С. В. Барабан, Р. С. Белзецький, І. Р. Арсенюк

Комп’ютерна інженерія та основи робототехніки

Міністерство освіти і науки України
Вінницький національний технічний університет

С. В. Барабан, Р. С. Белзецький, І. Р. Арсенюк

Комп’ютерна інженерія та основи робототехніки

Електронний навчальний посібник комбінованого (локального та мережного) використання

Вінниця
ВНТУ
2024

УДК 378.147
Б24

Рекомендовано до видання Вченою радою Вінницького національного технічного університету Міністерства освіти і науки України (протокол № 11 від 30.04.2024 р.).

Рецензенти:
Р. Н. Квєтний, доктор технічних наук, професор
А. Я. Кулик, доктор технічних наук, професор
М. Г. Тарновський, кандидат технічних наук, доцент

Барабан, С. В.

Б24 Комп'ютерна інженерія та основи робототехніки : електронний навчальний посібник комбінованого (локального та мережного) використання [Електронний ресурс] / Барабан С. В., Белзецький Р. С., Арсенюк І. Р. – Вінниця : ВНТУ, 2024. – 155 с.

Посібник присвячений матеріалам лекційного курсу з дисципліни «Комп'ютерна інженерія та основи робототехніки» для студентів, які навчаються за спеціальністю 122 «Комп'ютерні науки» денної форми навчання.

У посібнику наведено історію розвитку робототехніки, а також її застосування у різних сферах життєдіяльності людей. Детально розглянуто основи роботи з мобільною автономною платформою mbot у середовищах mBlock та Arduino IDE, наведено низку практичних схем. Також розглянуто основи програмування роботів із застосуванням популярної мови програмування С.

УДК 378.147

ЗМІСТ

ВСТУП

Ласкаво просимо у світ комп'ютерної інженерії та робототехніки, де передові технології зустрічаються з цариною можливостей. У цьому посібнику ми розпочинаємо надзвичайно цікаву подорож основами комп'ютерної інженерії та заглиблюємося у тонкощі робототехніки, досліджуючи її багату історію, практичне застосування та реалізацію на практиці.

Робототехніка – це галузь, яка стрімко розвивається та проникає фактично в усі сфери нашого життя. З кожним роком її популярність зростає, оскільки вона все сильніше впливає на наше повсякдення, роблячи його зручнішим, комфортнішим ефективнішим і безпечнішим. Така надзвичайно висока популярність робототехніки пояснюється такими двома факторами. По-перше, стрімкий технологічний прогрес, особливо у галузі штучного інтелекту, датчиків, матеріалознавства, мехатроніки забезпечує можливість створення дуже потужних та розумних роботів. Вони стають доступнішими, надійнішими та функціональнішими, що робить їх надзвичайно важливими для різних царин, від виробництва до медицини та військової сфери. По-друге, роботи стають критично важливим елементом у вирішенні завдань, які раніше були дуже складними, або взагалі недосяжними для людей. Наприклад, роботи в агропромисловості можуть покращувати врожайність, зменшувати негативний влив добрив як на врожай, так і на навколишнє середовище; у військовій сфері – допомогати ефективно виконувати різноманітні бойові завдання (зокрема, захист своїх територій, розвідку, знищення ворожих цілей, мінування або розміновування певних територій, корегування ведення артилерійського вогню тощо), зберігаючи при цьому життя людей; у медицині – виконувати ефективну діагностику та лікувати різні захворювання.

Крім того, робототехніка привертає увагу завдяки своїм потужним можливостям автономної робо*т*и та роботи у реальному часі. Роботи здатні виконувати складні завдання без участі людини, що робить їх корисними у ситуаціях, де безпека та швидкість вирішення завдання є критичними. Тож, робототехніка стає не лише популярною, а й невід'ємною частиною нашого життя, привносячи інновації, ефективність і багато нових можливостей.

У першому розділі посібника ми поринаємо у захопливі дослідження, які починаються з всебічного огляду історії та перспектив розвитку робототехніки. Від стародавніх автоматів до сучасних автономних систем ми простежуємо еволюцію робототехніки та досліджуємо її трансформаційний вплив на суспільство. Через цікаві розповіді та глибокий аналіз ми розкриваємо рушійні сили розвитку робототехніки та передбачаємо її багатообіцяюче майбутнє. Крім того, цей розділ заглиблюється у численні сфери застосування робототехніки у різних галузях. Від промислової автоматиза-

ції та охорони здоров’я до сільського господарства та геологічної розвідки, де роботи відіграють ключову роль у підвищенні продуктивності, ефективності та безпеки. На реальних прикладах ми ілюструємо різноманітні сфери застосування робототехніки та надихаємо читачів уявити безмежні можливості, які чекають на них попереду.

Другий розділ присвячено основам роботи з мобільною автономною платформою mbot у популярних середовищах mBlock та Arduino IDE. Розглянуто такі питання, як основні характеристики mBot, підключення mBot до персонального комп’ютера; керування різними світлодіодами, зокрема й світлодіодними матрицями; особливості керування серводвигунами; керування mBot за допомогою інфрачервоного пульта дистанційного керування; а також керування рухом mBot за допомогою модуля ультразвукового далекоміра.

Третій розділ посібника присвячено питанню основ програмування роботів на досить популярній мові програмування С. Зокрема розглянуто арифметичні вирази та їх особливості; типи даних і змінні; особливості реалізації різних алгоритмів розгалуження; особливості реалізації циклічних програм; основи роботи з масивами; основи застосування процедур і функцій та їх відмінності, а також поняття та особливості реалізації рекурсивних програм.

До кожного розділу наведено контрольні питання, що дозволять студентам контролювати як процес сприйняття матеріалу, так і його закріплення.

Отже, цей посібник пропонує всебічний вступ до комп’ютерної інженерії та робототехніки, охоплюючи історію, застосування та практичну реалізацію робототехніки. Незалежно від того, чи є Ви студентом, викладачем або ентузіастом робототехніки, посібник надає знання та інструменти, потрібні для того, щоб розпочати подорож до відкриттів та інновацій у захопливій галузі робототехніки.

1 ОСНОВИ РОБОТОТЕХНІКИ

1.1 Історія та перспективи розвитку робототехніки

XXI століття – це час змін, технологій та майбутнього. Особливо це відображається у розвитку науково-технічного прогресу, який став поштовхом для реалізації чогось надзвичайного в історії минулого.

Одним з таких явищ, яке поступово проникає в наше життя і відіграє все більш важливу роль, є виробництво роботів. Існує ціла прикладна наука під назвою робототехніка (robotics), яка займається розробкою та експлуатацією роботів і систем управління ними [1].

Рисунок 1.1 – Приклад взаємодії робота та людини

Існує багато дискусій щодо доцільності впровадження роботів у життя людини. Дехто вважає, що роботи – це позитивний актив для людства, а дехто вважає, що роботи – це неминуча і невідворотна смертельна загроза. Однак одне залишається зрозумілим: сьогоднішні технології перебувають на такому етапі розвитку, який робить можливим вдосконалення кібернетики для людини [2]. Іншими словами, технологічна еволюція в майбутньому поступово замінить біологічну еволюцію. Принаймні, ці зміни вже давно передбачені сучасним кінематографом.

Щороку на екрани виходять десятки фільмів, що зображують реальність життя в найближчі роки, де людина зображується як робот з незалежним штучним інтелектом, який не потребує допомоги живої людини.

Втім, не всі схильні вірити прогнозам кіно. Наприклад, роботехнік і творець робота-людини Хіроші Ішігуро. Його перше творіння повторювало його власну зовнішність. Хіроші заперечує існування робота без допомоги людини. Він закликає не дивитися американські фільми з нереалістичними сценаріями майбутнього.

Довіряти пророцтвам у фільмі чи ні – вибір кожного. Зараз ми можемо говорити лише про силу людського розуму та інтелекту, які можуть зробити машини схожими на людей. Але чи зможуть роботи існувати без людини – питання, на яке поки що немає відповіді [3].

Робототехніка – це прикладна наука, яка займається проєктуванням, розробкою, конструюванням, експлуатацією та використанням роботів, а також комп'ютерних систем управління, сенсорного зворотного зв'язку (на основі вихідних датчиків) і обробки інформації автоматизованих технічних систем (роботів) [4].

Робототехніка займається створенням роботів і робототехнічних систем для автоматизації складних технічних процесів і операцій, зокрема тих, що виконуються в недетермінованих умовах, з метою заміни людини на важких, дорогих і небезпечних роботах [5]. Роботи можуть мати будь-яку форму, але деякі з них зроблені схожими на людей. Вважається, що це допомагає зрозуміти робота з певною поведінкою копіювання, яка загалом характерна для людини. Такі роботи намагаються повторити все, що може робити людина, а саме: ходити, піднімати, косити.

Основна задача робототехніки – програмування задля контрольованої співпраці електроніки та механіки роботів [6].

Роботів також можна розділити на керованих і автономних; мобільних і стаціонарних [7]. Роботи можуть використовуватися в будівництві, промисловості, побуті, авіації та в екстремальних умовах (військові, космічні, підводні).

Термін «робототехніка», який перетнув сферу наукової фантастики і увійшов у реальне життя, був винайдений у 1942 році Айзеком Азімовим, американським вченим і популяризатором наукової фантастики. Слово «робототехніка» походить від слова «робот», яке представив публіці чеський письменник Карел Чапек у своїй п'єсі «R. U. R.» (Rossum's Universal Robots) 1920 року [3]. Слово «робот» походить від словацького слова «robot», що означає робота. Події відбуваються на фабриці, яка виробляє штучних людей (роботів), істот, яких можна помилково прийняти за людей (дуже схоже на сучасне уявлення про андроїдів). Сам Карел Чапек слово «робот» не вигадував. Це зробив Йозеф Чапек – його рідний брат.

Перші експерименти з машинами проводилися ще в Античні часи. Наприклад, знаменита музична машина Герона Александрійського або літаючий голуб Архіта. У III столітті до нашої ери з'явився один з перших описів автоматизації у роботі Лі Цзи. Із занепадом античної культури, наукові свідчення того часу також тимчасово зникли.

Галузь робототехніки пройшла дивовижний шлях інновацій та еволюції, що охоплює тисячоліття людської винахідливості та технологічного прогресу. Від найперших механічних пристроїв стародавніх цивілізацій до найсучасніших автономних систем сучасності [8] – робототехніка постійно розширює межі можливого у взаємодії людини і машини.

Період Відродження став свідком відродження інтересу до автоматів і механічних пристроїв, який підживлювався творчим генієм таких винахідників, як Леонардо да Вінчі.

Серед помітних досягнень цієї епохи – проєкти людиноподібних роботів і програмованих машин да Вінчі, а також реалістичні автомати, створені Жаком де Вокансоном і П'єром Жаке-Дрозом.

Поява промислової революції стала поворотним моментом в історії робототехніки, оскільки механізація і автоматизація трансформували виробничі процеси і трудові практики.

У 1942 році письменник-фантаст Айзек Азімов сформулював три закони робототехніки [3] для керування «розумними» машинами:

- робот не може завдати шкоди людині або, якщо не працює, завдати травм людям;
- р*о*боти мають підкорятися командам людини, за винятком тих, що порушують перший закон;
- робот має захищатися, якщо його дії не суперечать першому і другому законам.

Ці три фундаментальні закони складають основну тему історій про роботів Айзека Азімова. Письменник створив особливу професію – робота-психолога, експерта, який пояснює поведінку роботів на основі трьох законів робототехніки. Сьюзен Келвін – чудовий робот-психолог.

Після закінчення Другої світової війни у сфері робототехніки спостерігався швидкий розвиток.

У 1948 році американський математик, який створив основу кібернетики, Норберт Вінер, сформулював принципи кібернетики, які лягли в основу практичної робототехніки.

Протягом другої половини XX століття дослідники досягли значних успіхів у теорії управління, сенсорних технологіях та в інформатиці, що дозволило роботам виконувати все більш складні завдання з більшою точністю та автономністю [9].

Прориви у кінематиці роботів, плануванні руху та інтеграції датчиків заклали основу для розробки автономних й інтелектуальних робототехнічних систем, здатних сприймати навколишнє середовище та взаємодіяти з ним [10].

На початку XXI століття широкого поширення набули комерційні та промислові роботи, які можуть дешевше, точніше та надійніше, ніж люди, виконувати різноманітні завдання (рис. 1.2). Вони також використовуються для завдань, які є занадто брудними, небезпечними або трудомісткими для людей [11]. Роботи працюють у виробництві, транспортуванні, дослі-

дженні Землі та космосу, хірургії, військовій сфері, лабораторних дослідженнях, безпеці та масовому виробництві споживчих і промислових товарів [12].

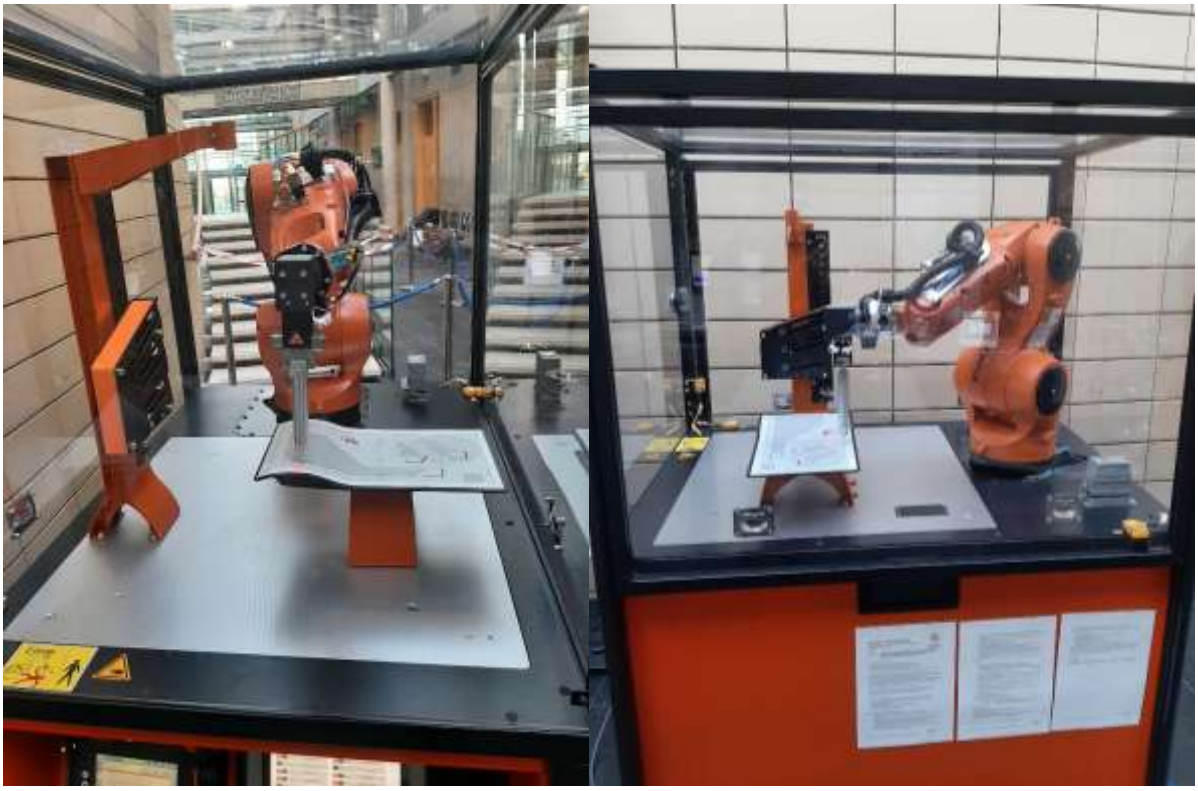


Рисунок 1.2 – Приклад сучасних промислових роботів

За останні десятиліття робототехніка вийшла за межі промислового застосування і охопила широкий спектр сервісних роботів, призначених для допомоги людині в різних сферах.

Сервісні роботи, зокрема домашні помічники, медичні компаньйони та автономні дрони, стали невід'ємною частиною таких секторів, як охорона здоров'я, сільське господарство, логістика та індустрія розваг.

Існує безліч типів роботів, які використовуються в різних середовищах і сферах застосування. Незважаючи на їх різноманітні функціональні можливості, всі роботи мають три фундаментальні структурні подібності: механічну, електричну та комп'ютерну.

Роботи мають механічний каркас або конфігурацію, пристосовану для виконання конкретних завдань. Наприклад, робот, призначений для пересування по густій багнюці, може використовувати гусениці. Механічний аспект слугує основним фактором, на який розробники звертають увагу при виконанні завдань, дотримуючись принципу «форма слідує за функцією».

Роботи містять електричні компоненти, що відповідають за керування механізмами. Наприклад, робот, оснащений гусеницями, потребує електричної енергії, щоб приводити до руху протектори. Ця енергія подається

через дроти від акумулятора, що є первинним електричним колом. Електричні компоненти забезпечують рух (за допомогою двигунів), зондування (використання електричних сигналів для вимірювання таких параметрів, як тепло, звук, положення та енергетичний стан) і роботу (роботи покладаються на електричну енергію для живлення двигунів і датчиків, щоб виконувати основні функції).

Кожен робот використовує певний рівень комп'ютерного програмування, що диктує його дії та поведінку. Програмування є критично важливим елементом функціональності робота; навіть при бездоганній механічній та електричній конструкції некоректно складена програма може серйозно погіршити продуктивність або зробити робота нездатним виконувати завдання. Існує три основні типи робототехнічних програм: дистанційне керування, штучний інтелект і гібридні. Роботи з дистанційним керуванням дотримуються заздалегідь визначеного набору команд, виконуючи їх після отримання сигналів від джерела управління, як правило, від людини-оператора, що використовує пульт дистанційного керування. Роботи зі штучним інтелектом автономно взаємодіють з навколишнім середовищем, реагуючи на виклики, що виникають, на основі запрограмованих алгоритмів. Гібридне програмування поєднує в собі елементи штучного інтелекту і дистанційного керування, пропонуючи поєднання автономної роботи і втручання людини.

