації роботи диспетчера торгової фірми з отримання та виконання замовлень від клієнтів. В даному випадку необхідно розробити бази даних, які будуть містити такі таблиці:

- «Товари»;
- «Клієнти»;
- «Виконані замовлення».

Таблиця 7.1 – Опис файлів бази даних

База	Тип файлу	Назва
Товари	Бінарний	**Prodbase.dat**
Клієнти	Бінарний	**Klntbase.dat**
Виконані замовлення	Бінарний	**Cmplbase.dat**

База даних товарів у вигляді файлу *Prodbase.dat* містить у собі код, назву товару, назву фірми-виробника, характеристику товару, ціну за 1шт., кількість товару на складі та примітки. Структура файлу *Prodbase.dat* наведена в таблиці 7.2.

Таблиця 7.2 – Структура файлу **Prodbase.dat**

№ поля	Зміст поля	Найменування (на С) поля	Розмір	Тип поля
1	Код товару	code	3	char[3]
2	Назва товару	tovar	15	char[15]
3	Фірма-виробник	producer	15	char[15]
4	Характеристика товару	discription	15	char[15]
5	Ціна за 1шт.	price	5	char[5]
6	Кількість товару на складі	number	5	char[5]
7	Примітки	notes	8	char[8]

Структура файлу з програмної точки зору буде виглядати таким чином:

```
struct Product //задаємо назву типу структури
{
char code[3];
char name[15];
char producer[10];
char description[30];
char price[5];
char number[5];
char notes[10];
```

}

База даних клієнта у вигляді файлу *Klntbase.dat* містить у собі повну назву фірми-клієнта (назва, адреса та телефон), покупця (ПІБ, посада та телефон) і заборгованість клієнта фірмі АСКО-Електрик. Структура файлу *Klntbase.dat* наведена в таблиці 7.3.

Таблиця 7.3 – Структура файлу **Klntbase.dat**

№ поля	Зміст поля	Найменування (на С) поля	Розмір	Тип поля
1	Фірма	Firma	-	Struct
2	Ім'я покупця	Pokupatel	-	Struct
3	Борг	Borg	5	char[5]

Структуру файлу *Klntbase.dat* мовою С можна описати у вигляді:

```
struct Klient
{
struct  Firma firma;
struct Pokupatel pokupatel;
char borg[5];
}
```

Структури файлів *Firma.dat* та *Pokupatel.dat*, що містить дані про фірму та покупця, наведені в таблицях 7.4, 7.5.

Таблиця 7.4 – Структура **Firma**

№ поля	Зміст поля	Найменування поля	Розмір	Тип поля
1	Назва фірми	name	10	char[10]
2	Адреса	address	10	char[10]
3	Телефон фірми	phone	15	char[15]

Структуру файлу *Firma.dat* мовою С можна описати у вигляді:

```
struct Firma
{
  char name[10];
  char address[10];
  char fphone[15];
}
```

Таблиця 7.5 – Структура **Pokupatel**

№ поля	Зміст поля	Найменування поля	Розмір	Тип поля
1	Ім'я покупця	name	15	char[15]
2	Посада	posada	10	char[10]
3	Телефон покупця	phone	15	char[15]

Структуру файлу *Pokupatel.dat* мовою С можна описати у вигляді:

```
struct Pokupatel
{
  char name[15];
  char posada[10];
  char phone[15];
}
```

База даних замовлень у вигляді файлу *Cmplbase.dat* містить у собі номер замовлення, код товару, дату замовлення, фірму-клієнта, ім'я покупця, назву товару, ціну одиниці товару, кількість купленого товару, загальну ціну (суму товару), оплату клієнта. Структура файлу *Cmplbase.dat* наведена в таблиці 7.6.

Таблиця 7.6 – Структура файлу Cmplbase.dat

№ поля	Зміст поля	Найменування (на С) поля	Розмір	Тип поля
1	Номер замовлення	znomer	3	char[3]
2	Код товару	code	3	char[3]
3	Дата замовлення	date	10	char[10]
4	Фірма (клієнт)	zfirma	10	char[10]
5	Ім'я покупця	FIO	15	char[15]
6	Назва товару	tovar	15	char[15]
7	Ціна за 1 шт.	Price	5	char[5]
8	Кількість купленого	number	5	char[5]
9	Ціна загальна	allprice	5	char[5]
10	Оплата клієнта	Payment	5	char[5]
11	Заборгованість	borg	5	char[5]

Структуру файлу *Cmplbase.dat* мовою С можна описати у вигляді:

```
struct Final
{
char znomer[3];
char code[3];
char date[10];
char zfirma[10];
char FIO[15];
char tovar [15];
char Price[5] ;
char  number[5];
char allprice[5];
char Payment[5];
char borg[5];
}
```

Для роботи з базами даних насамперед необхідно розробити функції, що виконують такі операції:

- створення файлу задля збереження інформації;
- запис даних в файл;
- зчитування даних з файлу;
- видалення даних з файлу.

Всі розглянуті вище файли бази даних відрізняються лише структурами записів, тому для прикладу розглянемо лише структуру запису файлу товарів *Prodbase.dat.*

```
struct Product
{
    char code[4];          // код товару
    char tovar[15];         // назва товару
    char producer[15];      //фірма-виробник
    char discription[15]; // характеристика товару
    char price[6];          // вартість товару
    char number[6];         // кількість товару на складі
    char notes[9];         // примітка
};
```

Розглянемо основні програмні функції, необхідні для обробки даних, що знаходяться в файлі з такою структурою.

`Функція int dbf_exists(char *filename)` – призначена для перевірки існування файлу. Якщо файл не існує (тобто файловий потік NULL), то функція повертає значення 0, в іншому випадку повертає значення 1.

Код даної функції наведений нижче:

```
int dbf_exists( char *filename )
{
FILE *instream=fopen(filename,"rb");
// створення файлового потоку
//*instream для роботи з файлом
// filename, що знаходиться на дисковi
  if (instream==NULL)
       return 0;
  else
  {
       fclose(instream);
       return 1;
  }
}
```

Функція `int dbf_records(char *fname, Product *prod)` – призначена для підрахунку кількості записів структурного типу Product у файлі.

`char *fname` – назва файлу

`Product *prod` – тип структури

Код даної функції має такий вигляд:

```
int dbf_records(char *fname, Product *prod)
{
  FILE *instream=fopen(fname,"rb");
  if (instream==NULL)
       return 0;
  else
  {
  fseek(instream,0L,SEEK_END);//встановлення вказівника в кінець файлу
  int flen=ftell(instream);// визначення кількості байтів в частині файлу від його початку до вказівника
  fclose(instream);
  return flen/sizeof(*prod);// повертаємо число – кількість записів
  }
}
```

Функція `void dbf_write_struct(char *filename,Product *prod)` – призначена для запису структури типу Product у файл.

Код даної функції має такий вигляд:

```
void dbf_write_struct(char *filename,Product *prod)
{
FILE *istream=fopen(filename,"ab");
fseek(istream,0L,SEEK_END);//встановлюємо вказівник в кінець файлу
fwrite(prod,sizeof(Product),1,istream);//записуємо з джерела prod розміром Product 1-у порцію у файл
fclose(istream);
}
```

Функція `void dbf_rewrite_struct(char *filename,Product *prod,int record)` – призначена для перезапису структури за заданим номером `record` типу `Product` у файлі.

Код даної функції має такий вигляд:

```
void dbf_rewrite_struct(char *filename,Product *prod,int record)
 {
     FILE *istream=fopen(filename,"r+b");
     fseek(istream,record*sizeof(Product),SEEK_SET);
//встановлення вказівника на місце заданого запису з початку файлу
     fwrite(prod,sizeof(Product),1,istream);
```

```
        fclose(istream);
    }
```

Функція `void dbf_delete_record(char *filename, int record)` – призначена для видалення запису за заданим номером record з файлу.

Код даної функції має такий вигляд:

```
void dbf_delete_record(char *filename, int record)
 {
  FILE *instream = fopen(filename,"rb");
  FILE *onstream = fopen(temp_filename_s, "wb");
  Product prod;
  //int count=-1;
  fseek(instream,0L,SEEK_END);//встановлюємо вказівник в кінець файлу
  int flen = ftell(instream);//знаходимо розмір файлу в байтах
  fseek(instream,0L,SEEK_SET);//встановлюємо вказівник на початок файлу

  for(int i=0;i<(flen/sizeof(Product));i++)
   {
     memset(&prod,0,sizeof(Product));
     fread(&prod,sizeof(Product),1,instream);//зчитуємо з файлового потоку
  // instream в prod одну порцію даних розміром sizeof(Product)
     if (i != record)
       fwrite(&prod,sizeof(Product),1,onstream);
   }
  fclose(instream);
  fclose(onstream);
  unlink(filename);// перевідкриття файлу
 dbf_rename(temp_filename_s,filename);//перейменування файлу
 }
```

Функція `void dbf_readrecord(char *fname, int record, Product *outdata)` призначена для зчитування запису за заданим номером record з файлу в структуру типу Product.

Код даної функції має такий вигляд:

```
void dbf_readrecord(char *fname, int record, Product *outdata)
 {
    FILE *istream = fopen(fname,"rb");
    fseek(istream,record*sizeof(Product),SEEK_SET);
    read(outdata,sizeof(Product),1,istream);  // зчитування з
```

файлового потоку istream в комірку пам'яті outdata даних розміром sizeof(Product) одноразово

```
    fclose(istream);
}
```

Функція `int dbf_search(char *fname, int field,char *keyword)` призначена для пошуку в файлі необхідних записів за заданим ключовим словом і за заданим полем. Тобто дана функція зчитує всі записи з файлу, що знаходяться за даними критеріями (поле структури field, ключове слово keyword) і записує всі знайдені записи (структури) в тимчасовий файл **temp_filename_s.**

Код даної функції має такий вигляд:

```
int dbf_search(char *fname, int field,char *keyword)
{
  FILE *instream = fopen(fname ,"rb");
  FILE *tmpstream     = fopen(temp_filename_s,"wb");

  Product   prod;
  char *pr, *num;
    //обнуляємо структуру prod
  memset(&prod,0,sizeof(Product));
  //встановлюємо вказівник в кінець
  //файлу, щоб визначити розмір файлу за допомогою ftell(instream)
  fseek(instream,     0L, SEEK_END);
     //визначаємо кількість записів
  int flen = ftell(instream)/sizeof(Product);
  fseek(instream,     0L, SEEK_SET);
//змінюємо всі великі символи рядка на малі
strlwr(keyword); for (int i = 0,     count     =     0;   i   <
flen; i++)
   {
        fread(&prod,sizeof(Product),1,instream);
        //наступний код призначений для порівняння ключового слова
        //з усіма властивостями структури
        switch (field)
        {
             case 1:
             //перевірка рівності двох рядків - prod.code і keyword
             if  (strstr(strlwr(prod.code),keyword)  !=  NULL)

             {
//запис в тимчасовий файл знайденого поля
                  fwrite(&prod,sizeof(Product),1,tmpstream);
                  count++;
             }
             break;

             case 2:
             if  (strstr(strlwr(prod.tovar),keyword)  !=  NULL)
```

```
                {
//запис в тимчасовий файл знайденого поля
                    fwrite(&prod,sizeof(Product),1,tmpstream);
                    count++;
                }
                break;

                case 3:
                if   (strstr(strlwr(prod.producer),keyword)   !=
NULL)
                {
//запис в тимчасовий файл знайденого поля
                    fwrite(&prod,sizeof(Product),1,tmpstream);
                    count++;
                }
                break;
                case 4:
                if  (strstr(strlwr(prod.discription),keyword)  !=
  NULL)
                {
//запис в тимчасовий файл знайденого поля
                    fwrite(&prod,sizeof(Product),1,tmpstream);
                    count++;
                }
                break;
                case 5:
                if  (strstr(strlwr(prod.price),keyword)  != NULL)

                {
//запис в тимчасовий файл знайденого поля
                    fwrite(&prod,sizeof(Product),1,tmpstream);
                    count++;
                }
                break;
                case 6:
                if (strstr(strlwr(prod.number),keyword) != NULL)
                {
//запис в тимчасовий файл знайденого поля
                    fwrite(&prod,sizeof(Product),1,tmpstream);
                    count++;
                }
                break;
                case 7:
                if  (strstr(strlwr(prod.notes),keyword)  != NULL)
                {
//запис в тимчасовий файл знайденого поля
                    fwrite(&prod,sizeof(Product),1,tmpstream);
                    count++;
                }
                break;
```

```
        }
    }
    fclose(instream);
    fclose(tmpstream);
    return count;
}
```

Функція `int dbf_sort(char *filename, int field)` – призначена для сортування записів в файлі за алфавітом та за необхідним полем. Тобто всі записи, які відсортовані за заданим полем, записуються в динамічний масив структур product типу Product. Далі дані з цього масиву записуються в тимчасовий файл **temp_filename_s.**

Код даної функції має такий вигляд:

```
int dbf_sort(char *filename, int field)
{
        //відкриваємо файл для читання
FILE *instream = fopen(filename,"rb");
            //відкриваємо тимчасовий файл для запису
  FILE *tmpstream     = fopen(temp_filename_s,"wb");
       Product   prod;
  char *pr, *num;
  memset(&prod,0,sizeof(Product));
       //визначення довжини файлу
  fseek(instream,     0L, SEEK_END);
  int flen = ftell(instream)/sizeof(Product);
  fseek(instream,     0L, SEEK_SET);
  int kk,ff=flen;
  Product   *product;//об'являємо динамічний масив структур
Product     vrem; //dopomigna zminna
  //виділяємо для нього пам'ять розміром flen*sizeof(Product) байтів:
  product=(Product*)malloc(flen*sizeof(Product));
  //зчитування з файлу даних в масив
  fread( product,sizeof(Product),flen,instream );
  int count = 0;
       do
  {
       kk=0;
       ff--;
       count++;
       for (int i = 0 ; i < ff; i++)
       {
            switch (field)
            {

                 case 1:
//порівнюємо, який з 2-х рядків за алфавітом перший
if(strcmp(strlwr(product[i].code),
```

```
      strlwr(product[i+1].cod  e)) >0)
                    {
                         vrem=product[i];//запис  в  тимчасову
змінну
                         product[i]=product[i+1];
                         product[i+1]=vrem;
                         count++; kk++;
                    }
                    break;

                    case 2:
                    if
(strcmp(strlwr(product[i].tovar),strlwr(product[i+1].tovar))
    > 0)
                    {
                         vrem=product[i];
                         product[i]=product[i+1];
                         product[i+1]=vrem;
                         count++; kk++;
                    }
                    break;

                    case 3:
                    if
(strcmp(strlwr(product[i].producer),strlwr(product[i+1].produc
er)) >    0)
                    {
                         vrem=product[i];
                         product[i]=product[i+1];
                         product[i+1]=vrem;
                         count++; kk++;
                    }
                    break;

                    case 4:
                    if
(strcmp(strlwr(product[i].discription),strlwr(product[i+1].dis
cription)) > 0)
                    {
                         vrem=product[i];
                         product[i]=product[i+1];
                         product[i+1]=vrem;
                         count++; kk++;
                    }
                    break;

                    case 5:
                    if
(strcmp(strlwr(product[i].price),strlwr(product[i+1].price))
    > 0)
                    {
                         vrem=product[i];
```

```
                    product[i]=product[i+1];
                    product[i+1]=vrem;
                    count++; kk++;
                }
                break;

                case 6:
                if
(strcmp(strlwr(product[i].number),strlwr(product[i+1].number))
> 0)
                {
                    vrem=product[i];
                    product[i]=product[i+1];
                    product[i+1]=vrem;
                    count++; kk++;
                }
                break;

                case 7:
                if
(strcmp(strlwr(product[i].notes),strlwr(product[i+1].notes))
    > 0)
                {
                    vrem=product[i];
                    product[i]=product[i+1];
                    product[i+1]=vrem;
                    count++; kk++;
                }
                break;


            }
        }
    } while( kk > 0     );
```

//запис в тимчасовий файл даних з масиву структур

```
    fwrite(product,sizeof(Product),flen,tmpstream);

    fclose(instream);
    fclose(tmpstream);
    free(product);

    return count;
}
```

7.4 Контрольні запитання

1. Що являє собою база даних?
2. Назвіть головні особливості програм для роботи з базами даних?
3. Які задачі необхідно розв'язати при розробці програм для роботи з базами даних?
4. Назвіть основні етапи проектування бази даних?
5. Напишіть приклад структури бази даних?
6. Як створити файл бази даних?
7. Як здійснити зчитування даних з файлу?
8. Як здійснити запис даних у файл?
9. Для чого в структурі бази даних використовується об'єднання *union*?
10. Які ви знаєте функції для роботи з базою даних?
11. Як здійснити сортування записів у файлі бази даних?
12. Як підключити до програми файл бази даних?

7.5 Практикум з програмування

1. Напишіть фрагмент програми для додавання записів в базу даних.
2. Напишіть фрагмент програми для редагування записів в базі даних.
3. Напишіть фрагмент програми для видалення записів з бази даних.
4. Напишіть фрагмент програми для створення порожнього файлу бази даних.
5. Напишіть фрагмент програми для зчитування даних з файлу бази даних.
6. Напишіть фрагмент програми для запису даних у файл бази даних.
7. Напишіть фрагмент програми для сортування записів в файлі у алфавітному порядку та за необхідним полем.
8. Напишіть фрагмент програми для видалення запису за заданим номером з файлу.

8 ОСОБЛИВОСТІ СТВОРЕННЯ ГРАФІЧНОГО МЕНЮ АЛГОРИТМІЧНОЮ МОВОЮ С

8.1 Загальні відомості

Для створення графічного меню насамперед необхідно підключити бібліотеку **graphics.h** за допомогою препроцесорної директиви `#include<graphics.h>`. Також потрібно підключити стандартні бібліотеки:

```
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
#include<stdlib.h>
```

Щоб ініціалізувати графічний режим використовуємо функцію

```
void GR_init()
void GR_init()
{
//знаходимо графічний драйвер
int gdriver=DETECT,gmode,errormode;
//ініціалізуємо графічний режим
initgraph(&gdriver,&gmode,"");
}
```

Для прикладу розглянемо алгоритм створення графічного меню програми автоматизації праці диспетчера торгової фірми з отримання та виконання замовлень від клієнтів.

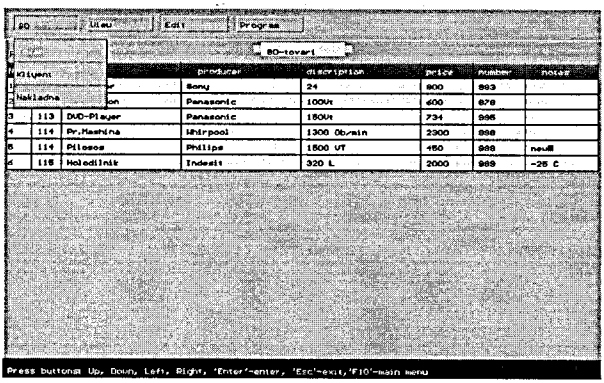


Рисунок 8.1 – Приклад вікна програми з графічним меню

8.2 Створення кнопок

Спочатку доцільно розглянути алгоритм створення кнопки та логіку зміни її стану.

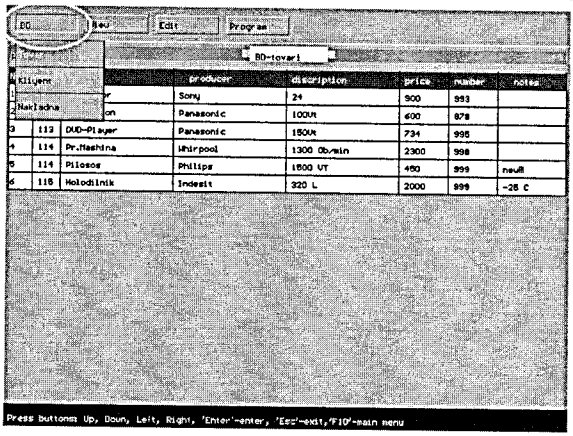


Рисунок 8.2 – Активна кнопка в меню

Кнопку можна створити за допомогою функції
`void GR_draw_button(struct menu_button menu),`
де **struct menu_button menu** – структура, в якій описані необхідні параметри кнопки. Тому розглянемо структуру **struct menu_button** :

```
struct menu_button
{
        int x0;// абсциса верхнього лівого кутка
        int y0;//ордината верхнього лівого кутка
        int x1; // абсциса нижнього правого кутка
        int y1; //ордината нижнього правого кутка
        char label[10];//назва кнопки
       int status;//стан кнопки, пасивний(0) чи активний(1)
};
```

Розглянемо саму функцію:

```
void GR_draw_button(struct menu_button menu)
{
       int x0    = menu.x0;//задаємо координати кутків кнопки
       int y0    = menu.y0;
       int x1    = menu.x1;
       int y1    = menu.y1;
       int state= menu.status;//задаємо стан кнопки
```

```
        char text_label[10]={"          "};
        strcpy(text_label,menu.label);
        setcolor(8);//задаємо колір 8 – темно-сірий колір
        rectangle(x0,y0,x1,y1);//рисуємо прямокутник за вхідними параметрами
        setfillstyle(SOLID_FILL,7); //задаємо стиль заливання, а саме: однотонний світло-сірого кольору
        floodfill((x0+x1)/2,(y0+y1)/2,8); //зафарбовуємо раніше нарисований //прямокутник нарисований кольором 8
        setcolor(0);//задаємо колір такий, як фон, оскільки фон у нас чорного кольору, //то олівець також має чорний колір
        settextstyle(SMALL_FONT,HORIZ_DIR,0);//задаємо стиль тексту
        outtextxy(x0+4,((y0+y1)/2)-5,text_label);//створюємо графічний текст //заданого стилю, тобто рисуємо назву кнопки
        setcolor(7);//задаємо колір границь кнопки
        line(x0,y1,x0,y0);//ліва вертикальна границя
        line(x0,y0,x1,y0);//верхня горизонтальна границя
        line(x0,y1,x1,y1);//нижня границя
        line(x1,y0,x1,y1);//права границя
        if(!state) //якщо кнопка неактивна, тобто не натиснута
        {setcolor(15);
         line(x0,y1,x0,y0);
         line(x0,y0,x1,y0);
         setcolor(8);
         line(x1,y0,x1,y1);
         line(x0,y1,x1,y1);
         }
      else {
// якщо кнопка активна, то рисуємо відповідно границі навпаки
         setcolor(7);
         settextstyle(SMALL_FONT,HORIZ_DIR,0);
//рисуємо текст трохи нижче, створюючи ефект об'єму, немов кнопка
//вгнулась вниз
         outtextxy(x0+4,((y0+y1)/2)-5,text_label);
//тут відповідно рисуємо границі навпаки відносно неактивного стану
         setcolor(1);
         settextstyle(SMALL_FONT,HORIZ_DIR,0);
         outtextxy(x0+6,((y0+y1)/2)-3,text_label);
         setcolor(8);
         line(x0,y1,x0,y0);
         line(x0,y0,x1,y0);
         setcolor(15);
         line(x1,y0,x1,y1);
         line(x0,y1,x1,y1);
      }
}
```

8.3 Створення графічного підменю

Розглянемо алгоритм створення графічного підменю, яке з'являється відразу після натискання клавіші Enter під активною кнопкою. Відповідно для кожної кнопки створюються різні підменю.