З поширенням спеціалізованих роботів, пристосованих для виконання певних завдань, потреба в системі класифікації стає все більш нагальною. Наприклад, деякі роботи розроблені спеціально для виконання завдань зі складання і не мають універсальності для альтернативних застосувань, тому їх називають «складальними роботами». Для зварювання швів виробники пропонують комплексні роботизовані зварювальні системи, що складаються зі зварювальних апаратів і таких додаткових компонентів для переміщення матеріалів, як поворотні столи, інтегровані в єдиний блок, відомий як «зварювальний робот». Аналогічно, роботів, спроєктованих для роботи зі значними навантаженнями, влучно називають «роботами для важких умов експлуатації».

У даному посібнику наведено комплексну класифікацію робототехніки, яка розмежовує різні категорії на основі функціональних можливостей, сфер застосування та технологічних атрибутів. Завдяки вичерпному дослідженню різних типів роботів та їхніх відповідних характеристик, ця робота має на меті забезпечити детальне розуміння різноманітного ландшафту робототехніки та її незліченних застосувань у різних галузях і секторах.

Робототехніка охоплює широкий спектр машин, призначених для автономного або напівавтономного виконання завдань у різних середовищах. Для розуміння можливостей, обмежень і сфер застосування різних типів роботів потрібна упорядкована система класифікації.

Класифікація на основі функціональності: роботів можна класифікувати на основі їхньої основної функціональності, охоплюючи маніпуляції, локомоцію, зондування та прийняття рішень.

Маніпуляційні роботи (рис. 1.3) призначені для виконання таких завдань, як захоплення, підйом і складання об’єктів [13], тоді як роботи для переміщення (рис. 1.4) зосереджені на мобільності на різних поверхнях [14].

Рисунок 1.3 – Приклад робота-маніпулятора
(зображення згенеровано штучним інтелектом Deep Dream Generator)

Рисунок 1.4 – Приклад мобільного робота

Сенсорні роботи (рис. 1.5) оснащені датчиками для сприйняття та інтерпретації навколишнього середовища, в той час як роботи, що приймають рішення (рис. 1.6), використовують алгоритми та штучний інтелект для прийняття автономних рішень.

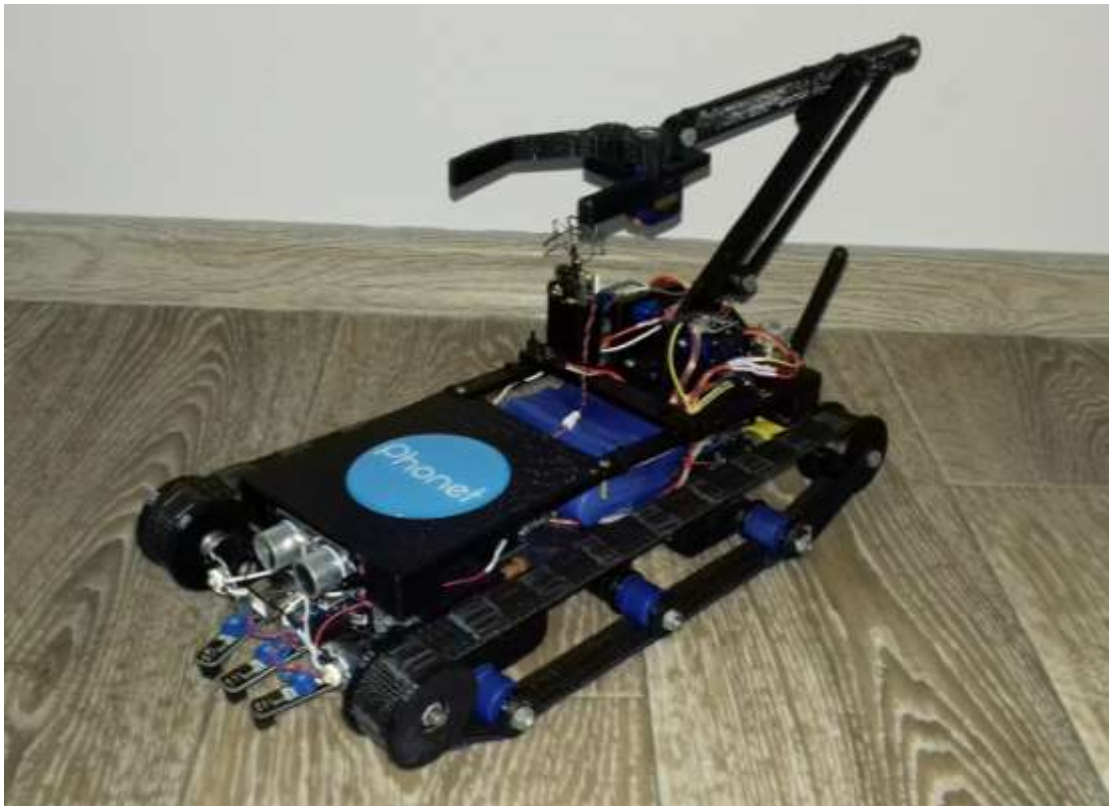


Рисунок 1.5 – Приклад сенсорного робота

Рисунок 1.6 – Приклад робота зі штучним інтелектом

Класифікація на основі сфер застосування: робототехніка знаходить застосування у широкому спектрі галузей, охоплюючи промислову автоматизацію (рис. 1.7), охорону здоров'я, сільське господарство, логістику та оборону.

Промислові роботи використовуються у таких виробничих процесах, як складання, зварювання, фарбування і переміщення матеріалів, оптимізуючи ефективність і продуктивність у заводських умовах.

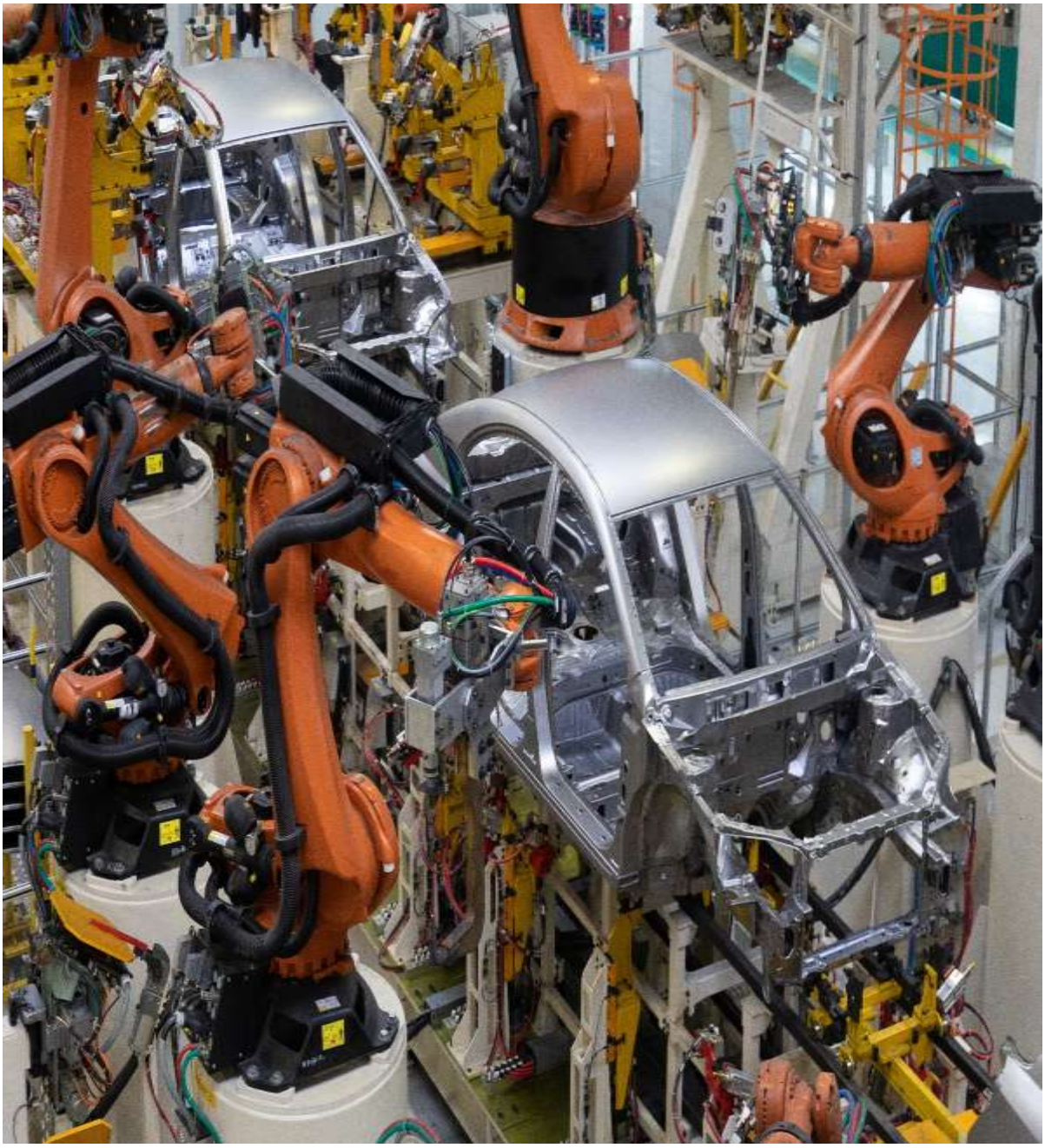


Рисунок 1.7 – Приклад роботизованого автомобільного виробництва

Медичні роботи допомагають хірургам у таких процедурах, як малоінвазивна хірургія, реабілітація (рис. 1.8) та телемедицина, підвищуючи точність і покращуючи результати лікування пацієнтів.

Рисунок 1.8 – Приклад застосування медичного робота для реабілітації

Сільськогосподарські роботи автоматизують такі завдання, як саджання, збирання врожаю і моніторинг посівів, підвищуючи врожайність і знижуючи витрати на робочу силу в сільському господарстві.

Логістичні роботи (рис. 1.9) сприяють автоматизації складів, охоплюючи сортування товарів, управління запасами і виконання замовлень, оптимізуючи операції ланцюга поставок.

Рисунок 1.9 – Приклад робота (kerfus),
що розвозить товари в супермаркеті Carrefour

Класифікація за технологічними ознаками: роботів також можна класифікувати на основі їхніх технологічних характеристик, серед них мобільність, рівень автономності та фізична форма.

До мобільних роботів відносять колісні, ножні та повітряні платформи, кожна з яких підходить для різних середовищ і застосувань.

Автономні роботи (рис. 1.10) працюють з різним ступенем незалежності, починаючи від систем з телеуправлінням і закінчуючи повністю автономними агентами, здатними приймати рішення і навчатися.

Рисунок 1.10 – Приклад автономного робота
(kettybot у ТРЦ «Posnania»)

Фізичні форми роботів охоплюють традиційні жорсткі конструкції, а також нові технології м’якої робототехніки, пропонуючи гнучкість і адаптивність до різноманітних завдань і середовищ.

Міжгалузева класифікація: багато роботів мають характеристики, які охоплюють кілька класифікаційних категорій, що зумовлює потребу у застосуванні міжгалузевого підходу до класифікації.

Наприклад, мобільний робот (рис. 1.11), оснащений маніпуляційними можливостями, може бути класифікований як на основі його мобільності (наприклад, колісний), так і на основі його функціональності (наприклад, маніпуляції).

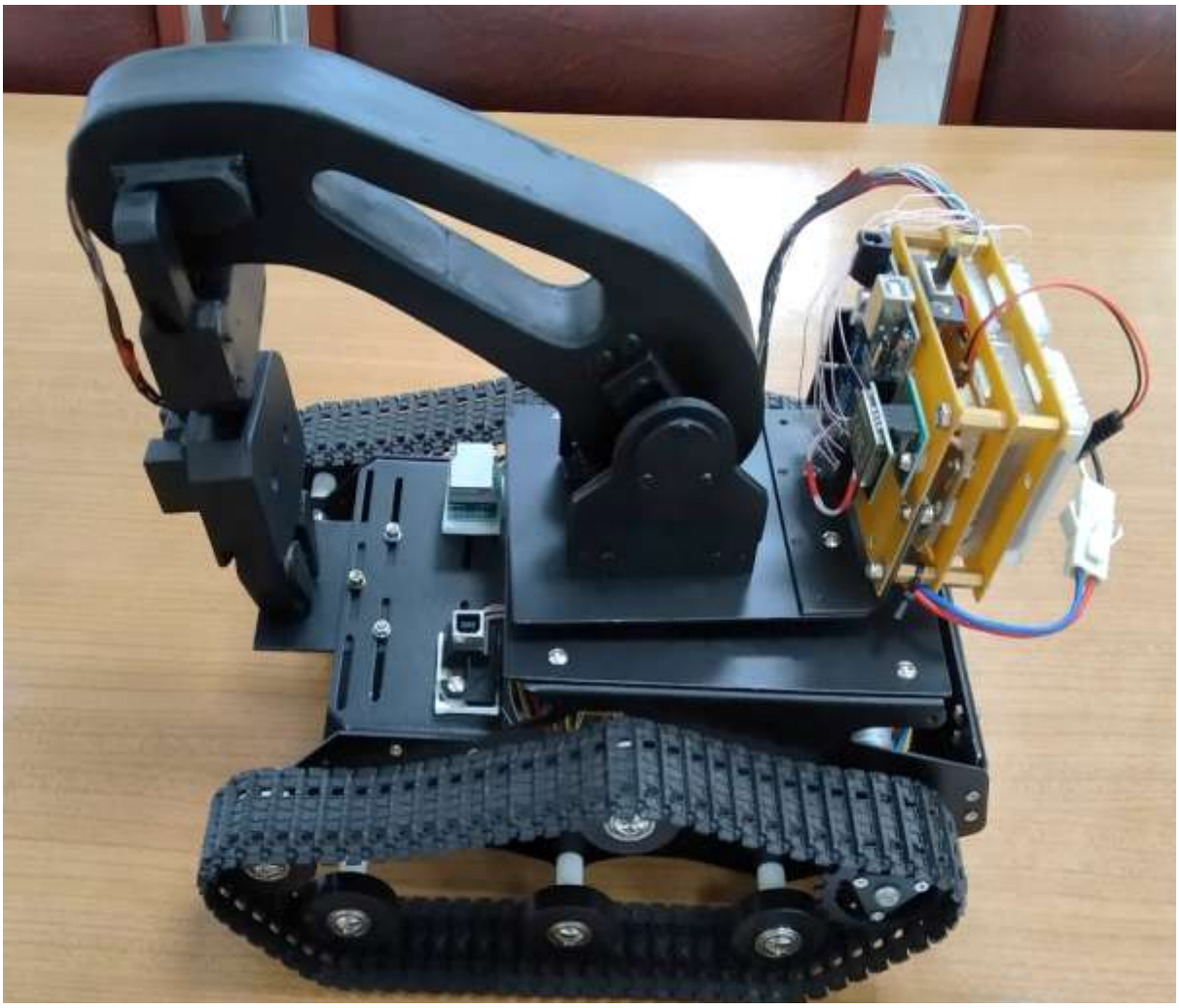


Рисунок 1.11 – Приклад робота, що поєднує мобільність
та маніпуляційні можливості

Всеосяжна класифікаційна структура має важливе значення для розуміння різноманітності і складності робототехніки та її застосування у різних сферах.

Систематично класифікуючи роботів за функціональністю, сферами застосування та технологічними характеристиками, дослідники і практики можуть краще використовувати технології робототехніки для вирішення реальних проблем і стимулювання інновацій у цій галузі.

Роботи стали поширеними для навчальних цілей у закладах початкової (рис. 1.12), середньої та вищої освіти, а також у численних літніх таборах молоді (рис. 1.13), підвищуючи інтерес у молоді до програмування, штучного інтелекту та робототехніки.

Рисунок 1.12 – Приклад майстер-класу
навчання вчителів робототехніки молодших класів у Польщі

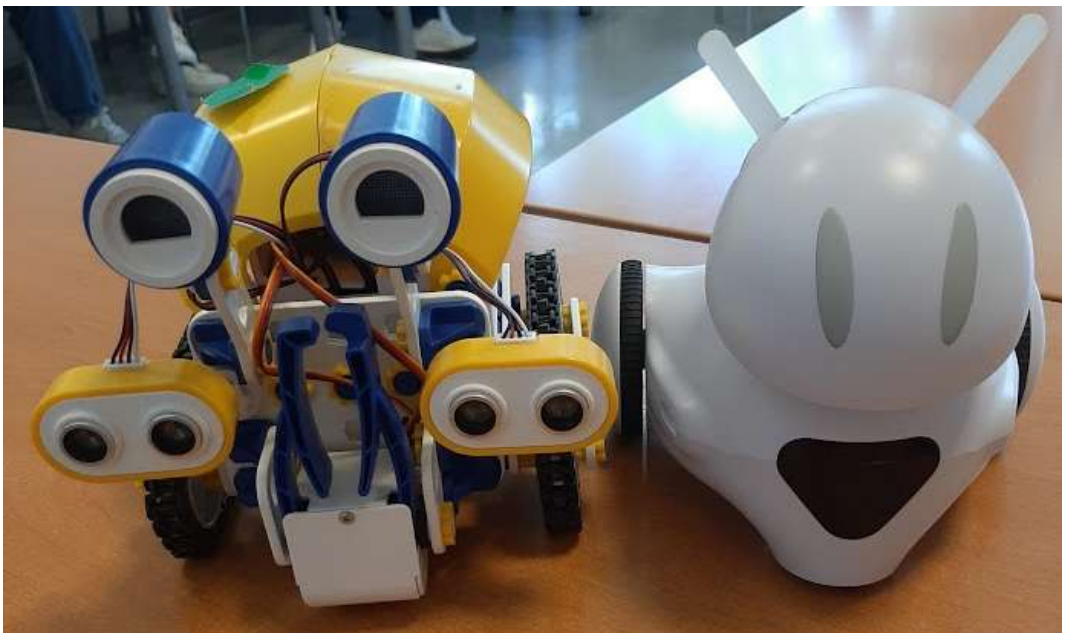


Рисунок 1.13 – Приклад роботів для STEM освіти
у початкових класах польських шкіл

Якщо у школах програмування викладають традиційними «абстрактними» методами, для більшості дітей предмет стає дуже важким і нудним. Навчаючись через гру, керуючи роботом і розуміючи, які процеси є правильними, а які – неправильними, учні отримують безпосередній досвід і розуміння керування робототехнічними системами (рис. 1.14).

Рисунок 1.14 – Приклад навчальної аудиторії з вивчення робототехніки

Оскільки робототехніка зараз ще знаходиться на стадіях свого розвитку, є всі передумови для того, щоб вона стала ще більш популярною як шкільний предмет і охопила максимальну кількість учнів. Робототехнічні дослідження тепер набагато молодші, що дозволяє кожному учневі стати справжнім вченим і дизайнером.

Платформи mBot ти Arduino є чудовими стартовими платформами для розвитку учнівських навичок робототехніки. Це невеликий комп'ютер, який може програмувати дії, що керують будь-яким механізмом, наприклад, складним конструктором або роботом. Програмувати електронні проєкти можуть діти будь-якого віку, але складність залежатиме від їхніх умінь і знань. Також можна створювати власні моделі, корисні для дому та відпочинку, які виконують різноманітні функції. Потрібно вивчити мови програмування C++ і Scratch для Arduino.

1.2 Застосування роботів у готельному бізнесі та туризмі

Після COVID-19 подорожі здавалися неможливими у короткостроковій перспективі, але зараз є країни, які почали з того, що відкрили підприємства для внутрішнього туризму, а потім і свої кордони для міжнародних відвідувачів. Сьогодні у деяких європейських містах здається, що пандемії ніколи не було, люди на вулицях більше не бояться. Якби не миття рук у кожному під'їзді та не кілька офіціантів у масках, люди забули б про нещо-

давню пандемію коронавірусу. Однак в інших регіонах ситуація не така. Країни Латинської Америки все ще стикаються з найвищим рівнем епідемії, а країни Азії стикаються з наступними хвилями епідемій, в яких влада вже знову запроваджує обмеження та заходи безпеки [15]. Проблема все ще існує і характеризується невизначеністю. Ніхто не знає напевно, чи буде новий спалах, і якщо так, то яких заходів слід вжити. Що ми знаємо, так це те, що туристичний сектор не витримає подальшого закриття, так само як і наше психічне здоров'я. Гостинність і туризм мають відновити довіру. Можливо, недостатньо встановити таблички, щоб інформувати відвідувачів про покращені протоколи прибирання, соціальне дистанціювання та рекомендації. Занепокоєні туристи очікують більшого. Вони хочуть бачити реальні зміни, щоб насолоджуватися часом, проведеним далеко від дому. Чим менше контакту з людським тілом, тим краще.

Безконтактний досвід може стати конкурентною перевагою для туристичних напрямків. Доцільно рухатися у цьому напрямку, якщо дестинація хоче підготуватися до наступних кількох років, коли пандемії та обмеження соціальної дистанції є можливими і ймовірними. Китай йде на крок попереду, тестуючи цифрову валюту і просуваючи нову «безконтактну економічну систему», де все можна зробити без зустрічі. Проривні технології, які дозволяють здійснювати безконтактні платежі, можуть допомогти відновити довіру туристів, але існує набагато більше точок дотику, які потребують вирішення [16]. Від реєстрації в готелі до таких простих завдань, як замовлення їжі в ресторані або бронювання турів. Потрібно допомогти споживачам і автоматизувати ці процеси за допомогою таких технологій, як голосові команди і роботи. Готельні номери управляються голосом, тому немає необхідності в управлінні телевізором або кондиціонером, автономні пилососи для підлоги, дрони для доставки продуктів, робот-консьєрж, який може дати вам пораду на день. Боти і віртуальні помічники зі штучним інтелектом можуть взаємодіяти з туристами у режимі реального часу, надавати їм інформацію або навіть бути їхніми гідами. Туристи мають відчувати, що технологія піклується про них, як добрий господар.

Здоров'я людини – безцінний скарб. Це найважливіша потреба людини. Саме здоров'я багато в чому визначає можливості людини. Так само, підтримання здоров'я на належному рівні – досить складне і комплексне завдання. Розглянемо одну з важливих складових цього завдання – моніторинг температури тіла людей у закладах освіти, торгових центрах, вокзалах, аеропортах, розважальних центрах та інших людних місцях з метою своєчасного виявлення потенційно хворих людей. Це особливо важливо у наш час, коли існує багато різних (зокрема смертельних) інфекційних захворювань, а також нових, таких досить небезпечних хвороб, як різні штами COVID-19, SARS-CoV-2 та інші. Досить детальна інформація про COVID-19 та SARS-CoV-2 наведена, наприклад, на сайтах Всесвітньої організації охорони здоров'я [17] та Європейського центру з профілактики та контролю захворювань [18]. Отже, головне завдання тут – максимально

зменшити поширення хвороб і якнайшвидше розпочати лікування хворих. Вимірювання температури, безумовно, надзвичайно актуальне для сфери гостинності, де моніторинг температури пацієнта дозволяє вчасно і правильно визначити і скоригувати процес лікування.

Досить часто найбільш використовуваним приладом для вимірювання температури людського тіла сьогодні є звичайний ртутний термометр. При своїй високій точності та невисокій ціні він має суттєві недоліки. Це висока токсичність ртуті, низька надійність скляного корпусу та досить тривалий (до 10 хвилин) час вимірювання температури. Останній недолік стає особливо помітним при вимірюванні температури у немовлят. Або, наприклад, при вимірюванні температури великої кількості людей в різних точках. Повна гігієнічність і практична миттєвість (2–7 секунд) безконтактного методу вимірювання температури тіла вже давно зробили безконтактні термометри одними з найперспективніших [19]. Незважаючи на розвиток технологій та ринку безконтактних інфрачервоних термометрів, вони залишаються лише перспективними, а прогнози щодо заміни ртутних термометрів у медичному застосуванні залишаються, здебільшого, на рівні дискусій. Основні проблеми безконтактних термометрів все ще залишаються невирішеними. Це досить низька точність вимірювання (похибка може досягати ± 0,50 С) та дуже висока ціна (удвічі дорожча за ртутні термометри) [20].

Характеристики існуючих безконтактних інфрачервоних термометрів можуть бути покращені за рахунок використання автоколивальних систем, заснованих на реактивних властивостях елементів з від'ємним опором. Такі системи мають низку переваг, а саме: високу завадостійкість (забезпечує високу точність вимірювання), потужний вихідний сигнал (що дозволяє відмовитися від прецизійних підсилювачів), простоту конструкції (підвищує ефективність пристроїв, побудованих на їх основі). Ми пропонуємо наділити робота-консьєржа функцією безконтактного вимірювання температури тіла людини з можливістю надання рекомендацій персоналу готелю щодо осіб, у яких виявлена температура, залежно від її значення. Таким чином, до звичайних сервісних функцій роботів-консьєржів буде додано функціонал охорони здоров'я.

Інтеграція робототехніки в індустрію гостинності привертає значну увагу в останні роки. Роботи-консьєржі, призначені для допомоги клієнтам готелів і курортів, є перспективною технологічною розробкою. Роботи-консьєржі – це інтерактивні помічники, які пропонують гостям інформацію та різноманітні послуги. Завдяки використанню штучного інтелекту (ШІ) та обробки природної мови (ОПМ) (Natural language processing, NLP) ці роботи можуть розуміти запити та ефективно орієнтуватися у навколишньому середовищі. Дослідження показують, що роботи-консьєржі позитивно впливають на задоволеність гостей, пропонуючи персоналізовані рекомендації та швидке обслуговування. Підвищення зручності у доставленні речей та покращення комунікації сприяють позитивному досвіду гостей.

Роботи-консьєржі продемонстрували потенціал для підвищення операційної ефективності за рахунок скорочення трудовитрат, пов'язаних із виконанням рутинних завдань. Їх доступність гарантується 24 години на добу, 7 днів на тиждень і забезпечує безперервне обслуговування без обмежень у людських ресурсах [21].

У реальному світі було розроблено і впроваджено кілька роботів-консьєржів у різних сферах гостинності. Ось кілька чудових прикладів.

1) Connie від готелю Hilton працює на базі штучного інтелекту IBM Watson і виконує функції інтерактивного консьєржа. Він допомагає гостям, надаючи інформацію про послуги готелю, місцеві визначні пам'ятки, ресторани та інші запити. Використовуючи обробку природної мови (NLP), Connie спілкується з гостями, розуміє їхні запитання та ефективно реагує на них [22].

2) Botlr готелю Aloft призначений для виконання завдань доставляння у приміщеннях готелю. Він самостійно пересувається коридорами та ліфтами, щоб доставити послуги або продукти у номери. Завдяки датчикам і картографічним технологіям Botlr забезпечує безпечну та ефективну доставку без втручання людини [23].

3) YO2D2 від Yotel зосереджується переважно на оперативному виконанні запитів гостей. Як і Botlr, YO2D2 орієнтується на території готелю, щоб доставити рушники, товари для ванної кімнати або інші потрібні речі у номери. Його функціональність зосереджена на підвищенні комфорту та задоволеності гостей завдяки ефективному наданню послуг [24].

4) Marriott's Mario розгорнутий компанією Marriott International, подібно до Botlr та YO2D2. Він допомагає доставляти предмети в номер і демонструє потенціал роботизованих послуг в індустрії гостинності. Його основні функції – допомога персоналу у виконанні рутинних завдань з метою підвищення ефективності [25].

5) Роботи у готелі Henn-na виконують різні функції, а саме: роботи-реєстратори здійснюють реєстрацію гостей, а пасажирські роботи допомагають із перевезенням багажу; робот-консьєрж надає гостям інформацію, наприклад, про місцеву погоду, інформацію про транспорт і найближчі визначні пам'ятки. Вони покликані залучати гостей до розмов і допомагати у виконанні запитів за допомогою технологій штучного інтелекту і розпізнавання мови [26].

Таким чином, сучасні роботи-консьєржі, зазвичай, використовують поєднання ШІ, ОПМ, датчиків (таких як камери, лідари та ультразвукові датчики), картографічних і навігаційних систем, технологій бездротового зв'язку, а іноді ще й алгоритмів машинного навчання, щоб покращити свою функціональність і надавати ефективні послуги гостям готелю.

1.3 Використання роботів у військовій справі

Використання роботів у військовому контексті є проблематичним на

багатьох рівнях. Існують соціальні, правові та етичні питання, які потрібно обговорити перед їх широким розгортанням.

Останніми роками ми спостерігаємо зростання інтересу до використання роботів і соціальних, правових та етичних аспектів їх застосування [27]. Конкретні питання стосуються окремих сфер, в яких застосовуються роботи, наприклад, існують специфічні проблеми з автономними автомобілями або сексуальними і любовними роботами, роботами-компаньйонами, медичними роботами, роботами-поліцейськими.

Розвиток технологій впливає на сучасні військові операції, які практично неможливо виконувати без опори на проривні технології, розвиток яких спрямований на подолання розриву у військових можливостях [28]. Військові р*о*боти все частіше використовуються в усіх сферах бойових дій, а саме: у повітрі, на суші, на воді і в кіберпросторі. Деякі з них створені за зразком тварин (наприклад, змій, комах, птахів, риб, морських і наземних ссавців), причому дедалі більше з них мініатюризуються та об'єднуються в рої, керовані однією людиною-оператором. Незважаючи на існування різних наративів щодо майбутньої роботизованої війни, на думку експертів, найбільш вірогідним є сценарій, в якому роботизовані системи діятимуть як коботи, підтримуючи, а не замінюючи дії солдатів-людей [29].

Найбільші військові та технологічні держави (Китай, Ізраїль, Росія, США) створили спеціальні підрозділи в уряді, відповідальні за інтеграцію алгоритмів, ШІ та машинного навчання у військові операції [30]. Таким чином, ми переживаємо нову гонку озброєнь [31].

Існує багато потенційних правових, соціальних і етичних проблем, пов'язаних з військовими роботами. Перша проблема – етична, яка зосереджена на цінності людського життя. Друга пов'язана з людською подобою як проблемою міжнародного гуманітарного права (МГП). Ці дві рамки не зовсім узгоджуються між собою – під час збройного конфлікту вбивство комбатантів (людей-солдатів) є юридично дозволеним.

На полі бою ще немає людиноподібних військових роботів. Однак можна припустити, що такі роботи можуть з'явитися в майбутньому. Людиноподібних роботів починають застосовувати і в інших сферах (секс-роботи, медичні роботи). Наше середовище пристосоване для людей з певним зростом, ногами і руками (сходи, двері, існуюче обладнання і т. д.). Це може бути фактором, що впливатиме на процес розробки роботів. Концепція людиноподібного солдата також закладена в культурі – у фільмах і книгах. Більш того, робот Atlas, людиноподібний робот компанії Boston Dynamics, був розроблений для американського військового агентства DARPA [32]. На нашу думку, військові роботи не мають виглядати як люди, оскільки це створює додаткові ризики для життя людей, що, в певному сенсі, суперечить головному обґрунтуванню використання роботів у військовому контексті. Роботи, схожі на людей, можуть бути легко прий-няті за людей, а їхня людська подоба може викликати психологічні реакції у людей, які наражають на небезпеку інших людей, що знаходяться поруч.

Одне з популярних означень робота відносять до парадигми «відчуття-мислення-дія» [33]. Якщо коротко, то під «відчуттям» маються на увазі можливості отримання інформації про зовнішній світ, під «дією» – здатність впливати на цей світ, а під компонентом «мислення» – можливості аналізу інформації та перетворення її на дії. Останній елемент пов'язаний з автономією. Тут ми розглядаємо роботів у широкому сенсі. Роботи охоплюють (потенційно) повністю автономні об'єкти (можливо, на основі ШІ) і керовані людиною одиниці з невеликою автономією або взагалі без неї (дрони). Під «військовими» ми маємо на увазі роботів, які використовуються у військовому контексті, особливо під час ведення бойових дій.

Розвиток роботів на полі бою залежить від розвитку, насамперед, ШІ. Деякі означення роботів прямо називають їх втіленим ШІ [34]. Проблемні питання, пов'язані з ШІ на полі бою, можуть бути посилені темами, пов'язаними з дизайном втілених агентів [35].

Дискусія про використання роботів у військових цілях набрала обертів з широким застосуванням дронів (безпілотних, переважно літальних апаратів). Масштабне застосування дронів, особливо в неміжнародних збройних конфліктах, сприяло обговоренню правових та етичних аспектів використання нових технологій на сучасному полі бою. Зокрема, було звернуто увагу на посилення асиметричного характеру війни у випадку конфліктів між технологічно розвиненою державою та державами, що розвиваються, або недержавними суб'єктами.

З правової точки зору, використання дронів вплинуло на розуміння таких понять, як застосування сили та обсяг права на самооборону, нагляд за процесом наведення на ціль з боку людини-оператора (human in the loop) або просто на часові та географічні рамки застосування міжнародного права. Водночас значне місце у доктринальних дебатах приділяється етичним і психологічним аспектам дистанційної війни. Прогресуюча дегуманізація поля бою, яка є результатом все більшого використання алгоритмічних процесів і усунення солдатів-людей із поля бою, стала джерелом питань про етику сучасних воїнів або про моральну допустимість дистанційного ураження противника.

Тим не менш, вже більше десяти років у темі проривних військових технологій домінує розвиток ШІ. На відміну від безпілотників, процес прийняття рішень у військових роботах, оснащених ШІ, залишатиметься поза людським контролем («людина поза циклом»). Це викликає низку правових і етичних проблем, які потрібно розглядати у контексті обіцянок ШІ щодо ефективності і корисності.

Хоча правові дебати щодо військових роботів найбільше розгорнулися навколо роботів-убивць (які розглядаються нижче), це не єдине застосування ШІ для військових. Насправді, найбільшою сферою застосування ШІ для військових є підтримка прийняття рішень, а не прийняття рішень. Збирання і аналіз великих обсягів даних, отриманих із внутрішніх джерел і складної оперативної обстановки на полі бою, може внести ясність щодо

категоризації певних об’єктів і осіб, виявлення аномалій і прогнозування можливих сценаріїв розвитку подій. Дікс розглядає використання систем підтримки прийняття рішень на основі ШІ в умовах збройного конфлікту для таких завдань, як, наприклад, перегляд рішень про затримання і звільнення, розпізнавання загроз або оцінювання пропорційності (сутність оцінювання пропорціності буде розглянуто дещо нижче).

Кінцеві результати роботи систем із ШІ мають доповнювати правові рішення, а не замінювати їх. Такі застосування ШІ покликані допомогти подолати людські обмеження у здатності швидко аналізувати великі обсяги даних і, за задумом, гармонізувати та стандартизувати інтерпретацію рішень, що ухвалюються. Хоча у таких системах зберігається людський контроль і прийняття рішень, це не означає, що вони залишаються безпроблемними. Першою проблемою є загальна кодованість МГП, що регулює ведення воєнних дій, яке значною мірою складається з дуже залежних від контексту, відкритих за структурою і, отже, дуже невизначених норм. Другою найважливішою проблемою є непрозорість процесу функціонування ШІ. З огляду на це, важливо вказати, що досі незрозуміло, як людські суб’єкти переходять від якісних суджень до кількісних (наприклад, визначення покарання з огляду на доведені обставини злочину, визнання військовим командиром того, що запланована атака відповідає принципу пропорційності). У цьому контексті можна вважати, що людське судження також не є прозорим, хоча ми все ще приймаємо його роль більше, ніж рішення, прийняті ШІ.