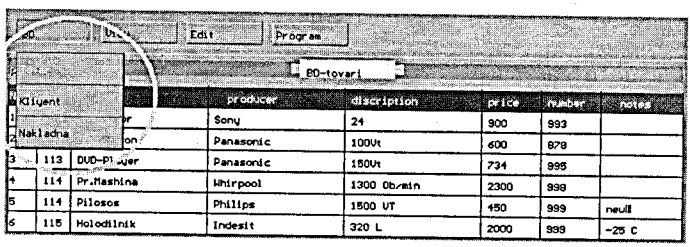


Рисунок 8.3 – Виділення графічного підменю

Підменю можна створити за допомогою функції

`void GR_draw_submenu(struct submenu smenu),`

де **struct submenu smenu** – містить структуру підменю.

```
struct submenu
{
    int items_number;//кількість пунктів в підменю
    char labels[menu_max_items][menu_max_chars];//масив назв
пунктів підменю
    int x0;//початкова абсциса підменю, лівий верхній куток підменю
    int y0;//початкова ордината підменю, лівий верхній куток підме-
ню
    int state;// стан підменю, пасивний чи активний
    int active_element;//номер активного пункту в підменю
};
```

`menu_max_items` // константа, що вказує на максимальну кількість пунктів в підменю

`menu_max_chars` // константа, що вказує на максимальний розмір назви пункту в підменю

```
void GR_draw_submenu(struct submenu smenu)
{
  setcolor(8);
  int
  hpe=30,//висота  в пікселях одного пункту підменю
  items=smenu.items_number+1,//кількість пунктів меню
  item_heigth=items*hpe,//висота в пікселях всіх пунктів меню
```

```
x0=smenu.x0,//початкова абсциса (Х-координата) верхнього лівого
кутка
y0=smenu.y0,//початкова ордината (У-координата)верхнього лівого
кутка
res_x_pos =x0+100,//права Х-координата підменю (x0+ширина)
res_y_pos =y0+item_heigth,// нижня У-координата підменю
(y0+висота)res_y0_pos=y0+30,//початкова У-координата підменю
state =smenu.state,// стан підменю, 1- активний, 0 - пасивний
element= smenu.active_element;//номер вибраного пункту підме-
ню
//якщо стан активний, тоді виконується рисування підменю
if (state)
  { //рисування прямокутника
    rectangle(x0,res_y0_pos,res_x_pos,res_y_pos);
    setfillstyle(SOLID_FILL,7);//fill style
    //зафарбовуємо прямокутник вище заданим заливанням
    floodfill(x0+1,res_y0_pos+1,8);
    //Рисуємо тінь підменю
    setcolor(0);
line(x0,res_y_pos+1,res_x_pos+1,res_y_pos+1);
line(res_x_pos+1,res_y_pos+1,res_x_pos+1,res_y0_pos+1);
    //Задаємо стиль тексту, яким ми відображаємо пункти підменю
    settextstyle(SMALL_FONT,HORIZ_DIR,0);
    //в циклі рисуємо послідовно всі пункти даного підменю.
   // step – У-координата поточного підпункту
for(int i=0,step =y0+hpe; i<items;i++,step+=hpe)
 {  setcolor(8);
    line(x0,step,res_x_pos,step);
    setcolor(0);
    char *str= smenu.labels[i];
//перевіряємо, якщо поточний пункт вибраний, то змінюємо колір тек-
сту на жовтий
    if(element==i&&element>-1&&element<=items)
    {
     setcolor(14);
     outtextxy(x0+4,step+10,str);
    }
   outtextxy(x0+4,step+10,str);
 }}
//якщо підменю неактивне, то просто зафарбовуємо все підменю ко-
льором фону
      else
        {
         setcolor(main_bgcolor);
         rectangle(x0,res_y0_pos,res_x_pos+1,res_y_pos+1);
         setfillstyle(SOLID_FILL,main_bgcolor);
         floodfill(x0+2,res_y0_pos+2,main_bgcolor);
        }}
```

При виклику підменю зазвичай вибирають необхідний пункт для того, щоб виконувати певну операцію в залежності від задачі і заданого меню. Але перед цим виконанням створюється графічне вікно, яке містить в собі базову інформацію. Так, наприклад, з початку роботи з програмою необхідно вибрати базу даних в пункті головного меню BD. Після цього на екрані з'явиться базове вікно, наведене на рисунку 8.4.

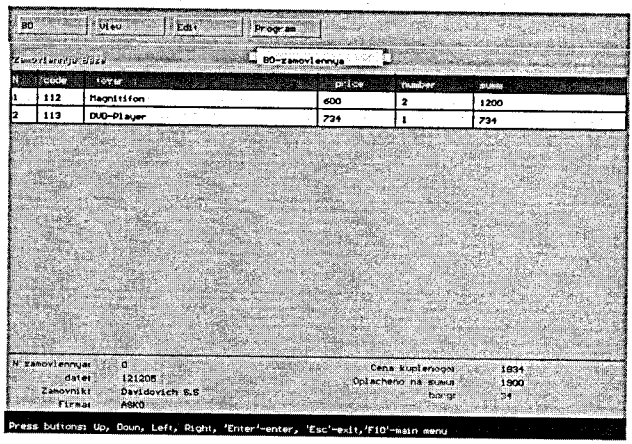


Рисунок 8.4 – Графічне вікно пункту меню BD

В залежності від вибраної бази даних в головному базовому вікні відображаються різні дані. Ці дані можуть мати певну структуру в залежності від поставленої задачі та функціональності програми, тому розглянемо лише базове вікно та функцію, за допомогою якої воно створюється, а саме: функцію `void DrawWorkWindow(int number_base,char *baza_name)`.

Вхідні параметри функції:

int number_base – номер бази даних, який служить для відтворення назви бази даних в кутку вікна і не є необхідним, був введений у зв'язку зі специфікою програмного продукту.

char *baza_name – назва операції, або назва пункту в підменю. Тобто це назва вікна, що виводиться по середині вікна

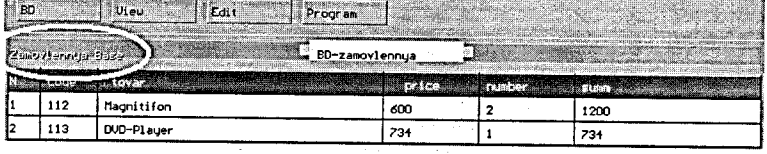


Рисунок 8.5 – Назва бази даних в меню

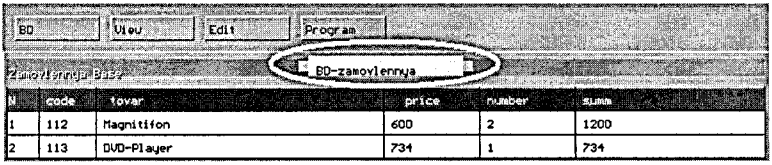


Рисунок 8.6 – Назва вікна

```
void DrawWorkWindow(int number_base,char *baza_name)
    {
      //рисуємо зафарбований прямокутник та границі вікна
setfillstyle(1,7);
int y0=40,
y1=getmaxy()-25;
bar(0,y0,getmaxx(),y1);
setcolor(15);
line(0,y0,getmaxx(),y0);
line(0,y0,0,y1);
      //створення заголовка вікна, а саме: зеленого прямокутника, жов-
того та назви вікна
setcolor(1);
setfillstyle(1,2);
bar(3,y0+6,635,y0+20);
setfillstyle(1,10);
bar(320-4*strlen(baza_name)-
16,y0+8,320+8*strlen(baza_name)/2+16,y0+16);
setfillstyle(1,14);
bar(320-4*strlen(baza_name)-
8,y0+4,320+8*strlen(baza_name)/2+8,y0+20);
setcolor(0);
outtextxy(getmaxx()/2-
8*strlen(baza_name)/2,y0+10,baza_name);
      //в залежності від номера бази відображаємо на екран відповідний
текст.
  switch(number_base)
  {
  case 1:
       setcolor(0); //задаємо чорний колір
outtextxy(6,y0+12+1,"Product Base");
       setcolor(15);//задаємо білий колір
       outtextxy(5,y0+12,"Product Base");
  break;
  case 2:
       setcolor(0); //задаємо чорний колір
       outtextxy(6,y0+12+1,"Klient Base");
       setcolor(15); //задаємо білий колір
       outtextxy(5,y0+12,"Klient Base");
  break;
  case 3:
```

```
        setcolor(0); //задаємо чорний колір
        outtextxy(6,y0+12+1,"Zamovlennya Base");
        setcolor(15); //задаємо білий колір
        outtextxy(5,y0+12,"Zamovlennya Base");
    break;
    }
}
```

8.4 Створення діалогових вікон

Для покращення та спрощення роботи з інтерфейсом програми було розроблено діалогове вікно, яке використовується як базове, наприклад, для того, щоб запитати у користувача про подальші дії.

Так, для запиту про введення інформації про новий продукт необхідно створити окреме вікно, яке показано на рисунку 8.7.