Найсуперечливішим застосуванням роботів на полі бою є летальні автономні системи озброєння (ЛАСО) (lethal autonomous weapons systems, LAWS). З 2013 року вони обговорюються експертами та державами у рамках Конвенції Організації Об’єднаних Націй про конкретні види звичайної зброї, укладеної у Женеві 10 жовтня 1980 року, яка обмежує та забороняє використання певних видів озброєнь. Незважаючи на відсутність офіційних переговорів щодо договірних рішень, форум ЛАСО слугує глобальним майданчиком для обговорення передачі рішень про життя і смерть людини, а отже, і процесу націлювання, до ШІ. Найбільшим досягненням цього процесу стало ухвалення у 2019 році 11 юридично необов’язкових Керівних принципів щодо розробки та використання ШІ. Поза сферою інтересів залишилися інші застосування, такі як вищезгадані системи підтримки прийняття рішень або використання військових роботів у рятувальних операціях, логістиці та транспортуванні, знешкодженні бомб або моделюванні бойових дій та підготовці солдатів. Передбачається, що під ЛАСО розуміються такі системи озброєнь, які після активації можуть ідентифікувати, вибирати і вражати цілі зі смертоносною силою без подальшого втручання оператора, хоча індивідуальні позиції держав щодо цього можуть дещо відрізнятися.

З огляду на широту застосування ШІ у військовій сфері, дискусії на форумі ЛАСО – це лише частина порушених питань. Водночас вони охоп-

люють той елемент, який має вирішальне значення для всього підходу людства до ШІ. Процес націлювання може призвести до позбавлення життя і тому викликає найбільше занепокоєння з правової та етичної точок зору. Саме тому в дискусіях щодо ЛАСО беруться до уваги оперативні, правові та етичні питання, що випливають з МГП і права прав людини.

Що стосується оперативних питань, то вони, передусім, пов'язані з корисністю ШІ на полі бою. Роботизовані системи часто розглядаються як так звані мультиплікатори сили: в той час як діапазон і тривалість військових операцій можуть бути збільшені, потреба у відправленні великої кількості солдатів на фронт зменшується, що знижує витрати, а також зменшує ризик втрат і страждань. Шоу навіть стверджує, що ведення бойових дій буде чистішим. Нерідко можна натрапити на такі гасла, як «роботи не ґвалтують», і що вони ідеально підходять для 4D-місій, які складаються з завдань, що є надто одноманітними (нудними), виконуються у забруднених умовах (брудними), складними і небезпечними для людини. Тим не менш, навіть прихильники розвитку БПЛА розуміють, що автономія передбачає певні компроміси, зокрема, щодо режимів контролю та підзвітності людини, регулювання яких вимагає виваженості та врахування нижчевикладених принципів МГП.

Принцип розрізнення вимагає, щоб атаки були спрямовані лише на військові об'єкти (людські і нелюдські), таким чином класифікуючи осіб і об'єкти як захищені від нападу чи ні. Принцип пропорційності вимагає визначення прямої військової переваги, отриманої від нападу, і передбачуваної шкоди як основи для прийняття рішення про те, чи варто здійснювати напад. Принцип обережності накладає на сторони, що воюють, зобов'язання проявляти постійну обережність і вживати всіх можливих заходів для мінімізації втрат серед цивільного населення. Крім того, низка принципів зобов'язує сторони, що воюють, надавати допомогу пораненим, хворим і тим, хто вижив, а також належним чином поводитися з військовополоненими або захищати об'єкти з особливим статусом, такі як культурні цінності, медичні установи або культові місця. Вищезазначені принципи покликані сприяти досягненню основної мети МГП – зменшенню втрат і страждань, спричинених війною. Ці норми є джерелом принципів, які, на відміну від правил, не діють на основі нульової суми, а тому мають відкриту структуру і вимагають людського судження та інтерпретації. Ця дія є дуже складним процесом для людей, які воюють, особливо враховуючи нинішній стан розвитку технології ШІ.

У правовому контексті, окрім технічної можливості дотримання принципів МГП, найважливішим питанням є присвоєння індивідуальної відповідальності за дії, вчинені з застосуванням ЛАСО. Так званий «розрив у підзвітності» пов'язаний з проблемами забезпечення пояснюваності процесів, що відбуваються в ЛАСО, специфічним і розподіленим процесом створення алгоритмів і нейронних мереж, а також з питанням демонстрації mens rea, тобто психічного стану, що вказує на умисел (робота або його

творця). Кінцевою метою є перетворення процесів «чорних скриньок» на «білі скриньки», щоб можна було зрозуміти, на якому етапі сталася «помилка», що призвела до порушення, і яка людина може нести за це певну відповідальність. Відповідальність держави, яка використовує закони, не є проблематичною щодо цього, оскільки ґрунтується на принципі об'єктивності.

Питання відповідальності є частиною проблематики дегуманізації війни і тісно пов'язане з концепцією, яка досі залишалася поза увагою МГП. Йдеться про контроль людини над процесами прийняття рішень і, більш конкретно, про концепцію змістовного контролю з боку людини. У майбутньому такий контроль може стати правовою нормою, але його аксіологічне підґрунтя лежить в етиці, а точніше – у веліннях суспільної свідомості. Серед іншого, доповідь «Втрачаючи людяність» висвітлила моральну проблематику передачі права прийняття рішень, що стосуються життя і смерті, від людей до нелюдей. Це стало поштовхом для аналізу того, як за попередніх методів і засобів ведення війни люди здійснювали контроль над цим процесом і як це має виглядати з появою ШІ. Це найбільш обговорюване етичне питання в контексті ЛАСО, хоча і не єдине.

Останнім часом концепція відповідального ШІ (Responsible artificial intelligence, RAI) розробляється у контексті військових застосувань такими країнами, як США, Великобританія та Франція. RAI базується на принципах, згідно з якими: ШІ має розроблятися відповідно до національного і міжнародного права (законність); відповідальність людини має бути чітко визначена, а використання ШІ має здійснюватися з увагою і обережністю (відповідальність і підзвітність); застосування ШІ має підпорядковуватися прозорим і зрозумілим процедурам, перевіркам і методологіям (пояснюваність і простежуваність); випадки застосування ШІ мають бути чітко визначені, а безпека і стійкість мають бути забезпечені протягом усього життєвого циклу цих можливостей (надійність); має бути забезпечена адекватна людино-машинна взаємодія і застосовані такі заходи безпеки, як вимкнення або деактивація у разі ненавмисної поведінки (керованість); а також мають бути вжиті проактивні заходи для зменшення упередженості. Загалом, наведені вище етичні принципи можна вважати спільними як для військових, так і для цивільних застосувань ШІ, оскільки, за винятком питання летального застосування, виклики дуже схожі.

Етичні питання також пов'язані з психологічними аспектами, зокрема з тим, як солдати будуть взаємодіяти з роботами. І хоча МГП вкрай мало говорить про психологічну шкоду, спричинену війною (за винятком використання терору як зброї проти цивільного населення), психологічні питання дуже важливі для згуртованості підрозділів, морального духу солдатів і оперативних спроможностей. Як наслідок, вони можуть мати вирішальне значення для ведення бойових дій. Дебати оберталися навколо питань, пов'язаних з «нутрощами» робота, тобто ШІ, і середовищем, в якому він працює, тобто сучасним полем бою. Ми вважаємо, що дебатам бракує

аналізу того, як має виглядати сам робот.

На початковому етапі дискусій, хоча образ Робокопа або Атласа був одним з найперших, що спадали на думку під час спроби візуалізувати ЛАСО, було лише побіжно згадано про оману Android і ризик антропоморфізації роботів, які прийдуть на зміну солдатам. Через відсутність конкретних моделей ЛАСО для аналізу (досі немає чітких позицій щодо того, чи існують такі роботи), дискусії щодо обмежень відбувалися на теоретичному та загальному рівнях. Тому часто підкреслювалося, що не потрібно зловживати такими гуманізувальними дієсловами, «вирішувати», «думати», «бачити» або «відчувати», описуючи функціонування ЛАСО. І хоча на лінгвістичному рівні склався певний консенсус, і ЛАСО чітко зображуються як засоби ведення війни, бойові системи, одиниці техніки, це не змінює того факту, що на реальному полі бою ці роботи можуть бути сприйняті як люди. Цей ризик і подальші загрози можуть матеріалізуватися у сценарії, коли військові роботи набувають форми і поведінки людей.

З правової точки зору людиноподібних роботів потрібно однозначно класифікувати як військову техніку, а отже, як військові об'єкти за своєю природою. Не має бути жодних сумнівів, що такі роботи не мають статусу учасника бойових дій і, відповідно, статусу військовополонених – вони мають розглядатися як об'єкти в будь-якому випадку. Їхнє впровадження в армійську техніку важко виправдати з оперативної та правової точок зору. З огляду на те, що людиноподібний робот є частиною військової техніки, він має бути відповідним чином позначений значками і символікою сторони, що воює. Він, безумовно, не може бути схожим на цивільних осіб, поранених, військовополонених, священнослужителів або медичний персонал, оскільки це було б актом віроломства і, отже, воєнним злочином. Теоретично, як військову хитрість, використання роботів для узурпації комбатантів не є незаконним. Однак непрямі наслідки такої дії можуть мати негативний вплив на дотримання сторонами конфлікту принципу розрізнення, пропорційності та запобіжних заходів під час нападу.

Людиноподібні роботи можуть внести додаткову плутанину на сучасне поле бою, яке і без них є досить складним і відповідальним для солдатів-людей. Розробці і використанню такого засобу ведення війни має передувати юридична експертиза, яка розглядає правові, етичні, політичні та медичні наслідки застосування таких роботів. Це має поєднуватися з оцінюванням ризиків і впровадженням заходів щодо їх зменшення. Однак, з огляду на мету МГП – мінімізувати випадкові людські жертви, поранення цивільних осіб і пошкодження цивільних об'єктів, захистити таку роботизовану конструкцію з правової точки зору неможливо.

Тепер ми заглибимося у ці дві загрози, починаючи з епістемологічної. Як уже згадувалося, люди мають обмежений апарат, з якого ми отримуємо інформацію про зовнішній світ. Ми не можемо, наприклад, бути впевненими у внутрішніх станах інших людей і не маємо до них прямого доступу. У філософії популярним є уявний експеримент щодо зомбі та різних супу-

тніх питань. Одне з них – як ми маємо ставитися до сутностей, які виглядають і поводяться як люди, але не мають внутрішніх станів людини. Данахер звертається до цього прикладу в контексті роботів і задається питанням, як ми маємо ставитися до роботів, які виглядають як істоти, що мають моральний статус, як люди і тварини. Він дійшов висновку, що через наші епістемологічні обмеження розумно ставитися до них як до суб'єктів, що мають такий статус. Свою позицію він називає етичним біхевіоризмом, оскільки зосереджується на спостережуваних рисах (зовнішній вигляд і поведінка) роботів.

На практиці, якщо існує робот, який виглядає і поводиться як людина, то його буде важко відрізнити від людини. У військовому контексті це може становити загрозу для життя людей. Перша проблема полягає в тому, що якщо будуть прийняті на озброєння людиноподібні роботи, то всі об'єкти, які присутні на полі бою, потенційно є військовими роботами. Навіть якщо нападник хоче знищити роботів, а не людей, може бути проблематично розрізнити ці дві категорії. У військовому середовищі існує ще одна проблема, яка ставить людей у невигідне становище порівняно, наприклад, з р*о*ботами, – це час. Це питання пов'язане з етичними рамками, орієнтованими на цінність людського життя.

Протистояння з роботами може бути смертельно небезпечним для людини-солдата, тому може бути критично важливим якнайшвидше прийняти рішення про його ліквідацію. Загроза життю нападника – чим менше часу на прийняття виважених рішень, тим більша ймовірність випадково завдати шкоди людині. Навіть якщо відмінності можна відстежити після оцінювання характеру об'єкта, то у військовому контексті час працює проти безпеки людей. Солдати можуть бути більш схильні до знищення техніки, ніж до вбивства людини (навіть якщо обидві дії дозволені законом), але прийняттю виважених рішень може перешкоджати загроза, пов'язана з можливістю наразитися на небезпеку в безпосередньому зіткненні з роботом. Саме тому проблематичним є також створення робота, який лише зовні схожий на людину, тобто приблизно людського зросту і ходить на двох ногах. Такі роботи здалеку можуть виглядати як люди, і знову ж таки, якщо безпосереднє зіткнення з роботом загрожує людині, то може бути розумним ліквідувати його на відстані, а це також збільшує шанси помилитися.

Існування людиноподібних роботів у військових зонах також створює ризик надання можливості уникнути відповідальності у разі вбивства людини. Це пов'язано з питаннями розрізнення легітимних і нелегітимних цілей. Якщо коротко, то особа, яка цілилася в робота, який виявився людиною (цивільною особою), може не нести відповідальності за злочин проти людини, яка є нелегітимною ціллю. Це пов'язано з інститутом помилки, яка може виправдати злочинця. Він застосовується в ситуації, коли, наприклад, відбувається полювання, і особа стріляє в об'єкт, який знаходиться в кущах, на ногах, розміром з кабана, і видає звук кабана. Особа, яка стріляє,

має обґрунтовані підстави вважати, що це не кабан, а людина. Особа не нестиме відповідальності за цей вчинок, навіть якщо внаслідок пострілу людина загине. Це здається виправданим, якщо людина дійсно вважає, що на неї нападає робот. Але є також проблема з використанням цього виправдання у випадках, коли людина навмисно стріляє в особу, яка перебуває під захистом. Особа, яка перебуває під слідством, може використати таке виправдання, щоб уникнути відповідальності за заподіяння смерті цивільній людині. Особа може стверджувати, що мала намір стріляти в робота або комбатанта, а не в особу, яка перебуває під захистом. Тому така фактична помилка може нівелювати психічний елемент, потрібний для злочину (згідно зі статтею 32(1) Римського статуту Міжнародного кримінального суду). Чим більш людиноподібними є роботи, тим більш правдоподібним є уникнення відповідальності в такий спосіб. Аргументація може полягати в тому, що людина, в яку стріляли здалеку, була цивільною особою, але нападник прийняв таке рішення, тому що на відстані подумав, що це військова ціль (або військовий робот), а оскільки робот може бути більш небезпечним, перебуваючи ближче, рішення було прийнято у стані невизначеності або помилки, яка була виправдана в очах того, хто приймав рішення (оскільки це було сприйнято як загроза для його життя). Ми не стверджуємо, що це може траплятися часто, але вказуємо на можливість додаткових аргументів, які можуть з’явитися при розгортанні роботів, що нагадують людей.

Підсумовуючи епістемологічну загрозу, якщо військові роботи виглядають як люди, це збільшує ризик того, що їх можуть прийняти за людей. Як люди, так і людиноподібні роботи можуть виглядати як військові роботи, що створює додаткові ризики для людей. Загроза стосується не лише роботів, яких важко відрізнити від людей, але й усіх роботів, які більшою чи меншою мірою мають людську подобу, оскільки рішення про напад на них може прийматися на певній відстані від об’єкта і у поспіху, обидва аспекти підвищують ймовірність помилки. Людиноподібність також створює можливість уникнути відповідальності, заспокоюючи, що ціллю був робот, а не людина.

Загроза з боку користувача не така проста, як епістемологічна загроза, яка базується лише на зовнішньому вигляді робота. Загроза користувача стосується можливої прихильності до людиноподібних роботів і пов’язана з антропоморфізацією, людською схильністю бачити людські якості в нелюдських об’єктах і подіях.

З’являється все більше літератури про взаємодію людини і робота, яка показує, що люди ставляться до роботів не як до об’єктів, а як до чогось більшого. Наприклад, Сальвіні та ін. показують, що люди ставляться до нападу на роботів не як до вандалізму, а скоріше як до знущання [36]. Люди співпереживають «стражданням» роботів, вони відчувають емпатію до них, якщо на них нападають [37].

У деяких країнах існує обов’язок рятувати, який іноді називають за-

коном самаритянина [38]. Якщо робот не знищується з метою порятунку людей, які перебувають у небезпеці, це може бути класифіковано як злочин [39].

У військовому контексті надмірна прив'язаність до роботів також може бути проблематичною: люди мають мати пріоритет у порятунку, а почуття до роботів можуть бути тягарем, який заважає людям діяти належним чином. Це пов'язано з етичними рамками, що стосуються цінності людського життя. Тут також є проблема часу, про яку ми згадували раніше, рішення може бути прийняте швидко, а людиноподібність роботів є додатковою проблемою, яка може уповільнити прийняття рішення. Потрібно додати, що прихильність до роботів можлива не лише тоді, коли вони схожі на людей, але й у випадку з іншими роботами. Навіть у військовому контексті відомі історії ставлення до роботів як до членів команди, наприклад, історії про похорони роботів, які влаштовували солдати-однополчани. Дарлінг зазначає, що вирішальним аспектом, який може виникнути в таких реакціях на роботів, є рух: якщо робот рухається, то він може бути сприйнятий як живий об'єкт, що може викликати додаткові реакції [40]. Але можна сказати, що чим більше робот схожий на людину, тим більше почуттів і людських якостей ми можемо приписати роботам.

У цьому випадку проблема не в тому, що ми можемо сплутати роботів з людьми. Ми знаємо, що маємо справу з роботами, але їхні особливості викликають певні реакції, які є небезпечними для інших людей.

Ця загроза також відрізняється у випадку груп потенційних жертв. Людиноподібність загрожує цивільним особам і комбатантам. Солдати можуть мати проблеми з тим, щоб залишити робота у небезпечних ситуаціях через свою прив'язаність до нього. Нерішучість залишити робота або пожертвувати ним заради порятунку інших може коштувати життів реальних людей. Це загроза, яку має враховувати і сторона, що використовує роботів. Їхні власні солдати можуть опинитися під загрозою через надмірну прихильність до них інших солдатів-роботів, що може завадити громадській підтримці розгортання роботів у країні.