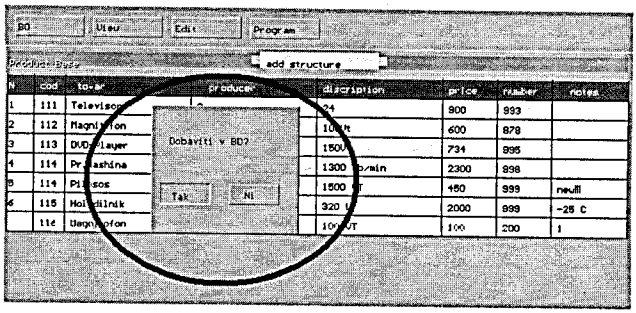


Рисунок 8.7 – Приклад діалогового вікна

Це вікно створюється за допомогою функції int GR_draw_ask(char *vopros)

Дана функція повертає значення 0 або 1 в залежності від дій користувача.

```
char *vopros – текст, що відображається у вікні
int GR_draw_ask(char *vopros)
{
            //створення зафарбованого прямокутника
setfillstyle(1,7);
bar(150,100,300,230);
              //виведення тексту в вікні
setcolor(1);
outtextxy((450-8*strlen(vopros))/2,100+30,vopros);
              //рисуємо границі вікна
setcolor(15);
line(150,230,150,100);
line(150,100,300,100);
```

```
setcolor(0);
line(300,100,300,230);
line(300,230,150,230);
            //створення масиву структур кнопок даного вікна
struct menu_button ask[2]=
{
  {160,180,210,200," Tak",0},
  {230,180,280,200,"  Ni",0}
};
  char chouse;//змінна, призначена для отримання коду з клавіатури
і, в залежності від нього, переходу між кнопками
  int cursor_pos=0;//позиція курсора
  ask[cursor_pos].status=1;//задаєм активний стан кнопки «так»
//В циклі рисуємо кнопки з використанням нашої функції
for(int i=0;i<2;i++)
{
GR_draw_button(ask[i]);//створення кнопки за заданою структурою
}
  //виконуємо цикл, поки не введемо Enter або Esc
  do
       {
       chouse=getch();//зберігаємо код клавіші з клавіатури
switch(chouse)
{

      case 77://якщо код дорівнює клавіші права стрілка
        if ((cursor_pos)==0) //якщо номер курсора нуль
        {
             cursor_pos++;//збільшуємо номер курсора
             ask[cursor_pos].status=1;//задаємо активний статус
//відповідній кнопці
             GR_draw_button(ask[cursor_pos]);//перерисовуємо
//активну кнопку
             ask[cursor_pos-1].status=0;//задаємо пасивний ста-
тус //попередній кнопці
        GR_draw_button(ask[cursor_pos-1]);//перерисовуємо
//попередню кнопку
                  }
        break;
             case 75://якщо натиснута кнопка з лівою стрілкою
                  //аналогічно як при натисненні лівої стрілки
  if(cursor_pos == 1)
       {
         cursor_pos--;
         ask[cursor_pos].status=1;
         GR_draw_button(ask[cursor_pos]);
         ask[cursor_pos+1].status=0;
```

```
            GR_draw_button(ask[cursor_pos+1]);
        }
    break;
    }
```

//цикл виконується поки не була натиснута клавіша Enter або Esc

```
}while(chouse!=13 && chouse!=27);
```

//перевіряємо, якщо позиція курсора на кнопці «Так» або натиснута клавіша Enter, то функція повертає значення 1

```
if (cursor_pos==0 && chouse==13)
        return 1;
```

//інакше функція повертає значення 0

```
else    return 0;
    }
```

Для введення ключового слова до пошуку номера запису розроблено спеціальне діалогове вікно (як базове), приклад якого зображено на рисунку 8.8.

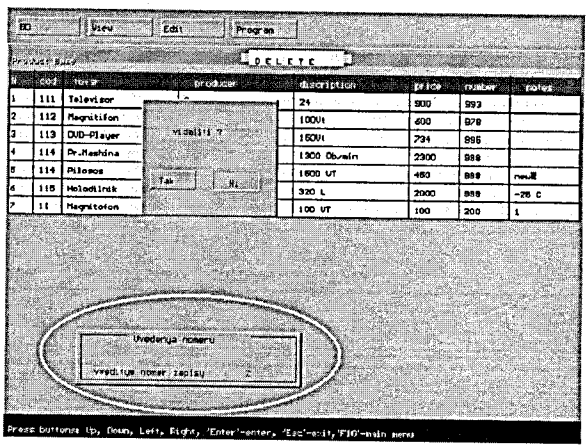


Рисунок 8.8 – Приклад діалогового вікна

Діалогове вікно програмно реалізується за допомогою функції:

```
void DrawWindow(int x0,int y0,int x1,int y1,char *head),
```

де

int x0, int y0 – координати лівого верхнього кутка вікна;
int x1, int y1 – координати правого нижнього кутка вікна;
char *head – текст, що відображається у верхній частині вікна;

Функція може містити такі оператори:

```
void DrawWindow(int x0,int y0,int x1,int y1,char *head)
{
```

```
//створення зафарбованого прямокутника сірого кольору
  setfillstyle(1,7);
  bar(x0,y0,x1,y1);
//рисуємо границі вікна
setcolor(15);
//білі границі
  line(x0,y1,x0,y0);
  line(x0,y0,x1,y0);
  setcolor(0);
//чорні границі
  line(x1,y0,x1,y1);
  line(x0,y1,x1,y1);
// рисування рамки
  setcolor(0);
  rectangle(x0+5,y0+5,x1-5,y1-5);
//зафарбовування верхньої частини рамки як основи для виведення тексту
  bar((x0+x1-8*strlen(head))/2-
8,y0+2,(x0+x1+8*strlen(head))/2+8,y0+20);
//виведення тексту (заголовок вікна)
  outtextxy((x0+x1-8*strlen(head))/2,y0+3,head);
  //рисування тіні вікна
  setcolor(0);
  line(x1+1,y0,x1+1,y1);
  line(x0,y1+1,x1,y1+1);
}
```

Розглянемо детальніше роботу та створення вікна, що використовується для номера запису або просто введення якогось числа. Даний діалог зображений на рисунку 8.8. Функція, яка відповідає за логіку роботи та створення даного графічного елемента, має такий заголовок:

```
int VvodNumber(int x0, int y0,char *vopros),
```

та може містити таки вхідні дані:

int x0, int y0 – координати верхнього лівого кутка вікна;

char *vopros – заголовок вікна.

Вихідним параметром такої функції може бути параметр типу int.

Функція може містити такі оператори:

```
int VvodNumber(int x0, int y0,char *vopros)
{
  char *str=vopros;//записуємо текст вікна
  int i=0;//індекс
  char cc;//символ введення
  *nzz;//тимчасовий рядок для редагування введеного числа
  //кінцеві координати визначаються з довжини тексту
  int xcur=x0+8*strlen(str);//початкова X-координата + довжина
в пікселях слова за умови, що один символ=8
  int ycur=y0;
```

```
//створення діалогового вікна
DrawWindow(x0-20, y0-40, xcur+8*5+20, y0+20, "Vvedenya nomeru");

//виведення тексту вікна
setcolor(1);
outtextxy(x0,y0,str);

//в циклі отримуємо код натиснутої клавіші і рисуємо символ підкреслення
while(cc!=13)
{
    while(!kbhit())//поки не були натиснуті клавіші з клавіатури
    {
        //рисуємо знак підкреслення
        setcolor(1);//задаємо чорний колір
        //знак підкреслення
        line(xcur+8*(i),ycur+8,xcur+8*(i+1),ycur+8);
        delay(50);//затримка на 50 мілісекунд
        setcolor(7);//задаємо сірий колір (колір фону)
        line(xcur+8*(i),ycur+8,xcur+8*(i+1),ycur+8);
        //знову рисуємо знак підкреслення
        delay(50);//затримка на 50 мілісекунд
    }
    cc=getch();//зберігаємо код клавіші з клавіатури
    setcolor(4);//задаємо колір тексту
    //перевірка на те чи не було введено //Backspace (код 8) чи 0 спочатку і чи не було введено більше ніж 5 символів
    //якщо так, то повертаємось на початок циклу
    if((cc==8 && i==0) || (cc!=8&&i>=5) || (cc=='0' && i==0))
    {
        continue;
    }
    //якщо введений Backspace (код 8), то в рядку видаляємо останній символ і зафарбовуємо його на екрані сірим кольором, тобто кольором фону
    if(cc==8&&i!=0)
    {
        nzz[i]=NULL;//поточний символ видаляється з рядка
        nzz[i-1]=NULL;//попередньо введений символ видаляється з рядка
        setfillstyle(1,7);
        bar(xcur+8*(i-1),ycur,xcur+8*(i),ycur+8);
    //зафарбовуємо попередній символ
        i--;//зменшуємо індекс рядка
```

```
        }
    //якщо введені цифри
        if(cc>=48&&cc<=57&&i<5)
        {
            nzz[i]=cc;//записуємо в рядок введений символ
            i++;//збільшуємо індекс
            nzz[i]='\0';//записуємо спец символ в кінець рядка для
коректної роботи з ним
            outtextxy(xcur,ycur,nzz);// виводимо на екран даний
рядок
        }
    }
    return atoi(nzz)-1;//конвертуємо даний рядок в число типу int та
повертаємо отриманий результат (вихідні дані)
    }
```

Для введення слова використовується аналогічна функція, яка відрізняється лише фільтром на введені символи, тобто не лише цифри враховуються, та в кінці повертається не число, а рядок.

`char *VvodSlovo(int x0, int y0,char *vopros) - така функція типу char *.`

Щоб очистити робочу область можна використати функцію `void clean_workspace()`, яка зафарбовує робочу область фоновим кольором, наприклад, таку функцію:

```
    void clean_workspace()
    {
        setcolor(main_bgcolor+1);//задаєм колір, де main_bgcolor -
константа
        setfillstyle(SOLID_FILL,main_bgcolor);//задаємо стиль
заливання
    //зафарбовуємо робочу область
        bar(0,topmargin,getmaxx(),getmaxy());
        setcolor(15);
    //виводимо текст як допомогу для користувача
        outtextxy(200,300,"Viberity Bazu danih v punkti
BD");
        outtextxy(5,465,"Press buttons: Up, Down, Left,
Right, 'Enter'-enter, 'Esc'-exit,'F10'-main menu ");
    }
```

Для більш глобального очищення екрана та встановлення базових параметрів можна використовувати функцію `void GR_set_base_params()`, структура якої наведена нижче.

```
    void GR_set_base_params()
    {
```

```
        setcolor(1);
        setfillstyle(1,7);
//зафарбовуємо весь екран
        floodfill(1,1,main_bgcolor);
}
```

Для роботи з такими базовими функціями, що створюють графічні елементи, можна розробити досить просту головну функцію `MyMenu()`, в якій прописана логіка роботи графічного меню. Розглянемо детально структуру такої функції.

```
int MyMenu()
  {
  char ch;//поточний  введений символ
//позиція курсора в головному меню (поточна активна кнопка)
  int cursor_position=0;
  int cursor_can_move_lr=1;//прапорець рухомості курсора
  int submenu_cursor_position=0;//позиція курсора в підменю
//додаткова змінна, де зберігається попередня позиція курсора
  int old_cursor_position=cursor_position;
 int old_submenu_cursor_position=submenu_cursor_position;
//присвоюємо поточну позицію
  int free_window=0;//прапорець на звільнення вікна
//вказівник на номер бази даних (динамічна змінна)
  int *nbd;
//виділення пам'яті для номера бази даних
  nbd = (int*)malloc(sizeof(int));
//присвоюємо 0-й номер
  *nbd=0;
//ініціалізація графіки
  GR_init();
//задаємо базові параметри
  GR_set_base_params();
//очищаємо робочу область
  clean_workspace();
setcolor(1);
  //задаємо масив структур для кнопок
  menu_button buttons[n_buttons]={
  {10,10,80,30,"BD",0},
  {90,10,90+70,30,"View",0},
  {160+10,10,160+10+70,30,"Edit",0},
  {230+20,10,230+20+70,30,"Program",0},
  };
  //задаємо масив структур для всіх підменю відповідно для кнопок
  submenu pull_down_menu[n_buttons]={
  {3,{"Tovar","Kliyent","Nakladna"},0,0,0,0},
  {2,{"Sort","Poshuk"},0,0,0,0},
  {3,{"Delete","Add","Rewrite"},0,0,0,0},
```

```
    {3,{"About","Avtor","Quit",},0,0,0,0}
    };
    //в циклі передаємо в функцію кожен елемент масиву структур
    // і рисуємо таким чином кнопки

    for(int i=0;i<n_buttons;i++)
         GR_draw_button(buttons[i]);
    //аналізуємо введені дані з клавіатури
    do
    {
         ch=getch();
    //записуємо код клавіші
    //перевіряємо на відповідні коди
         switch(ch)
         {
              case 68:// F10 Pressed
              //робимо поточну кнопку неактивною
              buttons[cursor_position].status=0;
              //перерисовуємо кнопку
              GR_draw_button(buttons[cursor_position]);
              cursor_can_move_lr=1;
              //встановлюємо прапорець можливості руху курсора на 1
– активний
              cursor_position=0;
              //збиваємо позицію курсора на першу кнопку (0)
              buttons[cursor_position].status=1;
              //задаємо статус першій кнопці
              GR_draw_button(buttons[cursor_position]);
              // перерисовуємо кнопку
         break;
              case 77://Right side movement
              // якщо введена права стрілка
              //якщо позиція курсора не більше максимальної
              //якщо курсор маже рухатись

   if ((cursor_position+1)<(n_buttons)&&cursor_can_move_lr)
   {
                   //збільшуємо величину курсора на 1 (номер кнопки)
                   cursor_position++;
                  //перерисовуємо нову кнопку на активну
     buttons[cursor_position].status=1;
     GR_draw_button(buttons[cursor_position]);
                   //перерисовуємо попередню кнопку на неактивну
        buttons[cursor_position-1].status=0;
          Gdraw_button(buttons[cursor_position-1]);
     }
         break;
              case 75://Left side movement
```

```
                    // якщо введена ліва стрілка, виконуємо все аналогічно як
                при введенні правої стрілки, тільки рухаємось в іншому напря-
                мку
        if(cursor_position !=0 && cursor_can_move_lr)
                    {
                         if(cursor_position-1!=n_buttons)
                              cursor_position--;
                         buttons[cursor_position].status=1;
                        GR_draw_button(buttons[cursor_position]);
                        buttons[cursor_position+1].status=0;
                    GR_draw_button(buttons[cursor_position+1]);
                    }
                    break;

                //при введенні Enter створюється підменю
                    case 13://Enter key
                    buttons[cursor_position].status=1;
                    GR_draw_button(buttons[cursor_position]);
                 //вимикаємо рухливість курсора
                //рисуємо відповідне меню
                    pull_down_menu[cursor_position].state=1;
                //активуємо поточне підменю
                //задаємо координати підменю такі, як координати кнопки
      pull_down_menu[cursor_position].x0=buttons[cursor_
position].x0;
      pull_down_menu[cursor_position].y0=buttons[cursor_
position].y0;

                //рисуємо вибране підменю
                GR_draw_submenu(pull_down_menu[cursor_position]);
                    char chr;
                //встановлюємо номер вибраного пункту (курсор підменю) на
                //перший
                    submenu_cursor_position=0;
                //аналізуємо введені символи з клавіатури
                    do
                    {
                //записуємо код клавіші
                        chr=getch();
                //перевіряємо введені символи

    switch(chr)
                    {
                    //якщо введена клавіша Escape, то ми знищуємо дане під-
дменю
                    //тобто виходимо з підменю в головне меню
                        case 27://If Escape pressed
                        pull_down_menu[cursor_position].state=0;
```

```
    GR_draw_submenu(pull_down_menu[cursor_position]);
                        cursor_can_move_lr=1;
                    //дозволяємо рухатись курсору в сторони
                        //очищуємо робочу область
                        clean_workspace();
  //якщо вікна вільні, тоді перерисовуємо попередню інформацію
                        if(free_window)
  //дана функція призначена для виконання певної операції при виборі
певного пункту підменю
    execute_clik(nbd,old_cursor_position,old_submenu_cursor_po
sition);
                        break;
                        //якщо натиснута стрілка вниз
                        case 80://If down button pressed
  //перевіряємо, чи позиція курсора не більша за допустиму

    if(pull_down_menu[cursor_position].items_number>submenu_cu
rsor_position)
          submenu_cursor_position++;
  //якщо більша, то повертаємось на початок
  if(pull_down_menu[cursor_position].items_number<=submenu_cu
rsor_position) submenu_cursor_position=0;
  //перерисовуємо підменю з новим вибраним пунктом

    pull_down_menu[cursor_position].active_element=submenu_cur
sor_position;

    GR_draw_submenu(pull_down_menu[cursor_position]);
                        break;
  //якщо натиснута стрілка вверх, виконуємо аналогічні дії
  //як і при натисканні клавіші зі стрілкою вниз, але рухаємось в іншому
напрямку
                        case 72://If up button pressed
                        if (0>=submenu_cursor_position)

    submenu_cursor_position=pull_down_menu[cursor_position].it
ems_number;

                        if(submenu_cursor_position>0)
                             submenu_cursor_position--;
                        //submenu

    pull_down_menu[cursor_position].active_element=submenu_cur
sor_position;

    GR_draw_submenu(pull_down_menu[cursor_position]);
                        break;
                        //якщо введена клавіша Enter
                        case 13:
```

```
                    buttons[cursor_position].status=0;
                    //задаємо пасивний статус поточній кнопці
```
//задаємо пасивний статус підменю, стираємо підменю (перерисовуємо в пасивному стані) та перерисовуємо кнопку
```
  pull_down_menu[cursor_position].state=0;
                    GR_draw_button(buttons[cursor_position]);

    GR_draw_submenu(pull_down_menu[cursor_position]);
```
//очищуємо робочу область
```
                    clean_workspace();
```
//виконуємо необхідну операцію в залежності від вибраного пункту в підменю
```
    execute_clik(nbd,cursor_position,submenu_cursor_position);
                    old_cursor_position=cursor_position,

    old_submenu_cursor_position=submenu_cursor_position;
```
//встановлюємо прапорець 1 – так
```
                    free_window=1;
```
//присвоюємо символу значення 27 для виходу з циклу
```
                    chr=27;
                    ch=75;
                    break;
                    }
                } while(chr!=27);
```
//цикл виконується поки не була введена клавіша
//Esc або не вибраний пункт в підменю
```
                break;
            }
    } while(ch!=27);
```
//виходимо з програми, натиснувши клавішу Esc
```
  free(nbd);
```
//очищуємо екран
```
  cleardevice();
```
//закриваємо графіку
```
  closegraph();
  return 0;
}
```

8.5 Контрольні запитання

1. Як створюються кнопки в графічному меню?
2. З якою метою створюються діалогові вікна? Як це реалізувати?
3. Як створити просто діалогове вікно з вибором кнопки «ТАК» чи «НІ» для руху по програмі?
4. Як зафарбувати робочу область фоновим кольором?
5. Як задається стиль тексту, яким ми відображуємо пункти меню?

6. Як відбувається аналіз даних, введених з клавіатури?

7. Як забезпечити неперервне виконання циклу до тих пір, поки не буде натиснута клавіша?

8. Як забезпечується рух по пунктах меню?

9. Яке призначення функції роботи з головним меню?

8.6 Практикум з програмування

1. Написати фрагмент програми для створення вікна за заданими координатами.

2. Написати фрагмент програми для створення вікна з подвійною рамкою.

3. Написати фрагмент програми, в якому б задавалася кількість пунктів меню.

4. Розробити фрагмент програми, яка б створювала рядок горизонтального меню.

5. Написати фрагмент програми, який би дозволяв здійснювати рух по пунктах головного меню.

6. Написати фрагмент програми для створення вікна висхідного меню.

7. Написати фрагмент програми для здійснення руху по пунктах підменю.

автоматизо́ваний (рос. автоматизированный, англ. *automatic*) — той, що виконує виробничі процеси за допомогою автоматичних приладів, машин; який здійснюється автоматично.

адре́са (рос. адрес, англ. *address*) — символ чи група символів, які називають регістр, окремі частини пам'яті, інші джерела даних або місце призначення інформації.

алгори́тм, -у (рос. алгоритм, англ. *algorithm*) — задана заздалегідь послідовність чітко визначених команд для розв'язання задачі за скінченне число кроків. В інформаційних системах використовують а. пошуку даних, а. сортування даних та інші.

архі́в, -у (рос. архив, англ. *archives*) — 1. Сукупність даних чи програм, які зберігаються на зовнішньому носієві, тимчасово не використовуються, але у разі потреби можуть бути віднайдені. 2. Стиснута інформація.

атрибу́т, -а (рос. атрибут, англ. attribute) — інформаційне відображення властивості будь-якого об'єкта, процесу чи явища.

база (рос. база, англ. *base*) — основні дані або елементи.

б. да́них (рос. база данных, англ. *database*) — сукупність даних, організованих за певними правилами, які передбачені загальними принципами описання, зберігання і маніпулювання даними.

ба́йт, -а (рос. байт, англ. *byte*) — 1. Одиниця обсягу інформації, що дорівнює восьми бітам. 2. Послідовність восьми бітів.

бі́т, -а (рос. бит, англ. *bit*) — 1. Фундаментальна одиниця інформації, яку використовують у теорії інформації. Означає найменшу кількість інформації, необхідної для розрізнення двох рівноймовірних подій. 2. Найменша «порція» пам'яті, реалізована в ЕОМ.

бло́к, -а (рос. блок, англ. *block*) — 1. Сукупність взаємопов'язаних елементів і вузлів пристрою, які виконують певну функцію. 2. Фізичний запис групи байтів на носієві даних. 3. Сукупність елементів, які записують як одне ціле. 4. Набір інформаційних одиниць, таких, як слова, знаки чи записи, які записані в сусідніх позиціях пам'яті або периферійного пристрою. 5. У теорії кодування — упорядкований набір символів, що має, як правило, фіксовану довжину. 6. У паралельному програмуванні блокування — заборона подальшого виконання однієї послідовності команд доти, доки інша послідовність не запропонує необхідну команду для її розблокування.

бу́фер, -а (рос. буфер, англ. *buffer*) — 1. Пам'ять для проміжного зберігання даних. 2. Схема або прилад, розміщені між двома іншими приладами для згладжування змін швидкості, для забезпечення асинхронної роботи. 3. Робоча частина пам'яті під час пересилання даних.

введе́ння і уведення (рос. ввод, англ. *input*) — процес отримання даних будь-якою частиною комп'ютера.

ве́рсія (рос. версия, англ. *version*) — 1. Варіант програмного продукту. 2. Файл, що є модифікацією іншого файлу.

ви́клик, -у (рос. вызов, англ. *calling*) — передавання сигналів селекції, призначених для встановлення зв'язку між станціями передавання даних.