Відмова від людиноподібності роботів може майже повністю усунути епістемологічну загрозу і обмежити загрозу користувача. Обмежити, а не вирішити, тому що солдати можуть розвинути прив'язаність і до нелюдей, таких як роботи.

Насамкінець, важливо підкреслити, що дискусія про використання роботів у військовому контексті триває і буде тривати ще досить довго. І перш, ніж розгортати роботів на полі бою, ще потрібно прийняти багато важливих рішень.

1.4 Контрольні питання

1. Поясніть, що таке робототехніка.
2. Обгрунтйуте актуальність застосування роботів на сучасному етапі розвитку людства.

3. Наведіть основні складові сучасної робототехніки.

4. Наведіть основні типи роботів на базі їхніх рухових можливостей.

5. Поясніть, чим промислові роботи відрізняються від сервісних. Наведіть приклади роботів до кожної категорії.

6. Обгрунтуйте важливість застосування сенсорів у робототехніці та наведіть приклади найпоширеніших сенсорів.

7. Поясніть, яке значення мають актуатори в робототехнічних системах і чим вони відрізняються від сенсорів.

8. Проаналізуйте концепцію ступенів свободи (DOF) у робототехніці та її значення для проєктування роботів.

9. Проаналізуйте, які переваги та недоліки мають колісні та ножні роботи, а також роботи, що літають у різних сферах застосування.

10. Поясніть, як робототехніки підходять до проєктування систем взаємодії між людиною та роботом, і які ключові міркування при цьому враховуються.

11. Проаналізуйте роль штучного інтелекту та машинного навчання у робототехніці.

12. Поясніть, які етичні міркування пов’язані з розробкою та розгортанням автономних роботів.

13. Наведіть приклади використання роботів у готельному бізнесі та туризмі.

14. Проаналізуйте основні аспекти використання роботів у військовій справі.

15. Поясніть, як робототехніки забезпечують безпеку роботів у різних середовищах, зокрема на спільних робочих місцях людини і робота.

2 МОБІЛЬНА АВТОНОМНА ПЛАТФОРМА MBOT

2.1 Підключення робота mBot до персонального комп'ютера

До зібраного робота mBot потрібно під'єднати блок живлення з чотирма батарейками AA, або літій-іонний акумулятор. Далі потрібно під'єднати mBot до персонального комп'ютера (ПК) кабелем USB і перевести перемикач живлення (на правій стороні робота) у положення ON. Якщо робот зібрано правильно і блок живлення видає достатню напругу, то на першому засвітяться світлодіоди, а на ПК з'явиться повідомлення про підключення нового пристрою та встановлення драйверів.

Якщо Ви працюєте на ПК під керуванням ОС Windows, то в «Диспетчері пристроїв» у розділі «Порти (COM і LPT)» з'явиться пристрій «USB-SERIAL CH340, у нашому випадку (COM7)» (рис. 2.1).

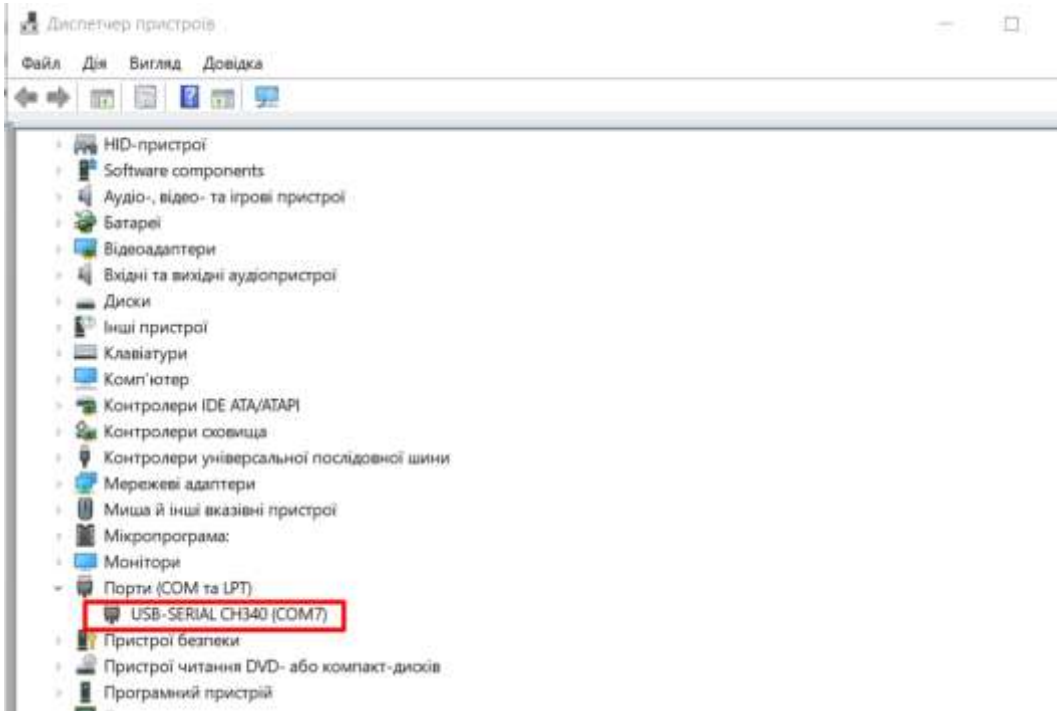


Рисунок 2.1 – Вигляд диспетчера пристроїв

Для того, щоб запустити додаток, потрібно натиснути кнопку «Пуск», розташовану в дальньому лівому кутку панелі завдань. Далі потрібно натиснути правою кнопкою миші на піктограмці «Комп'ютер», і лівою кнопкою миші «Властивості». У меню з'явиться «Диспетчер пристроїв», на якому потрібно натиснути. Нас цікавить номер COM-порту, який написаний у дужках (скоріш за все, на Вашому ПК назва такого пристрою буде відрізнятися номером у дужках). Запам'ятайте цей номер COM-порту, адже його у подальшому потрібно буде використовувати для програмування роботи mBot.

2.1.1 Налаштування для програмування у середовищі mBlock

Для програмування mBot спочатку потрібно завантажити середовище програмування mBlock. Для цього на сайті http://www.mblock.cc/ download/ [41] потрібно вибрати версію mBlock, відповідну ОС на Вашому ПК. Наприклад, для ОС Windows потрібно виконати такі кроки:

1. Відкрити у браузері сайт http://www.mblock.cc/download/,
2. Вибрати посилання «mblock 3 (Stop updating)» (рис. 2.2),
3. Завантажити версію для Вашої операційної системи (ОС), наприклад, «Windows 7 і вище V3.4.12».

Рисунок 2.2 – Вигляд вікна для завантаження середовища mBlock

Тепер потрібно встановити середовище mBlock на вашому ПК. Якщо у Вас недостатньо системних прав для встановлення програми, встановіть її від імені адміністратора.

Після встановлення середовища mBlock його потрібно правильно налаштувати. З цією метою:

1. Запускаємо mBlock,
2. Спочатку у верхньому меню потрібно вибрати пункт «Board» (плата пристрою) і у відкритий список поставити галочку навпроти «mBot» (рис. 2.3),
3. Потім для з'єднання з роботом потрібно налаштувати COM-порт, який був присвоєний Вашому роботу в «Диспетчері пристроїв». Для цього у верхньому меню потрібно вибрати Connect → Serial Port → COM3 (тут COM3 вказано для нашого приладу, Вам же потрібно вказати COM-порт саме Вашого робота) (рис. 2.4).

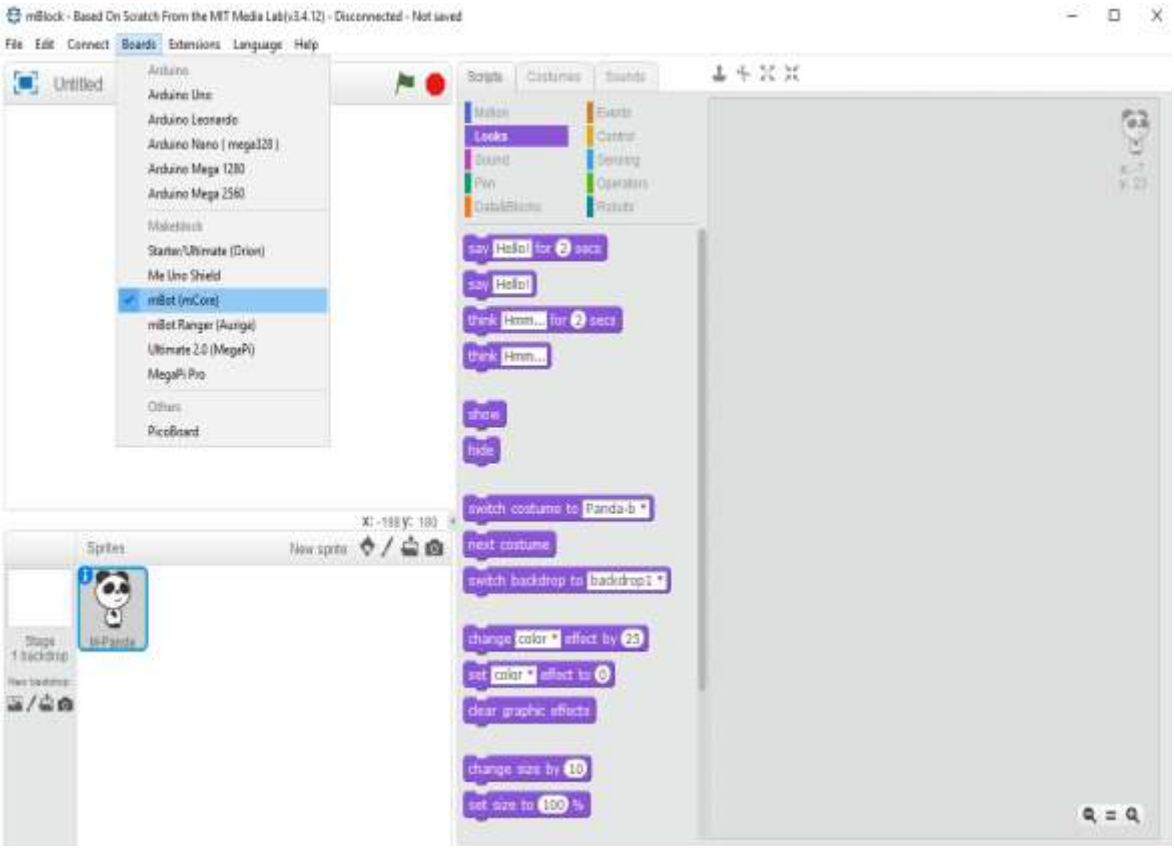


Рисунок 2.3 – Вибір плати робота mBot

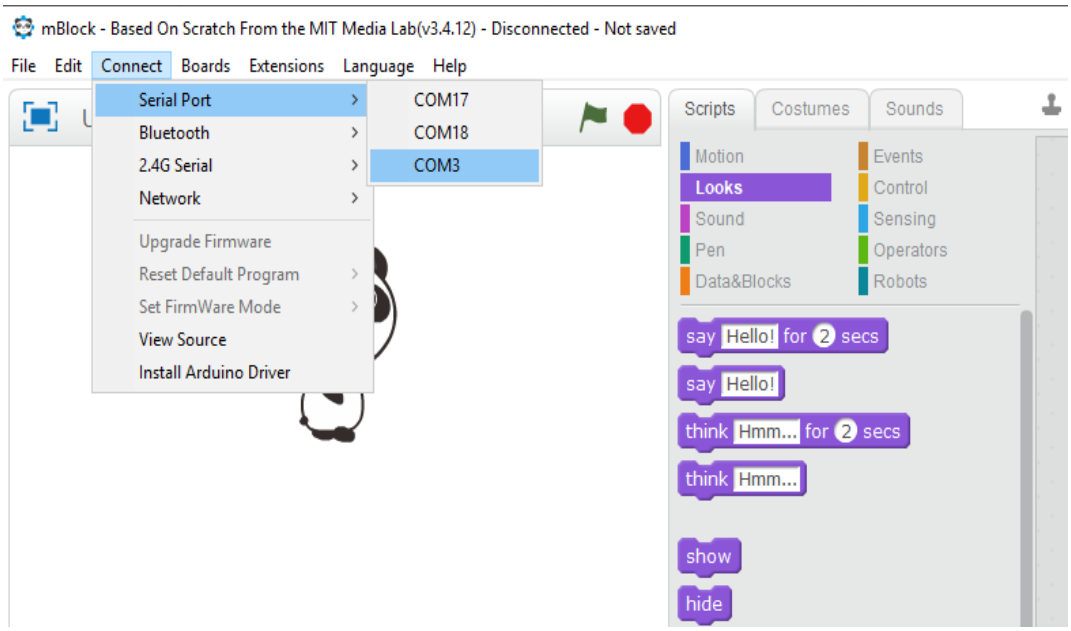


Рисунок 2.4 – Вибір COM-порта під’єднання робота

Якщо у «Диспетчері пристроїв» роботу не було присвоєно номер COM-порта, то спробуйте встановити відповідний драйвер. Для цього у програмі mBlock у верхньому меню потрібно вибрати пункт Connect → Install Arduino Driver (рис. 2.5).

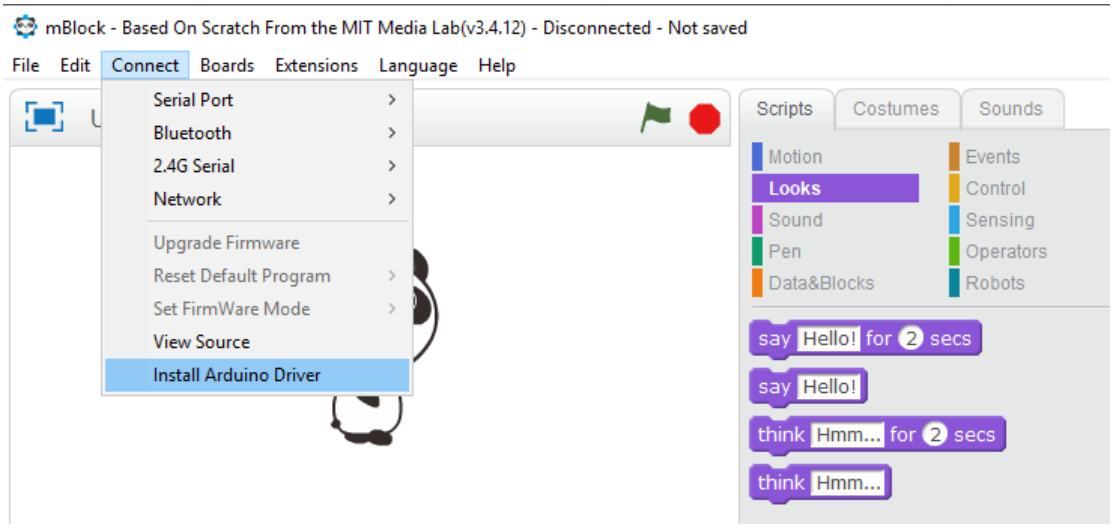


Рисунок 2.5 – Встановлення драйвера робота

Або потрібно завантажити драйвери, пройшовши за посиланням https://raw.githubusercontent.com/Makeblock-official/Makeblock-USB-Driver/master/Makeblock_Driver_Installer.zip і встановити їх у ручному режимі [42].

Середовище mBlock відразу після запуску дозволяє писати програму мовою Scratch.

Для того, щоб мати можливість писати власні програми для робота та завантажувати їх у пам’ять мікроконтролера робота, потрібно перемкнути середовище mBlock у режим «Arduino mode», для цього у верхньому меню потрібно вибрати пункт Edit → Arduino mode (рис. 2.6).

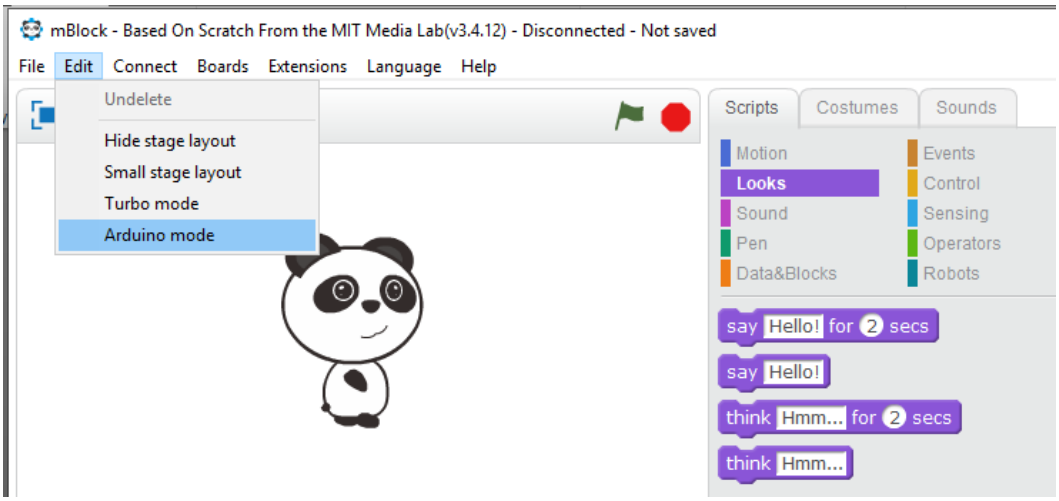


Рисунок 2.6 – Перехід у режим програмування

Після цього вигляд mBlock зміниться, і можна буде створювати програми для mBot і завантажувати їх у пам’ять робота mBot, натиснувши мишкою кнопку «Upload to Arduino» (рис. 2.7). *Важливо*: перед завантаженням програми у пам’ять робота обов’язково переконайтеся, що він

під’єднаний: у верхньому рядку mBlock має бути напис «Serial Port Connected», в іншому випадку перевірте з’єднання mBot з ПК.