в. за і́менем (рос. вызов по имени, англ. *call by name*) — виклик підпрограми з чітким зазначенням її імені.

ви́лучення (рос. извлечение, англ. *erasing*) — вилучення даних з пам'яті без залишення запису про них. У комп'ютері стирання (вилучення) рівноцінне очищенню. На дисплеї — це видалення зображення з екрана.

відеока́рта (рос. видеокарта, англ. *video card, display card*) — допоміжна плата, за допомогою якої комп'ютер забезпечує зображення на екрані дисплея тексту, графіки, відеоматеріалів; підключається до основної (материнської) плати через спеціальне гніздо.

відеоконтро́лер, -а (рос. видеоконтроллер, англ. video controller) — мікропроцесор у відеокарті, який виконує програму управління відеокартою.

вікно́ (рос. окно, англ. window) — 1. Прямокутна частина простору відображення інформації на дисплеї. 2. Засіб фрагментації повідомлень і блоків даних приладом, що визначається протоколом передавання даних.

вказівни́к (рос. указатель, англ. pointer або англ. reference) — тип даних в комп'ютерних мовах програмування. Значення вказівника посилається на інше значення, що записане будь-де в пам'яті комп'ютера (фактично містить його адресу).

гра́фіка, -и (рос. графика, англ. graphics) — 1. Вид образотворчого мистецтва, основним зображальним засобом якого є рисунок. 2. Лінгв. Письмові чи друковані знаки, що відбивають звуки мови, інтонацію; зображення живої мови письмовими знаками. 3. Розділ учення про різні системи письмових або друкованих знаків, літер.

г. комп'ю́терна (рос. графика компьютерная, англ. computer graphics) — методи та способи взаємоперетворень графічних зображень і даних за допомогою комп'ютера.

да́ні, -них (рос. данные, англ. data) — набір тверджень, фактів, чисел, лексично і синтаксично взаємопов'язаних між собою.

д. вихідні́ (рос. выходные данные, англ. output data) — дані, що виводяться або призначені для виведення будь-якою частиною комп'ютера.

д. вхідні́ (рос. входные данные, англ. input data) — дані, отримувані чи призначені для отримання будь-якою складовою комп'ютера.

диспле́й, -я (рос. дисплей, англ. display) — пристрій відтворення текстової чи графічної інформації без її довготривалої фіксації.

докуме́нт, -а (рос. документ, англ. document) — матеріальний носій інформації, який містить оформлені в установленому порядку повідомлення і має юридичну чинність.

д. електро́нний (рос. д. электронный, англ. electronic d.) — сукупність даних у пам'яті комп'ютера, які сприймаються користувачем за допомогою певних програмних і апаратних засобів.

до́ступ, -у (рос. доступ, англ. access) — надання або прийняття процесом опрацювання даних порції даних здійсненням послідовних операцій пошуку, читання і записування.

дра́йвер, -а (рос. драйвер, англ. driver) — файл, що містить інформацію, необхідну для програми керування роботою периферійного пристрою (драйвер дисплея, драйвер дисковода, драйвер сканера тощо).

електро́нний (рос. электронный, англ. electronic) — пов'язаний із застосуванням властивостей електронів, заснований на їх використанні.

елеме́нт, -у (рос. элемент, англ. element) — 1. Динамічний об'єкт модельованого процесу. 2. Неподільна одиниця множини.

забезпе́чення (рос. обеспечение, англ. maintenance, support) — 1. Матеріальні засоби до існування. 2. Сукупність математичних, програмних, мікропрограмних, апаратурних засобів та організаційних заходів, спрямованих на автоматичне оброблення даних за допомогою цифрових обчислювальних машин і пристроїв.

з. програ́мне (рос. программное обеспечение, англ. software) — сукупність програм системи опрацювання інформації й програмних документів, необхідних для експлуатації цих програм.

з. систе́мне (рос. системное программное обеспечение, англ. system software) — програмне забезпечення, призначене для експлуатації і технічного обслуговування комп'ютера, організації обчислювальних робіт, автоматизації розроблення використовуваних програм.

з. станда́ртне (рос. стандартное программное обеспечение, англ. bundled software) — програмне забезпечення, яке постачається разом з комп'ютером без додаткової оплати.

завантаження (рос. загрузка, англ. load) — пересилання даних з носія даних до основної пам'яті або з основної у регістрову з метою безпосереднього використання їх в операціях процесора.

з. програ́ми (рос. загрузка программы, англ. program loading) — перенесення програми із зовнішньої пам'яті в основну.

зале́жність, -ності (рос. зависимость, англ. relationship) — зв'язок між двома чи більше атрибутами, що цілком або частково визначає значення одного при заданих значеннях інших.

за́пис, -у (рос. запись, англ. record) — упорядкований набір даних, що називаються полями, де кожне поле має власне ім'я і тип.

за́пит, -у (рос. запрос, англ. enquiry) — завдання на пошук певних даних у базі даних.

здатність, -ності (рос. способность, англ. ability) — 1. Властивість до значення здатний. Уміння здійснювати, виконувати, робити що-небудь. 2. Природне обдарування, здібність.

з. пропускна́ (рос. пропускная способность, англ. capacity) — здатність системи виконувати (пропускати через себе) за визначений час певну кількість завдань (інформації). Найчастіше мається на увазі пропускна здатність каналу зв'язку.

з. роздільна (рос. разрешающая способность, средства отображения информации, англ. resolution) — максимальна кількість тих елементів інформації на екрані засобу відображення інформації, які розрізняються за лінійним розміром.

зна́к, -а (керува́ння) (рос. знак управления, англ. control character) — символ, що стимулює виконання певної комп'ютерної операції.

зчи́тування (рос. считывание, англ. read-out) — 1. Процес сприймання даних у комп'ютері або його частині та передання їх до зовнішнього носія пам'яті чи до друкувального пристрою. 2. Відображення на дисплеї (як правило, висвітлюється на панелі керування) відомостей про стан комп'ютера.

ім'я́, -ені (рос. имя, англ. name, identifier) — умовне найменування відповідного об'єкта.

інсталя́ція (рос. инсталляция, англ. installation) — встановлення програмного забезпечення на комп'ютер.

інтерпрета́тор, -а (рос. интерпретатор, англ. interpreter) — 1. Мовний процесор, який аналізує кожний рядок вхідної програми й одночасно виконує зазначені в ньому дії. На відміну від компілятора не формує машинною мовою повністю скомпільовану програму, яка виконується потім. 2. Програма чи технічний засіб, що виконує інтерпретацію.

інтерфе́йс, -у (рос. интерфейс, англ. interface) — 1. Сукупність засобів і правил, що забезпечують взаємодію пристроїв комп'ютера. 2. Частина програмного забезпечення для реалізації діалогу користувача з прикладною програмою.

і. вну́трішній (рос. внутренний интерфейс, англ. back-end interface) — інтерфейс із внутрішнім компонентом системи (напр., комунікаційного процесора з головним комп'ютером, комп'ютера зі спецпроцесором).

і. зо́внішній (рос. внешний интерфейс, англ. front-end interface) — засоби і правила взаємодії підсистеми із зовнішніми об'єктами на відміну від її взаємодії з іншими об'єктами системи.

і. користувача́ (рос. интерфейс пользователя, пользовательский интерфейс, англ. user interface) — 1. Комплекс програмних засобів, які забезпечують взаємодію користувача з системою. 2. Засоби зв'язку між користувачем і системою.

інформа́ція (рос. информация, англ. information) — 1. Нові відомості, які дають можливість поліпшити процеси, пов'язані з перетворенням речовини, енергії та самої інформації. 2. Нові відомості, прийняті, зрозумілі й оцінені користувачем як корисні. 3. Нові відомості, які розширюють запас знань користувача.

катало́г, -у (рос. каталог, англ. directory) — 1. Структура даних, яка забезпечує пошук об'єкта за текстовим іменем. 2. Набір файлів чи набір каталогів або набір файлів і каталогів, названий текстовим іменем (в операційних системах).

ке́ш-па́м'ять (рос. кэш-память, англ. cache memory) — надшвидкодійна (надоперативна) пам'ять, яка є проміжною між основною пам'яттю та процесором і використовується для зберігання даних, що часто викликаються програмою.

кілоба́йт, -а (рос. килобайт, англ. kilobyte) — одиниця обсягу інформації, що дорівнює 1024 байтам (приблизно 1000 байтів).

клавіату́ра (рос. клавиатура, англ. keyboard) — пристрій, що складається з сукупності розташованих у певному порядку клавіш (кнопок) з літерами, які використовують для введення і редагування даних, а також для керування виконанням операцій.

кла́віша (рос. клавиша, англ. key) — кнопка клавіатури, натискання якої ініціює певну дію.

кла́стер, -а (рос. кластер, англ. cluster) — 1. Група накопичувачів на магнітній стрічці, відеоприладів чи терміналів із загальним контролером. 2. Група секторів на магнітному (оптичному) диску, яка читається або записується як одне ціле. Це мінімальна одиниця інформації, яку операційна система записує на диск або зчитує з диска.

клієнт, -а (рос. клиент, англ. client) — у комп'ютерній мережі — робоча станція з можливостями автономного оброблення даних (наприклад, персональний комп'ютер), яка для отримання інформації звертається до сервера.

клієнт-се́рвер, -а (рос. клиент-сервер, англ. client server) — технологія оброблення інформації в мережі, яка базується на взаємодії клієнтів з інформаційним сервером.

кно́пка (рос. кнопка, англ. button) — 1. Графічний образ на екрані дисплея, клацання мишкою на якому має ефект натискання звичайної кнопки приладу (клавіші клавіатури). Використовується у системах графічного інтерфейсу для подання команд, перемикання режимів роботи та ін. за допомогою мишки. 2. Елемент мишки, призначений для подання команд.

ко́д, -у (рос. код, англ. code) — система символів для подання різноманітних типів даних або команд у комп'ютерній системі.

кома́нда (рос. команда, англ. instruction) — мовна конструкція, яка точно визначає операцію і, за необхідності, ідентифікує операнди.

комі́рка (рос. ячейка, англ. cell) — 1. Мінімальна ділянка оперативної пам'яті комп'ютера, яка адресується і заповнюється (читається) однією командою процесора. 2. Прямокутна ділянка в електронних таблицях, обмежена горизонтальними та вертикальними лініями, куди заноситься інформація, що має конкретну адресу.

комп'ю́тер, -а (рос. компьютер, англ. computer) — сукупність технічних засобів, які дають можливість обробляти інформацію за наперед заданими алгоритмами, записаними в програмах, та отримувати результат у необхідній формі.

к. персона́льний (рос. персональная ЭВМ, англ. personal computer) — настільний мікрокомп'ютер, що має експлуатаційні характеристики побутового приладу й універсальні функціональні можливості.

компа́кт-диск, -а (рос. компакт-диск, англ. compact disk) — диск для тривалого зберігання інформації, де запис проводять за допомогою лазерного променя, а для читання запису використовують оптичні прилади. Як правило, допускає одноразовий запис і багатократне читання.

компіля́ція (рос. компиляция, англ. compiling) — переведення тексту програми, написаної мовою високого рівня, в еквівалентну за діями програму, записану машинними кодами, тобто кодами, які сприймаються процесором як команди.

конфігура́ція (рос. конфигурация, англ. configuration) — група машин, об'єднаних і запрограмованих для роботи як єдина інтегрована система.

користува́ч, -а (рос. пользователь, потребитель, англ. user, subscriber) — особа, яка користується послугами обчислювальної техніки для отримання інформації чи розв'язання різних завдань.

корте́ж, -у (рос. кортеж, англ. n-tuple) — упорядкована послідовність атрибутів, що описують матеріальний об'єкт. При утворенні таблиці бази даних, як правило, перетворюється в запис.

курсо́р, -у (рос. курсор, англ. cursor) — символ, що вказує на екрані дисплея на активну позицію, тобто позицію, на якій відображатиметься наступний введений з клавіатури знак.

маніпуля́тор, -а (рос. манипулятор, англ. grabber hand) — 1. Пристрій для виконання певних операцій під керуванням комп'ютера. 2. Пристрій для спрощеного керування комп'ютером (маніпулятор «миша», ігрові маніпулятори типу «джойстик»).

маси́в, -у (рос. массив, англ. array) — сукупність однотипних (логічно однорідних) елементів, упорядкованих за індексами, що визначають положення елемента в масиві.

мегаба́йт, -а (рос. мегабайт, англ. megabyte) — одиниця об'єму інформації, що дорівнює 1024 кілобайтам, або 1024 × 1024 = 1 048 576 байтам (приблизно 1 млн байтів).

меню́, невідм. (рос. меню, англ. menu) — зображуваний на екрані дисплея список команд чи варіантів відповіді, з якого користувач вибирає необхідний варіант, вводячи номер чи букву, або вказуючи на пункт меню курсором.

ми́шка (рос. мышка, англ. mouse) — портативний пристрій (маніпулятор) для введення координат, який функціонує внаслідок пересування його по площині.

мо́ва (рос. язык, англ. language) — набір символів, домовленостей і правил, які використовують з метою подання інформації.

м. алгоритмі́чна (алгори́тмова) (рос. алгоритмический язык, англ. algorithmic language) — штучна мова, призначена для алгоритмів.

м. з бло́ковою структу́рою (рос. язык с блочной структурой, англ. blockstructured language) — мова високого рівня, у якій опис однієї дії може містити опис об'єкта того самого класу (напр., вкладені процедури і блоки).

м. інформаці́йно-пошуко́ва (рос. информационно-поисковый язык, англ. information retrieval language) — у документальних інформаційно-пошукових системах — мова, призначена для формалізованого описання змісту документів, які повинні зберігатися в системі, а також для складання запитів до системи на пошук і видавання документів.

м. керува́ння да́ними (рос. язык управления данными, англ. data manipulation language, DML, data base control language, DBCL) — непроцедурна мова, яка дозволяє задавати функції введення та коректування даних, реорганізації та переміщення файлів даних тощо.

м. маши́нна (рос. язык машинный, англ. machine language) — мова, яка читається машиною безпосередньо, без будь-якого перекладу, на відміну від штучної мови чи мови символів, що потребують певного тлумачення своїх термінів машинною мовою в комп'ютері, перш ніж він зможе виконувати подані команди.

м. моделюва́ння (рос. язык моделирования, англ. simulation language) — мова програмування, призначена для розроблення програм моделювання.

м. програмува́ння (рос. язык программирования, англ. programming language) — штучна мова написання програм для комп'ютера.

м. шту́чна (рос. язык искусственный, англ. artificial language) — в інформатиці та обчислювальній техніці — мова, яка базується на комплексі точно визначених правил, встановлених для складання тексту.

моде́ль (рос. модель, англ. model) — матеріальний об'єкт, система математичних залежностей чи програма, яка імітує структуру або функціонування досліджуваного об'єкта.

м. да́них (рос. модель данных, англ. data model) — передбачає формування множини допустимих інформаційних конструкцій, множини операцій над даними і множини обмежень для даних, що зберігаються, описуючи потреби користувача в даних.

мо́дуль, -я (рос. модуль, англ. module) — одиниця програмного забезпечення.

м. завантáжувальний (рос. загрузочный модуль, англ. load module) — програмний модуль, поданий у формі, придатній для завантаження в основну пам'ять з метою виконання.

моні́тор, -а (рос. монитор, англ. monitor) — пристрій, призначений для спостереження і запису певних дій у системі опрацювання даних з метою їх подальшого аналізу. Син.: дисплей.

нала́годжування (рос. налаживание, англ. debugging) — процес виявлення та усунення помилок у комп'ютерних програмах або обладнанні.

носі́й, -я да́них (рос. носитель данных, англ. data medium) — матеріальний об'єкт, призначений для записування і зберігання даних.

обла́днання (рос. оборудование, англ. equipment) — сукупність механізмів, приладів, пристроїв, необхідних для чого-небудь; спорядження.

он-ла́йн, -у (рос. он-лайн, англ. on-line) — зв'язок, який підтримується у режимі реального часу (безперервно).

о. режи́м (рос. онлайновый режим, англ. on-line mode) — див.: режим.

опера́ція (рос. операция, англ. operation) — певна дія, яку виконує комп'ютер відповідно до команд, закладених у його програмі.

о́пція (рос. опция, англ. option) — уточнення команди за допомогою спеціальних символів чи меню.

опрацюва́ння (рос. обработка, англ. processing) — дія за значенням опрацювати. Глибоке вивчення чого-небудь, докладне ознайомлення з чимось, ретельна підготовка.

о. елекро́нне (рос. обработка электронная, англ. electronic data processing) — опрацювання даних за допомогою електронних систем.

о. паке́тне (рос. пакетная обработка, англ. batch processing) — опрацювання комп'ютерних даних групами для подальшого зберігання у пам'яті комп'ютера у вигляді окремих випусків (пакетів).

о. паралéльне (рос. параллельная обработка, англ. concurrent processing) — здатність комп'ютера виконувати кілька програм або завдань одночасно.

офла́йновий режи́м (рос. офлайновый режим, англ. off-line mode) — див.: режим.

паке́т, -а (рос. пакет, англ. package) — об'єднана група компонентів у комп'ютерній системі, наприклад, блок взаємопов'язаних програм чи додатковий комплект обладнання та програм.

п. ди́сків (рос. пакет дисков, англ. disk pack) — у комп'ютерній системі — комплект магнітних дисків, що її використовують для зберігання інформації. Пакети дисків дають широкі можливості для зберігання значних обсягів даних та доступу до збереженої інформації.

па́м'ять (рос. память, англ. memory) — функціональна частина комп'ютера, призначена для приймання, зберігання даних.

п. операти́вна (рос. память оперативная (ОЗУ), англ. RAM (random-access memory) — пристрій для зберігання інформації в комп'ютері, який функціонує лише протягом часу, коли увімкнено живлення; використовується для зберігання програм і даних під час роботи комп'ютера.

п. пості́йна (рос. память постоянная (ПЗУ), англ. ROM (read-only memory) — пристрій для тривалого зберігання інформації в комп'ютері незалежно від живлення; використовується для зберігання важливих системних параметрів і програм, необхідних для запускання операційної системи після вмикання та роботи комп'ютера.

пане́ль (рос. панель, англ. bar) — у графічному інтерфейсі — набір кнопок, пов'язаних з групою операцій.

па́пка (рос. папка, англ. folder) — те саме, що й каталог; використовується для позначення каталогу файлів у системах, орієнтованих на непідготовленого користувача (напр., у системах підготування текстів).

паро́ль, -я (рос. пароль, англ. password) — секретне слово, яке пред'являє користувач системі з метою отримання доступу до даних і програм.

пло́тер, -а (рос. плоттер, англ. plotter) — пристрій для виведення на папір або інший матеріал широкоформатної графіки; використовується переважно для виведення технічних креслень, схем.

по́ле (рос. поле, участок, англ. field) — група пов'язаних між собою символів, що опрацьовуються як єдине ціле в комп'ютерних операціях, особливо при налагодженні пам'яті та файлових схем для різних цілей.

по́рт, -у (рос. порт, англ. port) — 1. Місце взаємодії двох процесів. 2. Апаратні та програмні засоби для підключення периферійного пристрою до комп'ютера.

посила́ння (рос. ссылка, англ. reference) — запис в електронному документі, що задає адресу переходу на частину документа чи на інший документ.

по́шук, -у (рос. поиск, англ. search) — у комп'ютерній пам'яті — добір (ототожнення, зіставлення) кодованих даних за заданими ознаками (критеріями), також закодованими, для знаходження необхідної інформації.

прапоре́ць, -рця (рос. флажок, англ. flag) — 1. Елемент графічного інтерфейсу, який позначає об'єкт, що може перебувати у двох станах (вимкнено — прапорець опущений, увімкнено — прапорець піднято). 2. Бітова комірка процесора, що використовується для фіксації його режимів.

приміpник, -а (рос. экземпляр, англ. copy) — друкована одиниця, комп'ютерна операція, що полягає у перенесенні даних з пам'яті комп'ютера в будь-яке інше місце; при цьому оригінальний запис залишається в пам'яті і не знищується.

при́нтер, -а (рос. принтер, печатающее устройство, англ. printer) — пристрій, призначений для виведення текстової чи графічної інформації на папір або інший матеріал.

при́стрій, -рою (рос. устройство, англ. device, arrangement) — пристосування, обладнання, за допомогою якого виконується робота або спрощується певний виробничий процес.

п. керува́ння (рос. устройство управления, англ. control unit) — блок комп'ютера, який керує усіма операціями в системі згідно з інструкціями, що надходять з програми.

програ́ма (рос. программа, англ. program) — упорядкована послідовність команд, яка призначена для виконання комп'ютером.

програмі́ст, -а (рос. программист, англ. programmer) — спеціаліст, що виконує розроблення і налагодження програм.

програмува́ння (рос. программирование, англ. programming) — теоретична і практична діяльність щодо забезпечення програмного керування

опрацюванням даних, що охоплює створення програм, а також вибір структури і кодування даних.

протоко́л, -у (рос. протокол, англ. protocol) — сукупність правил взаємодії комп'ютерів у мережі.

проце́сор, -а (рос. процессор, англ. processor) — функціональна частина комп'ютера або системи опрацювання інформації, призначена для з'ясування змісту програм.

реда́ктор, -а (рос. редактор, англ. editor) — 1. Той, хто редагує текст. 2. Керівник видання.

р. графі́чний (рос. графический редактор, редактор изображений, англ. graphics editor) — програма редагування графічної інформації.

р. текстови́й (рос. текстовый редактор, англ. word processor) — технологічно удосконалений механізм, який звичайно складається із катодної трубки і клавіатури, що дає можливість працювати зі словами і великими блоками тексту, які зберігаються у пам'яті комп'ютера. Кінцевий результат може бути автоматично перевірений і відредагований на екрані, видрукуваний на принтері, або матеріал може бути переданий будь-куди і перероблений для подальшого опрацювання.

розря́дність, -ості (при́строю) (рос. разрядность, англ. capacity) — обсяг інформації в бітах, яку обробляє пристрій за одну команду.