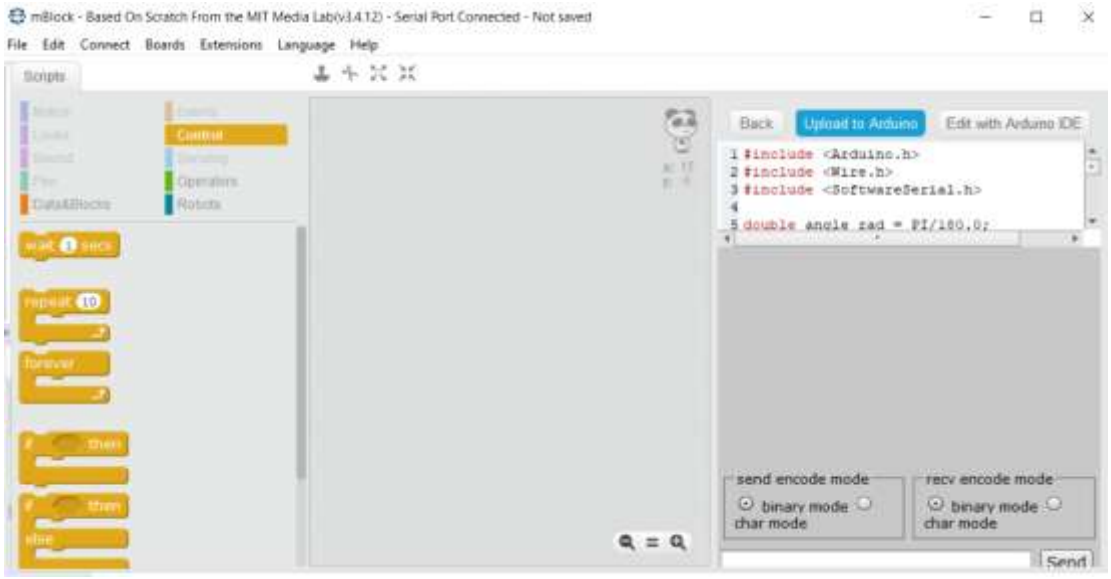


Рисунок 2.7 – Вигляд вікна створення та завантаження програм у робот

Якщо Вам зручно працювати з меню та блоками іншою мовою, то виберіть у верхньому меню пункт «Language» і поставте галочку навпроти відповідної мови (рис. 2.8).

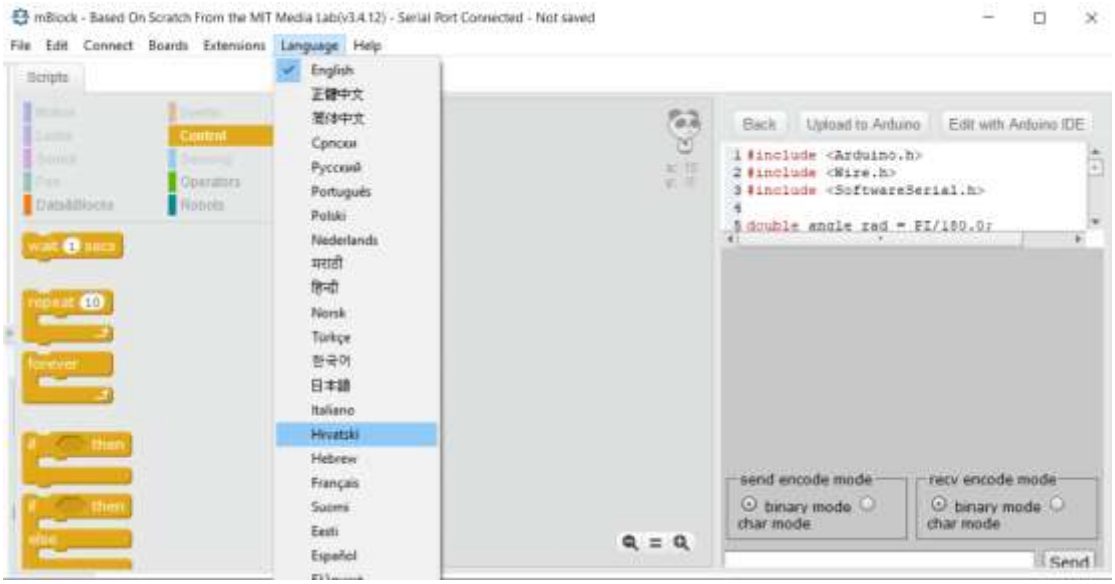


Рисунок 2.8 – Вікно вибору мови

2.1.2 Налаштування для програмування у середовищі Arduino IDE

Робот mBot можна програмувати мовою програмування C/C++ за допомогою середовища Arduino IDE. Для цього виконайте такі дії.

1. Завантажте за посиланням http://arduino.cc/download версію Arduino IDE для ОС, встановленої на Вашому ПК [43].

2. Встановіть Arduino IDE на Вашому ПК або розпакуйте з завантаженого архівного файлу.

3. Запустіть на виконання Arduino IDE (arduino.exe).

4. Плата робота mBot під назвою mCore сумісна з Arduino UNO, тому у верхньому меню в пункті «Інструменти (Tools)» потрібно вибрати Плата → Arduino Uno (рис. 2.9).

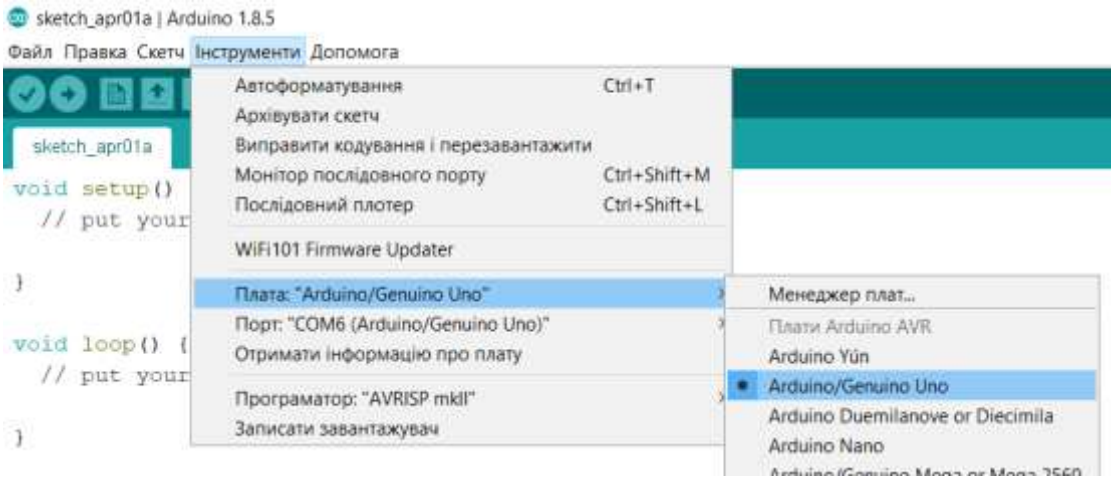


Рисунок 2.9 – Перехід до програмування робота у середовищі Arduino IDE

5. Для підключення робота mBot до ПК через USB-кабель у верхньому меню в пункті «Інструменти (Tools)» потрібно вибрати Порт → COM3 (тут COM3 наведено для нашого приладу (рис. 2.10), Вам потрібно вказати COM-порт Вашого робота).

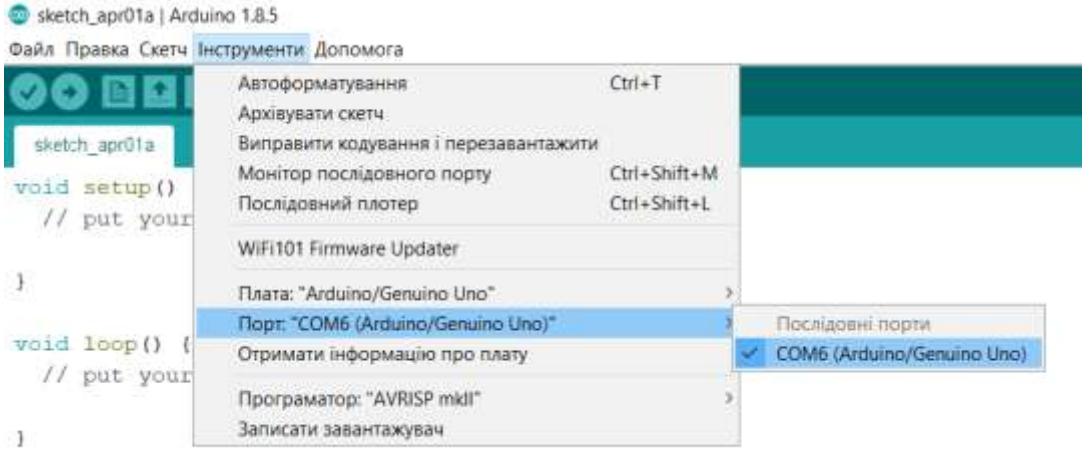


Рисунок 2.10 – Підключення робота до ПК у середовищі Arduino IDE

6. Для завантаження програми у пам’ять робота mBot можна клікнути мишкою за допомогою стрілки «Завантажити» на верхній панелі інструментів (рис. 2.11). Або вибрати у верхньому меню пункт Скетч → Завантажити, чи на клавіатурі одночасно натиснути комбінацію клавіш Ctrl+U (рис. 2.12).

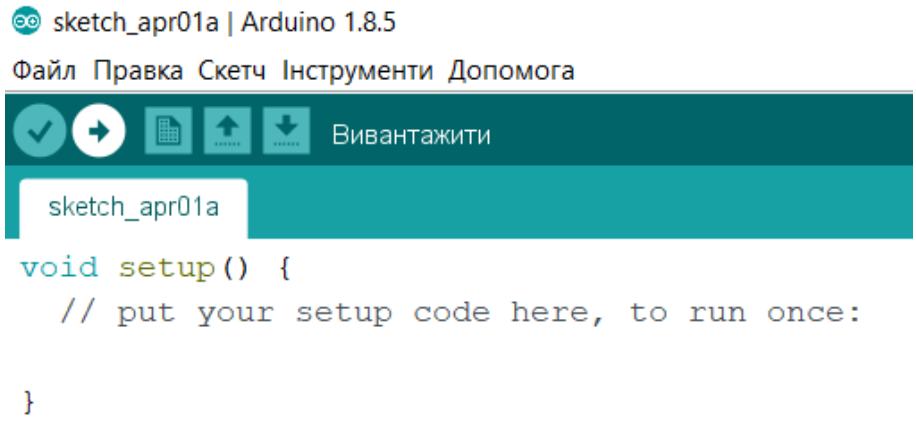


Рисунок 2.11 – Завантаження програми у mBot через кнопку стрілки

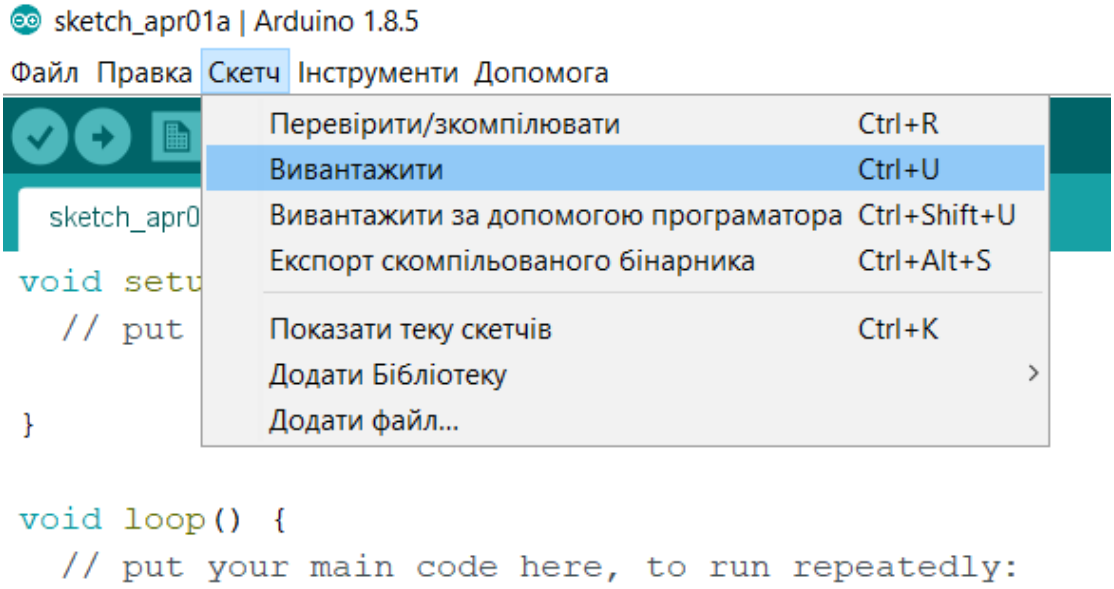


Рисунок 2.12 – Завантаження програми у mBot через комбінацію Ctrl+U

Важливо, перед завантаженням програми у пам’ять робота переконатися, що останній підключений: у нижньому правому кутку робочого вікна Arduino IDE виводиться інформація про підключення плат «Arduino Uno» з COM-портом (рис. 2.13).

7. Якщо у «Диспетчері пристроїв» роботу не було присвоєно COM-порт, то потрібно завантажити драйвер за посиланням https://raw.githubusercontent.com/Makeblock-official/Makeblock-USB-Driver/master/Makeblock_Driver_Installer.zip і встановити його вручну.

8. Для програмування робота mBot у середовищі Arduino IDE потрібно додати бібліотеку MakeBlock. Скачати бібліотеку можна за посиланням https://github.com/Makeblock-official/Makeblock-Libraries. Потім її потрібно розпакувати та скопіювати у папку makeblock у підпапку libraries, яка знаходиться у папці arduino (наприклад, у C:\Program Files\Arduino). Або, не розпаковуючи, вибрати у верхньому меню пункт Скетч → Під’єднати бібліотеку → Додати .ZIP бібліотеку… (рис. 2.14)

Рисунок 2.13 – Визначення порта підключення плати Arduino Uno

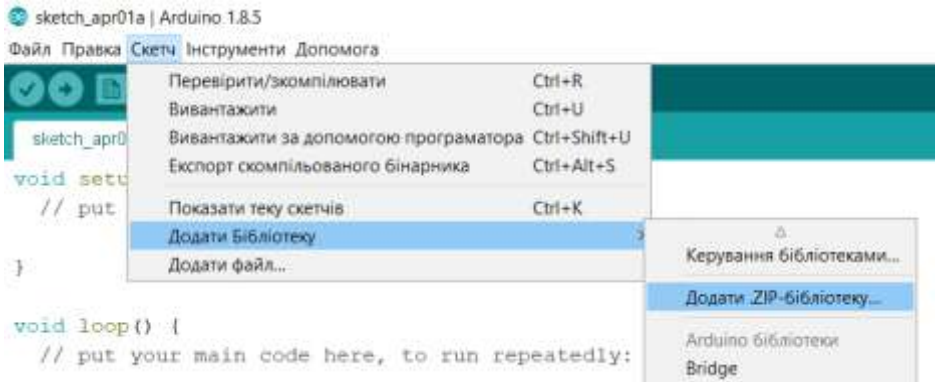


Рисунок 2.14 – Додавання .ZIP бібліотеки в Arduino IDE

2.1.3 Основні характеристики плати mCore робота mBot

Загальний вигляд плати mCore наведено на рис. 2.15, а її основні характеристики у таблиці 2.1 [44].

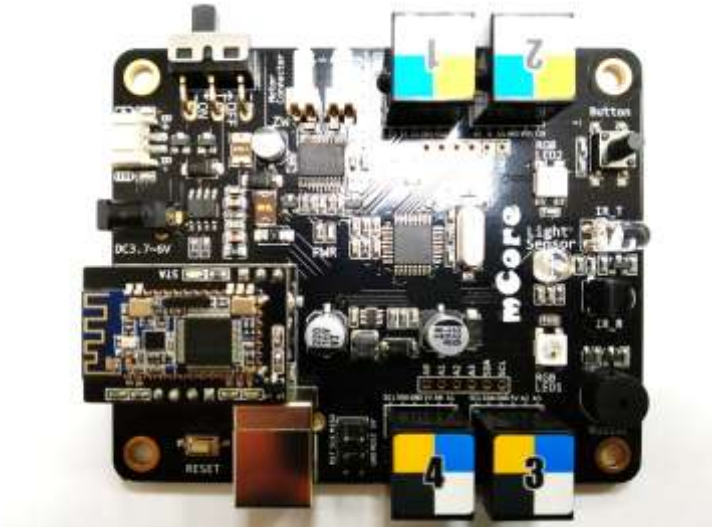


Рисунок 2.15 – Загальний вигляд плати mCore

Наведемо деякі пояснення.

Пристрої введення – пристрої, за допомогою яких дані (інформація) виводяться в mCore.

Пристрої виведення – пристрої, через які дані (інформація) від mCore передаються у зовнішній світ.