рядо́к, -дка́ (рос. строка, англ. string) — 1. Набір символів, розміщених в одному рядку текстового документа. 2. Набір комірок електронної таблиці, розміщених на одній горизонталі.

се́рвер, -а (рос. сервер, англ. server) — 1. Комп'ютер, що надає послуги іншим користувачам мережі. 2. Програма, призначена для обслуговування користувачів у мережі.

се́рвіс, -у (рос. сервис, англ. service) — 1. Засоби для обслуговування програмного забезпечення. 2. Засоби для допомоги користувачеві програмного забезпечення.

середо́вище (рос. среда, англ. environment) — оточення, в якому функціонує об'єкт.

с. програ́мне (рос. программная среда, англ. software environment) — програмні засоби, з якими взаємодіє програма чи система.

с. програмува́ння (рос. среда программирования, англ. programming environment) — сукупність технічного і програмного забезпечення, яка дає можливість розробляти програми для комп'ютера.

систе́ма (рос. система, англ. system) — будь-який об'єкт, який, з одного боку, може розглядатись як одне ціле, а з другого — як множина пов'язаних між собою або взаємодійних частин.

с. інформаці́йна (рос. система информационная, англ. information system) — обчислювальна система, яка забезпечує доступ користувачів і програм до загальної інформації.

с. інформаці́йно-пошуко́ва (рос. информационно-поисковая система, англ. information system) — автоматизована система, яка забезпечує інфор-

мацією користувача і реалізує пошук інформації за попередньо заданими ознаками.

с. керува́ння ба́зами да́них (СКБД) (рос. система управления базами данных (СУБД), англ. database management system) — програма (набір програм), призначена для маніпулювання базами даних (охоплює введення, редагування, оброблення даних).

с. обчи́слювальна (рос. система вычислительная, англ. computer system) — система, що має обчислювальне обладнання та необхідний обслуговувальний персонал, виконує функції введення, опрацювання, зберігання, виведення та керування з метою здійснення послідовності операцій над даними.

с. операційна (рос. система операционная, англ. operating system) — набір програм, призначених для організації взаємодії між складовими комп'ютера, розподілу ресурсів комп'ютера між прикладними програмами, управління виконанням прикладних програм, видавання інформації для користувача, приймання та виконання команд користувача.

с. програмува́ння (рос. система программирования, англ. program system) — система, призначена для надання програмам форми, придатної для виконання процесором комп'ютера. Охоплює мову програмування, компілятори чи інтерпретатори програм, написані цією мовою, відповідну документацію, допоміжні засоби (редактори та ін.).

с. автоматизованого проектування (рос. автоматизированного проектирования, англ. computer-aided design) — комплекс засобів, що допомагає користувачу створювати нові об'єкти.

с. файлова (рос. система файловая, англ. file system) — частина операційної системи, що забезпечує виконання операцій над файлами.

сло́во (рос. слово, англ. word) — мовна одиниця, що є звуковим вираженням поняття про предмет або явище об'єктивного світу.

с. ключове́ (рос. ключевое слово, англ. keyword) — інформативне слово, пов'язане з документом у комп'ютерній інформаційно-пошуковій системі, яке використовують для пошуку за предметом з метою отримання відповідного матеріалу, що зберігається в системі. Дві найбільш поширені пошукові системи: KWIC (ключове слово з контексту, тобто з назви чи тексту) і KWOC (ключове слово, вибране поза контекстом).

сортува́ння (рос. сортировка, англ. sorting) — упорядкування даних за певним правилом (наприклад, в алфавітному порядку).

с. за зме́ншенням (рос. сортировка по убыванию, англ. descending sort) — сортування, коли записи впорядковуються за спаданням значень ключових полів.

с. за зроста́нням (рос. сортировка по возрастанию, англ. ascending sort) — сортування, коли записи впорядковуються за зростанням значень ключових полів.

с. за ключе́м (рос. сортировка по ключу, англ. key sorting) — сортування за таблицею адрес, коли ключ сортування поданий разом з адресами.

спи́сок, -ску (рос. список, англ. list) — послідовність однотипних записів, у якій на місце розташування наступного елемента списку вказує поточний запис.

сто́впець, -я (рос. столбец, англ. column) — набір клітинок електронної таблиці, розміщених на одній вертикалі.

структу́ра (рос. структура, англ. structure) — множина елементів системи та взаємозв'язків між ними.

сумі́сність, -ості (рос. совместимость, англ. compatibility) — здатність однієї комп'ютеризованої системи приймати та опрацьовувати дані іншої системи. Здатність одного комплекту обладнання та програм бути внесеним до системи з іншим обладнанням та програмами.

трансля́тор, -а (рос. транслятор, англ. translator) — програма для переведення програм з однієї на іншу мову програмування, найчастіше з мови високого рівня на мову команд процесора ЕОМ.

трансля́ція (рос. трансляция, англ. translation) — перетворення програми, поданої однією мовою програмування, у рівнозначну програму іншою мовою.

укладач, -а (рос. компоновщик, англ. linkage editor, linker) — завантажувач, який у процесі завантажування виконує компонування однієї програми з незалежно трансльованих програм.

установлення (рос. инсталляция, англ. installation) — 1. Установлення програмного виробу на персональному комп'ютері. 2. Одне з обмежень на програмний виріб при продажу його фірмою, див. ще інсталяція.

утиліти (рос. утилиты, англ. utilities) — допоміжні програми.

фа́йл, -у (рос. файл, англ. file) — набір однотипних або пов'язаних між собою даних, який ідентифікується унікальним іменем, зберігається на зовнішньому носієві інформації та обробляється спеціальними програмами.

форма́т, -у (рос. формат, англ. format) — елемент мови, який у символьному вигляді описує подання інформаційних об'єктів у файлі.

форматува́ння (рос. форматирование, англ. formatting) — процедура розбивання доріжок магнітного диска на фізичні записи (блоки), що виконується перед першим використанням диска.

ци́кл, -у (рос. цикл, англ. loop) — набір команд, які можуть повторно виконуватись доти, доки діятиме певна умова.

чита́ння (да́них) (рос. чтение, англ. read) — у комп'ютерному опрацюванні — функція розпізнавання та зчитування даних у комп'ютері чи частині комп'ютера.

шабло́н, -у (рос. шаблон, англ. picture) — послідовність символів, яка визначає певне подання елемента даних. Служить для стандартного опрацювання масиву елементарних даних.

СПИСОК ЛІТЕРАТУРИ

1. Ашарина И. В. Основы программирования на языках С и С++ / Ашарина И. В. – М. : Горячая линия-Телеком, 2002. – 207 с.
2. Боэм Б. У. Инженерное проектирование программного обеспечения / Боэм Б. У.; пер. с англ. – М. : Радио и связь, 1985. – 512 с.
3. Давыдов В. Г. Программирование и основы алгоритмизации : учеб. пособие / Давыдов В. Г. – М. : Высш. шк., 2003. – 447 с.
4. Дубовой В. М. Програмування комп'ютеризованих систем управління та автоматики / В. М. Дубовой, Р. Н. Квєтний. – Вінниця : ВДТУ, 1997. – 208 с.
5. Иванова Г. С. Основы программирования / Иванова Г. С. : учебник для вузов. – 2-е изд., перераб. и доп. – М. : Изд-во МГТУ им. Н. Э. Баумана, 2002. – 416 с.
6. Информатика: Базовый курс / С. В. Симонович и др. – СПб. : Питер, 2003. – 640 с
7. Козловська Л. С. Короткий тлумачний словник з інформатики та інформаційних систем для економістів / Козловська Л. С. – К. : КНЕУ, 2003. Електронний ресурс. Режим доступу : http://6201.org.ua/load/72-1-0-631.
8. Макконнелл Дж. Основы современных алгоритмов / Дж. Макконнелл. - 2-е дополненное издание. - М. : Техносфера, 2004. - 368 с.
9. Мюррей У. Visual C++ / У. Мюррей, К. Паппас. Руководство для профессионалов: пер. с англ. – Спб. : BHV-Санкт-Петербург, 1996. – 912 с.
10. Петров В. Н. Информационные системы / Петров В. Н. – СПб. : Питер, 2003. – 688 с.
11. Павловская Т. А. С/С++: Программирование на языке высокого уровня : учебник / Павловская Т. А. – СПб. : Питер, 2001. – 464 с.
12. Подбельский В. В. Программирование на языке Си : учебное пособие / В. В. Подбельский, С. С. Фомин. – 2-е изд., доп. – М. : Финансы и статистика, 2002. – 600 с.
13. Руденко В. Д. Практичний курс інформатики / В. Д. Руденко, О. М. Макарчук. – К. : "Мін. Освіти", 1997. – 348 с.
14. Сван Т. Программирование на С / Сван Т. – М., СПб. : Издательский дом "Вильямс", 1999. – 543 с.
15. Седжвик Р. Фундаментальные алгоритмы на С++. Анализ / Структуры данных / Сортировка / Поиск / Седжвик Р. : Пер. с англ. – К. : Издательство "ДиаСофт", 2001. – 688 с.
16. Соммервилл И. Инженерия программного обеспечения / И. Соммервилл. – М. : Издательский дом "Вильямс", 2002. – 624 с.
17. Фельдман А. Я. Создаем информационные системы / Фельдман А. Я. – М. : СОЛОН-ПРЕСС, 2006. –120 с.

Навчальне видання

Москвіна Світлана Михайлівна
Грищук Тетяна Вікторівна

КОМП'ЮТЕРНІ ТЕХНОЛОГІЇ ТА ПРОГРАМУВАННЯ
Частина 4
ЕЛЕМЕНТИ ПРОФЕСІЙНОГО ПРОГРАМУВАННЯ

Навчальний посібник

Редактор В. Дружиніна

Оригінал-макет підготовлено Т. Грищук

Підписано до друку 03.11.2014 р.
Формат 29,7×42 1/4. Папір офсетний.
Гарнітура Times New Roman.
Друк різографічний. Ум. друк. арк. 12.5.
Наклад 75 прим. Зам. № 2014-089

Вінницький національний технічний університет,
навчально-методичний відділ ВНТУ.
21021, м. Вінниця, Хмельницьке шосе, 95.
ВНТУ, к. 2201.
Тел. (0432) 59-87-36.
Свідоцтво суб'єкта видавничої справи
серія ДК № 3516 від 01.07.2009

Віддруковано у Вінницькому національному технічному університеті
в комп'ютерному інформаційно-видавничому центрі
21021, м. Вінниця, Хмельницьке шосе, 95.
ВНТУ, ГНК, к. 114.
Тел. (0432) 59-87-38.
Свідоцтво суб'єкта видавничої справи
серія ДК № 3516 від 01.07.2009

Москвіна Світлана Михайлівна,
кандидат технічних наук, професор кафедри комп'ютерних систем управління Вінницького національного технічного університету. Автор понад 120 наукових робіт з напрямку математичне моделювання та обчислювальні методи.

Грищук Тетяна Вікторівна,
кандидат технічних наук, доцент кафедри комп'ютерних систем управління Вінницького національного технічного університету. Головний науковий напрямок: "Розробка моделей та методів проектування голосових інтерфейсів". Автор понад 25 публікацій.