Таблиця 2.1 – Загальні характеристики плати mCore

Середовище програмування	Arduino IDE, mBlock (Mac OS, Windows, Linux*), mBlock APP (Android, IOS)
Пристрої введення	Сенсор освітленості, тактова кнопка, інфрачервоний приймач.
Пристрої виведення	Бузер, RGB-світлодіод, інфрачервоний передавач, 2 порти для двигунів
Мікроконтролер	ATmega328, Arduino UNO-сумісний
Напруга живлення	Від 3,7 Вольт до 6 Вольт
Безпровідне з'єднання	Bluetooth
Зв'язок з комп'ютером	USB
Роз'єми для електроживлення	Холдер для 4-х батарейок AA, роз'єм для Li-ion акумулятора
Перезапуск	Кнопка Reset для перезавантаження програми в mCore
Перемикач	Вмикання живлення – (ON – OFF)
Роз'єм для підключення двигунів	Motor Connector M1 – лівий двигун, M2 – правий двигун

Якщо розглянути електричну схему mCore, то можна відзначити, що:

- Зумер (buzzer, бузер, звуковипромінювач) під'єднаний до контакту (pin) 8.
- Кнопка підключена до контакту A7.
- Сенсор освітленості під'єднаний до контакту A6.
- Світлодіод (LED), вмонтований на плату, під'єднаний до контакту 13.
- RGB-світлодіоди (світлодіоди WS2812) під'єднані до контакту 13.
- Двигун1 під'єднаний до контакту 5 (широтно-імпульсний модулятор (ШІМ) (pulse-width modulation, PWM) визначає швидкість обертання) та до контакту 4 (напрям обертання (Direction, DIR).
- Двигун 2 під'єднаний до контакту 6 (ШІМ визначає швидкість обертання) та до контакту 7 (напрям обертання).
- Інфрачервоний (ІЧ) приймач під'єднаний до контакту 2.
- ІЧ передавач під'єднаний до контакту 3.
- У бібліотеці makeblock наявні «Порти», логічно суміщені з двох pin-портів:
  - Port_1 {11, 12} = RJ25 port 1.
  - Port_2 {9, 10} = RJ25 port 2.
  - Port_3 {A2, A3} = RJ25 port 3.
  - Port_4 {A0, A1} = RJ25 port 4.
  - Port_5 {NC, NC} (Not Connected).
  - Port_6 {8, A6} = зумер, датчик світла (Light sensor).

- Port_7 {A7, 13} = зумер, два світлодіоди WS2812.
- Port_8 {8, A6} = зумер, датчик світла.
- Port_9 {6, 7} = ШІМ (PWM) двигуна 2, напрям (DIR) двигуна 2.
- Port_10 {5, 4} = ШІМ двигуна 1, напрям двигуна 1.

Для підключення зовнішніх датчиків або модулів на платі mCore є чотири порти RJ25 з кольоровим маркуванням, яке означає:

- Чорний колір – один або два аналогових порти.
- Синій колір – два цифрових порти.
- Жовтий колір – один цифровий порт.
- Білий колір – порт I2C.

2.2 Застосування бібліотеки для програмування у середовищі Arduino IDE

Бібліотека – це програмний код для скетчу, що зберігається у зовнішньому файлі, який підключається до проєкту. Бібліотеки значно полегшують програмування для любителів, які бажають щось зробити своїми руками, але не глибоко розуміють мову програмування С++. Поряд з бібліотеками (в архіві) можна знайти також приклади використання команд у скетчі, щоб уникнути помилок під час програмування.

Усі бібліотеки Arduino можна розділити на стандартні (їх не потрібно встановлювати, вони відразу вбудовані у середовище Arduino IDE). Додаткові бібліотеки розробляє виробник датчиків і модулів, їх потрібно завантажувати та встановлювати. В Інтернеті розміщено сотні готових бібліотек для модулів. Крім того, бібліотеку можна написати і самостійно.

Розглянемо детальніше процес встановлення бібліотеки в Arduino IDE.

Перед використанням методів та функцій бібліотеки, її спершу потрібно завантажити та встановити на ПК. Додати бібліотеку можна через середовище Arduino IDE або вручну, розпакувавши архів із файлами у певне місце. Крім прикладів зі скетчами в архіві мають бути такі файли: example.h – заголовний файл, example.cpp – файл з кодом (сирцем), keywords.txt – файл з інформацією щодо виділення команд кольором.

Додавання бібліотек в Arduino IDE виглядає так, як наведено нижче.

Якщо Ви завантажили бібліотеку для свого проєкту, запаковану в ZIP-архів, можна зробити встановлення через середовище Arduino IDE. Для цього потрібно перейти в меню: Скетч → Підключити бібліотеку → Додати бібліотеку .ZIP і вибрати ZIP-архів на ПК (див. рис. 2.14). Файл буде автоматично розпаковано та переміщено у директорію з бібліотеками. Після встановлення бібліотека буде доступна у меню Файл → Приклади.

Для встановлення бібліотеки вручну потрібно розпакувати папку з файлами бібліотеки та перемістити її у розділ ПК Мої документи → Arduino → libraries. Перед тим, як підключити бібліотеку в скетчі, потрібно закрити всі вікна Arduino IDE. Після запуску середовища програмування

функції та команди зі встановленої бібліотеки стануть доступними у скетчі.

Перед використанням команд і функцій у програмі потрібно підключити у скетчі потрібну бібліотеку. Для цього використовується директива #include, після якої у лапках " " або дужках < > вказується ім'я бібліотеки з розширенням. Наприклад, щоб у скетчі підключити бібліотеку Makeblock для плати mCore, потрібно додати рядок директиви #include "MeOrion.h" (рис. 2.16).

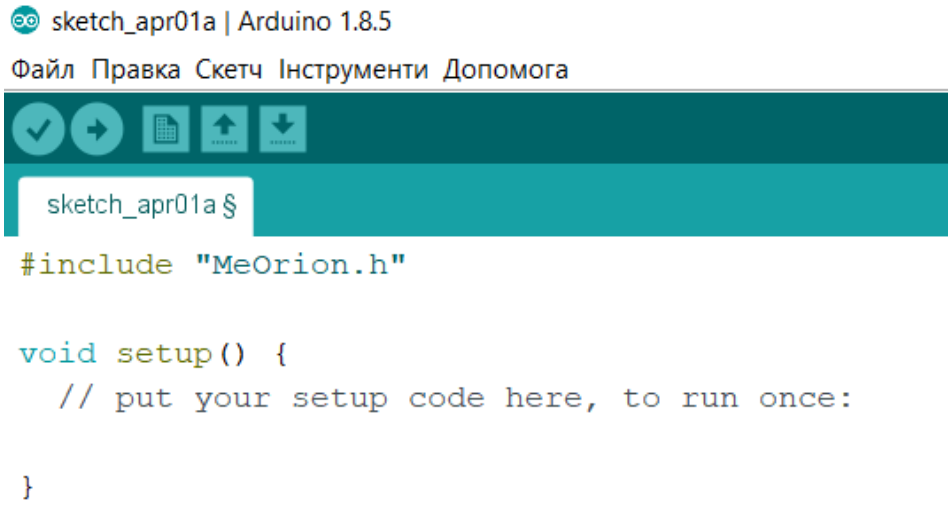


Рисунок 2.16 – Під'єднання бібліотеки Makeblock для плати mCore у середовищі Arduino IDE

Бібліотеки додають на початку скетчу до процедури void setup (). Якщо у програмі функція буде використана до того, як підключена бібліотека у скетчі, то це викликає помилку під час компіляції.

Далі розглянемо, як завантажити стандартні бібліотеки Arduino IDE.

Бібліотеки для Arduino діляться на дві групи – стандартні та користувацькі [45].

При встановленні Arduino IDE, у папці: Файли програм (x86)>Arduino>libraries (рис. 2.17) є набір стандартних бібліотек для базових функцій, комунікації плати та для підключення пристроїв: сервомоторів, крокових двигунів, LCD-дисплеїв і т. д. Стандартні бібліотеки можна завантажити на офіційному сайті www.arduino.cc.

Список стандартних бібліотек Arduino (рис. 2.18)

EEPROM – читання та запис в енергонезалежну пам'ять (завантажити eeprom.h).

Ethernet – зв'язок з Інтернет за допомогою Ethernet Shield (завантажити ethernet.h).

Firmata – для взаємодії Arduino та ПК (завантажити firmata.h).

GSM – комунікація за GSM/GRPS протоколом для GSM Shield (завантажити gsm.h).

LiquidCrystal – керування LCD дисплеєм (завантажити liquidcrystal.h).

SD – читання та запис на SD карту (завантажити sd.h).

Servo – керування серводвигуном (завантажити servo.h).

SPI – для взаємодії Arduino та периферійних пристроїв (завантажити spi.h).

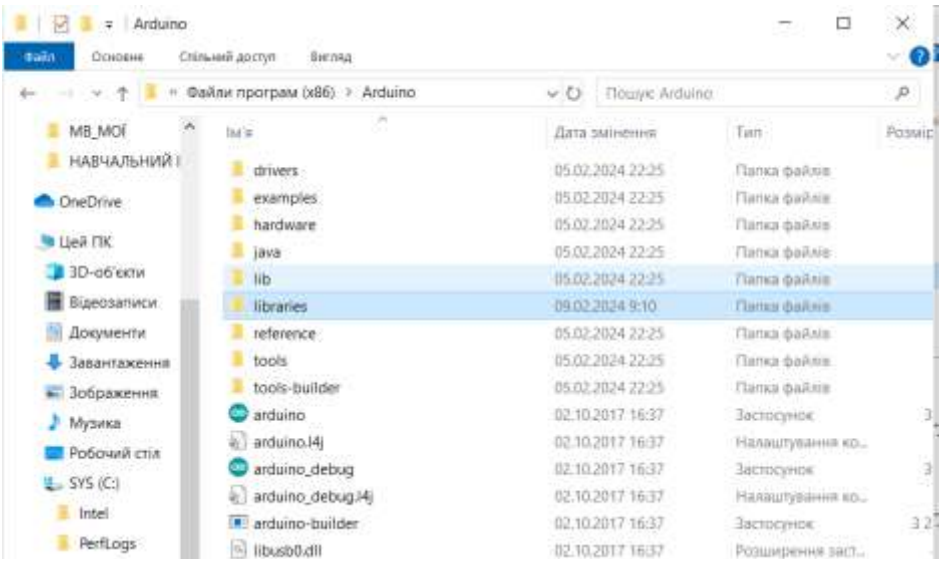


Рисунок 2.17 – Шлях до стандартних бібліотек

SoftwareSerial – комунікація за цифровим портом (завантажити softwareserial.h).

Stepper – управління кроковим двигуном (завантажити stepper.h).

TFT – виведення тексту та картинок на TFT дисплеї (завантажити ethernet.h).

WiFi – зв’язок з Інтернет за допомогою WiFi Shield (завантажити wifi.h).

Wire – комунікація за протоколом I2C (завантажити wire.h).

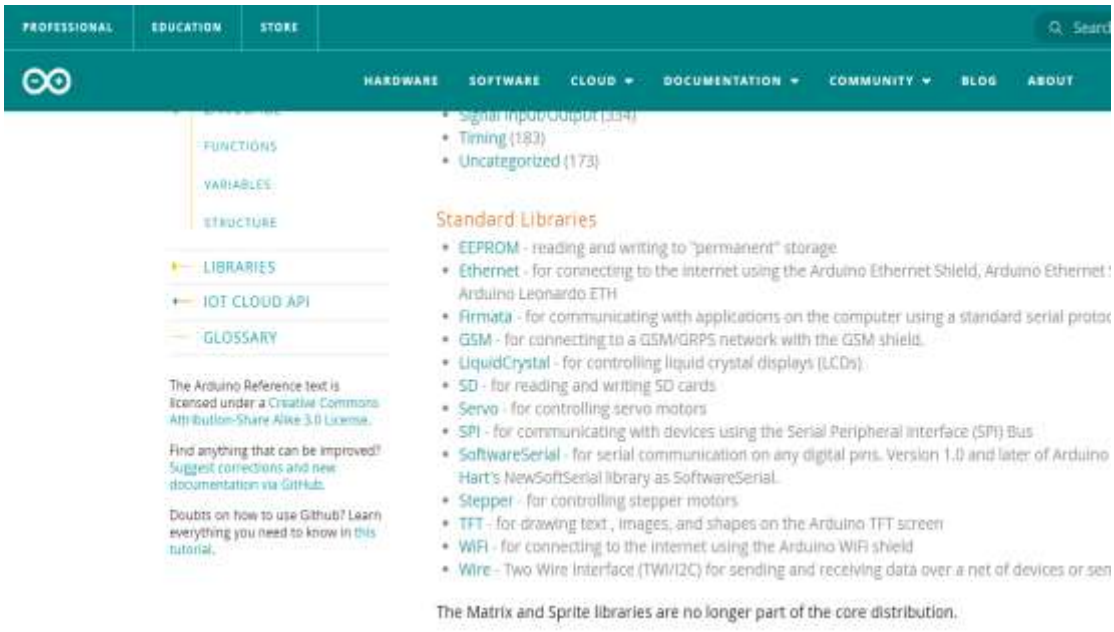


Рисунок 2.18 – Список стандартних бібліотек Arduino

Тепер покажемо, як завантажити популярні бібліотеки Arduino IDE. Користувацькі бібліотеки створюються розробниками модулів та плат розширень. Більшість популярних бібліотек Arduino можна завантажити на сайті GitHub. Це сервіс для спільної розробки IT-проєктів, де можна відстежити історію змін коду.

Наведемо приклади деяких бібліотек Arduino IDE, що можуть стати у нагоді.

TroykaCurrent – переведення аналогових значень в Ампери (завантажити TroykaCurrent.h)

RotaryEncoder – робота з модулем енкодера (завантажити RotaryEncoder.h)

Adafruit NeoPixel – робота з адресним рядком (завантажити Adafruit_NeoPixel.h)

Fast LED – робота зі світлодіодами ws2812b (завантажити FastLED.h)

TroykaMQ – робота з датчиками газу MQ (завантажити TroykaMQ.h)

MQ-2 sensor – робота з датчиком газу MQ2 (завантажити MQ2.h)

LCD1602 I2C – бібліотека для дисплея 1602 I2C (завантажити LiquidCrystal_I2C.h)

OLED I2C – бібліотека для OLED дисплея (завантажити OLED_I2C.h)

SFE_BMP180 – бібліотека для датчика тиску BMP180 (завантажити SFE_BMP180.h)

SD.h – бібліотека для роботи з SD-картою пам’яті (завантажити SD.h)

Стандартні файли Arduino IDE зберігаються у каталозі Program Files \Arduino\libraries. Якщо Вам потрібно додатково встановити бібліотеку, то архів можна розпакувати у цей каталог або у папку Мої документи Arduino libraries. Встановити бібліотеку можна також через меню програми Arduino IDE. Як це правильно зробити ми розповідали вище у цьому ж розділі посібника.

2.3 Керування світлодіодами, вбудованими у плату mCore робота mBot

Світлодіод (англ. *LED – light-emitting diode*) – напівпровідниковий пристрій, що випромінює світло під час пропускання через нього електричного струму. Колір випромінюваного світла залежить від хімічного складу використаного у світлодіоді напівпровідника.

У класичному розумінні світлодіод має два електроди: позитивний (анод) та негативний (катод).

На платі mCore робота mBot розташовано три світлодіоди (рис. 2.19), якими можна програмно керувати.

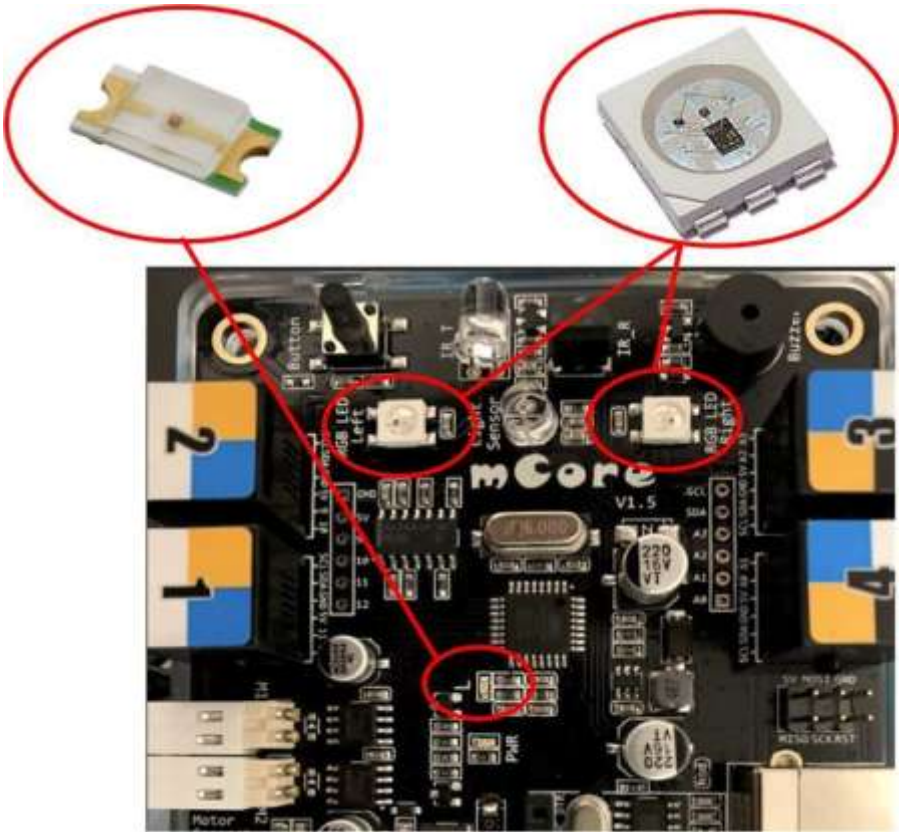


Рисунок 2.19 – Світлодіоди на платі mCore

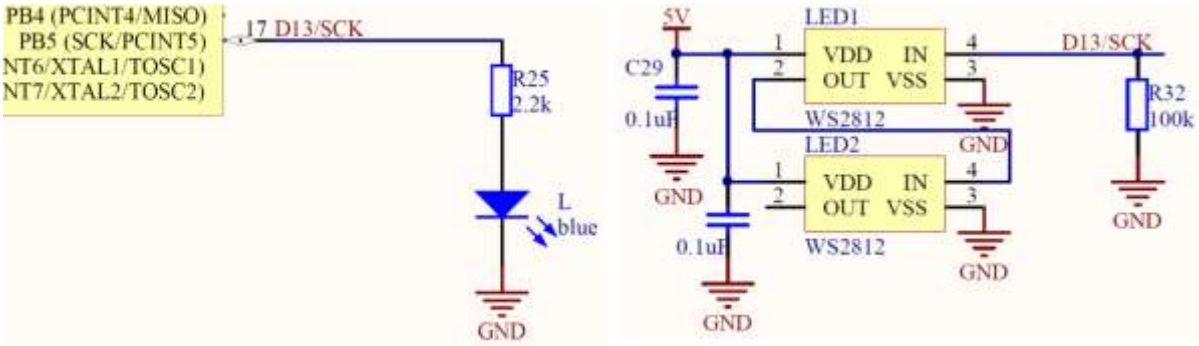


Рисунок 2.20 – Схема підключення світлодіодів на платі mCore [46].

WS2812B – це світлодіодне джерело світла з інтелектуальним керуванням, в яке вбудовані схема управління та мікросхема RGB у корпусі 5050 (5×5 мм) (рис. 2.21). Має вбудований інтелектуальний цифровий порт передачі даних і підсилювач сигналу.

Фізично у WS2812B є три випромінювальних світлодіоди (червоний, синій і зелений) і ШІМ-драйвери, що керують їх яскравістю. ШІМ-драйвери 8-бітові,

Рисунок 2.21 – Вигляд знизу та зверху світлодіода WS2812B

тобто для кожного з кольорів можливі 256 градацій яскравості, і, відповідно, для того, щоб встановити яскравості для кожного з трьох світло діодів, потрібно передати світлодіоду 8 × 3 = 24 біти (3 байти) інформації. Протокол передачі світлодіоду однолінійний з фіксованою швидкістю. Одиниці та нулі інформації про яскравість кодуються тривалістю високого та низького рівнів сигналу в лінії.

Кожен із пікселів WS2812B має 2 виводи живлення (VDD, VSS), вхід (DIN) та вихід (DOUT) (рис. 2.22).

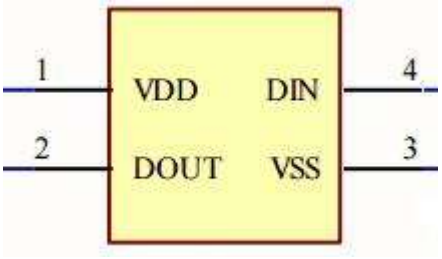


Рисунок 2.22 – Графічне подання світлодіода WS2812B

На вхід DIN подається інформація (24 біти) для встановлення нового кольору. Інформація про колір передається біт за бітом (починаючи з найстаршого) послідовно для кожного з G, R, B колірних компонентів.

Пікселі з'єднуються в ланцюжок. Запис значень кольору в ланцюжку пікселів відбувається у такий спосіб: перші 24 біти, подані на DIN, записує собі у тимчасову пам'ять (колір поки що залишається незмінним з попереднього разу) перший піксел. Наступні біти перший піксел пропускає через себе і надсилає на вихід DOUT. Другий піксел повторює дії першого (залишаючи собі перші 24 біти, що дійшли до нього) і так по ланцюжку. Для того, щоб значення кольорів з тимчасової пам'яті пікселів стали активними, має бути витримана пауза передачі (reset code) протягом 50 мкС. Після цієї паузи можна повторювати цикл знову.

RGB (скорочено від англ. Red, Green, Blue – червоний, зелений, синій) – адитивна колірна модель, що описує спосіб синтезу кольору, за яким червоне, зелене та синє світло накладаються разом, змішуючись у різноманітні кольори. Широко застосовується у техніці, що відтворює зображення за допомогою випромінення світла.

У RGB-моделі колір кодується градаціями складових каналів (Red, Green, Blue). Тому за збільшення величини градації деякого каналу зростає його інтенсивність під час синтезу.

Кількість градацій кожного каналу залежить від розрядності бітового значення RGB. Зазвичай використовують 24-бітову модель, в якій визначається по 8 бітів на кожен канал, і тому кількість градацій дорівнює 256, що дозволяє закодувати $256^3 = 16\,777\,216$ кольорів.

Колірна модель RGB призначена сприймати, подавати та відображати зображення в таких електронних системах, як телебачення та комп'ютери, хоча її також застосовували у традиційній фотографії. Вже до електронного віку, модель RGB мала за собою серйозну теорію, засновану на сприйнятті кольорів людиною.

Для спрощення програмування робота mBot, з точки зору вибору певного кольору, доцільно скористатись сервісами на теренах Інтернету, які надають користувачеві повний спектр палітр (рис. 2.23).

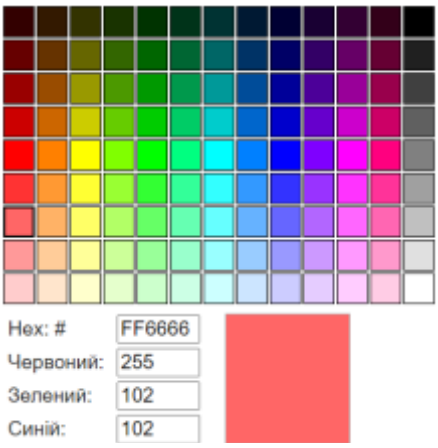


Рисунок 2.23 – Повний спектр палітр

Ось кілька посилань на такі ресурси [47, 48]:

- https://www.rapidtables.org/uk/web/color/RGB_Color.html
- https://paletter.beloweb.name/color-picker/?color=%23fbff00

Для роботи зі світлодіодами WS2812B використовуються три популярні бібліотеки: FastLED, AdafruitNeoPixel та LightWS2812. Працювати з бібліотеками FastLED та Adafruit NeoPixel досить просто. Їх відмінність лише у функціональності та обсязі займаної пам'яті.

2.3.1 Керування світлодіодом L плати mCore у середовищі Arduino IDE

Більшість Arduino-подібних плат (наприклад, mCore), мають вбудований SMT (Surface-mount techology) світлодіод, під'єднаний до виходу 13. Якщо Ви запустите код на таких платах без підключення зовнішнього світлодіода, Ви маєте побачити миготіння вбудованого світлодіода на платі. Для цього потрібно ввести код, наведений у прикладі 2.1. Цей код також можна завантажити з прикладів у середовищі розробки Arduino IDE. Для цього у верхньому меню потрібно вибрати Файл → Приклади → 01.Basics → Blink (рис. 2.24).

```
/*
Засвічуємо вбудований світлодіод L на три секунди, потім вимикаємо його на дві секунди у циклі.
*/

void setup() {
  // Ініціалізуємо цифровий ввід/вивід в режимі виводу.
  // Вихід 13 на платі mCore під'єднатий до світлодіоду які і на більшості плат Arduino
  pinMode(13, OUTPUT);
}

void loop() {
  digitalWrite(13, HIGH);   // засвічуємо світлодіод
  delay(3000);              // очікуємо 3 секунди
  digitalWrite(13, LOW);    // вимикаємо світлодіод
  delay(2000);              // очікуємо 2 секунди
}
```

Приклад 2.1 – Програма для миготіння світлодіода

Нижче покроково розберемось, як ця програма працює.

Обов'язкова процедура `void setup()` виконується один раз під час увімкнення робота або після того, як було натиснено кнопку Reset. У нашому випадку вона містить `pinMode(13, OUTPUT)` – ініціалізує цифровий вхід/вихід 13 у режим виходу.

```
void setup() {
  pinMode(13, OUTPUT);
}
```

Обов'язкова процедура `void loop()` – це нескінченний цикл. Вона містить усі команди для здійснення керуванням певними діями світлодіода, які розташовані між фігурними дужками `{ }`.

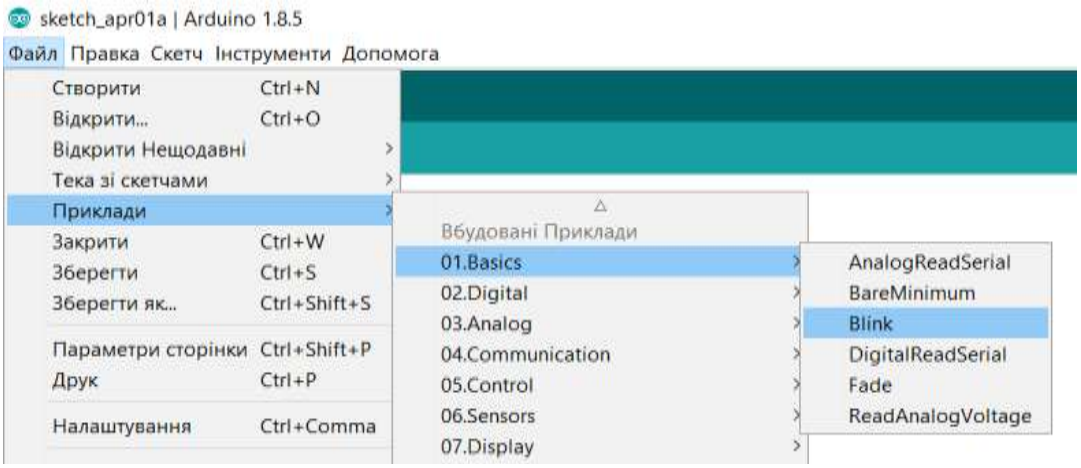


Рисунок 2.24 – Завантаження програми миготіння світлодіода у середовище розробки Arduino IDE

```
void loop() {
  digitalWrite(13, HIGH);
  delay(3000);
  digitalWrite(13, LOW);     // вимикаємо світлодіод
  delay(2000);               // очікуємо 2 секунди
}
```

Функція `digitalWrite()` дозволяє керувати під'єднаним обладнанням до піна шляхом подачі або зняття робочої напруги (у більшості випадків – 5 В). Насправді, `digitalWrite` перетворює 13 пін на реле, що вмикає або вимикає напругу залежно від наших завдань. Подаючи напругу на під'єднаний світлодіод, ми змушуємо його світитись. Крім того, за допомогою `digitalWrite` можна подавати імпульси та передавати інформацію закодованими сигналами та повідомленнями.

Виводимо високий рівень сигналу на 13 пін: `digitalWrite(13, HIGH)`

Очікуємо 3 секунди: `delay(3000);`

Виводимо низький рівень сигналу на 13 пін: `digitalWrite(13, LOW)`

Очікуємо 2 секунди: `delay(2000);`

Після цього підпрограма `void loop()` почне виконувати повторно команди, які знаходяться між її фігурними дужками `{}`, починаючи з циклу що здійснює функцію миготіння червоним світлодіодом. І так до нескінченності.

Зберегти дану програму можна, натиснувши на піктограму «Сохранить» у верхньому меню середовища розробки Arduino IDE.

2.3.2 Керування адресними світлодіодами плати mCore у середовищі mBlock

Запустіть програму mBlock і перевірте налаштування. Використовуйте розташовані справа-зверху в палітрі скриптів (Scripts) блоки з розділів «Контроль» (Control) і «Робот» (Robots) і, перетягуючи їх на робоче поле у центрі mBlock, побудуйте програму, що наведена на рис. 2.25.

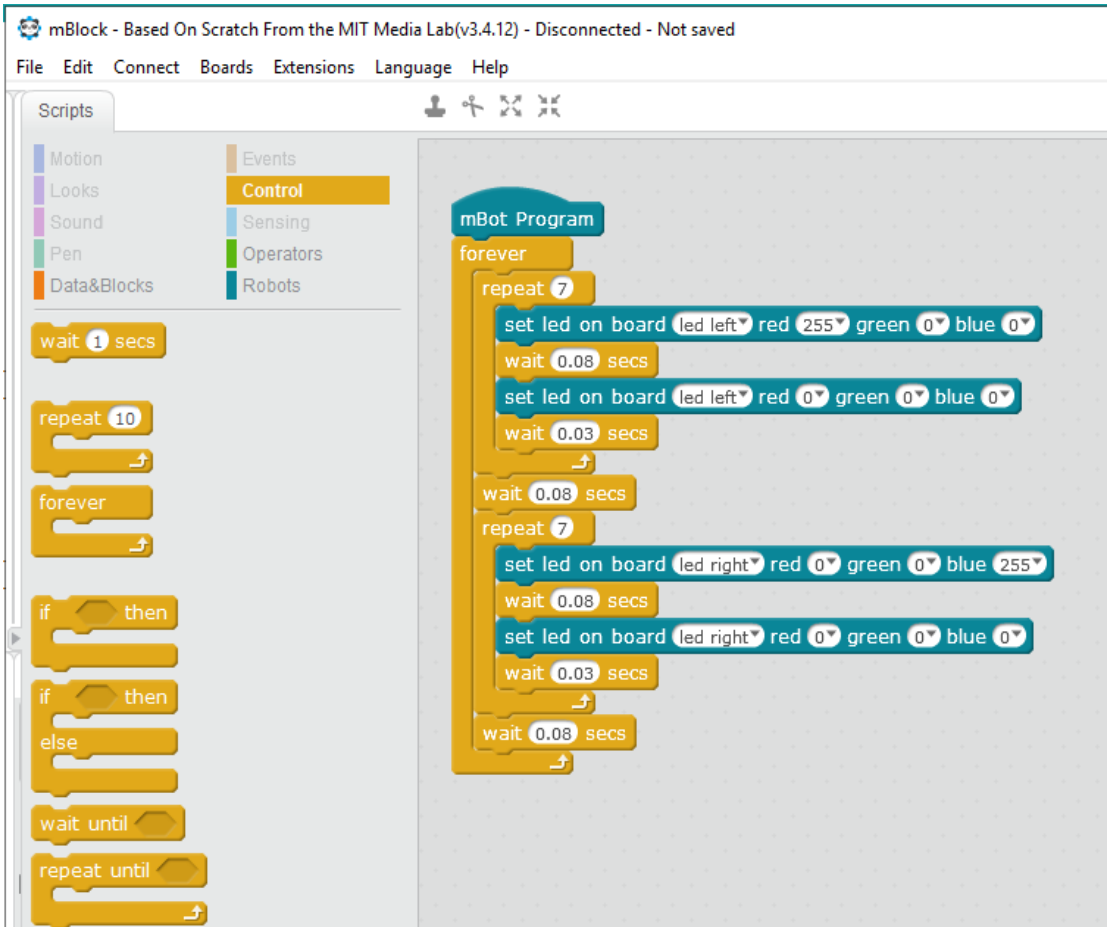


Рисунок 2.25 – Програма керування адресними світлодіодами в mBlock

Далі покроково розглянемо, як ця програма працює.

1. Блок «`mBot Program`» – заголовок програми для роботи mBot.

2. Блок «`forever`» – це нескінченний цикл, усередині якого усі блоки з командами виконуються нескінченну кількість разів, поки ввімкнене живлення робота.

3. Блок «repeat 7» сім разів повторює виконання коду, що міститься у ньому.

4. Блок «set led on board led left red 255, green 0, blue 0» засвічує на лівому світлодіоді плати робота червоний колір на максимальну яскравість, тобто 255, а зелений і синій – на мінімальну, тобто 0 (не засвічує).

5. Блок «wait 0.08 secs» потрібен для того, щоб світлодіод світився протягом 0,08 секунд.

Програма у цей час нічого не буде далі виконувати, просто чекатиме поки пройде 0,08 секунд, тому буде світитись лівий світлодіод червоним кольором.

6. Блок «set led on board led left red 0, green 0, blue 0» гасить усі три кольори на лівому світлодіоді, їх значення дорівнює 0.

7. Блок «wait 0.03 secs» потрібен для того, щоб припинити виконання програми упродовж 0,03 секунди.

Далі код програми буде повторювати блок «repeat» ще шість разів. Після чого продовжить виконання блоків далі.

8. Блок «wait 0.08 secs» потрібен для того, щоб припинити виконання програми упродовж 0,08 секунди. У цей час світлодіоди не світяться.

9. Блок «repeat 7» – сім разів повторює виконання коду, що міститься у ньому.

10. Блок «set led on board led right red 0, green 0, blue 255» – засвічує на правому світлодіоді плати робота синій колір на максимальну яскравість, тобто 255, а зелений і червоний – на мінімальну, тобто 0 (не засвічує).

11. Блок «wait 0.08 secs» потрібен для того, щоб припинити виконання програми впродовж 0,08 секунди. У цей час світлодіод світиться синім кольором.

12. Блок «set led on board led right red 0, green 0, blue 0» – гасить усі три кольори на правому світлодіоді, їх значення дорівнює 0.

13. Блок «wait 0.03 secs» потрібен для того, щоб припинити виконання програми упродовж 0,03 секунди.

Далі код програми буде повторювати блок «repeat» ще шість разів. Після чого продовжить виконання блоків далі.

14. Блок «wait 0.08 secs» потрібен для того, щоб припинити виконання програми упродовж 0,08 секунди. У цей час світлодіоди не світяться.

15. Після цього циклу «forever» повторює усі команди, починаючи з блоку «repeat» у пункті 3, і так нескінченну кількість разів.

Збережіть створену програму, обравши у верхньому меню File → Save Project As (рис. 2.26).

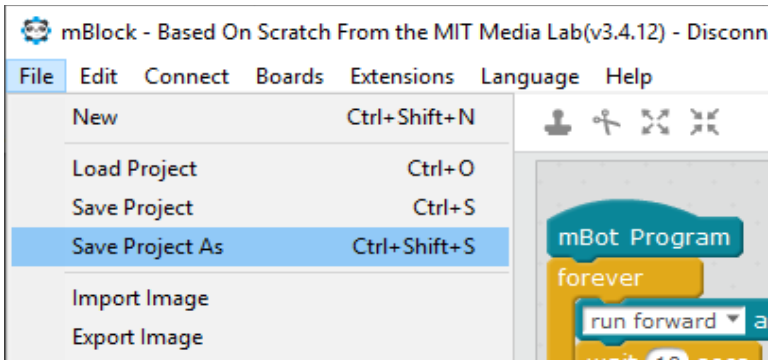


Рисунок 2.26 – Зберігання програми в mBlock

Завантаження програми у пам’ять робота з mBlock

Перед завантаженням програми у пам’ять mBot перевірте з’єднання робота з ПК та ввімкніть живлення робота. Коли будете готові до завантаження, натисніть кнопку «Upload to Arduino» і зачекайте деякий час, поки програма не завантажиться у робот mBot. Після закінчення з’явиться повідомлення про успішне завантаження «Upload Finish» (рис. 2.27)

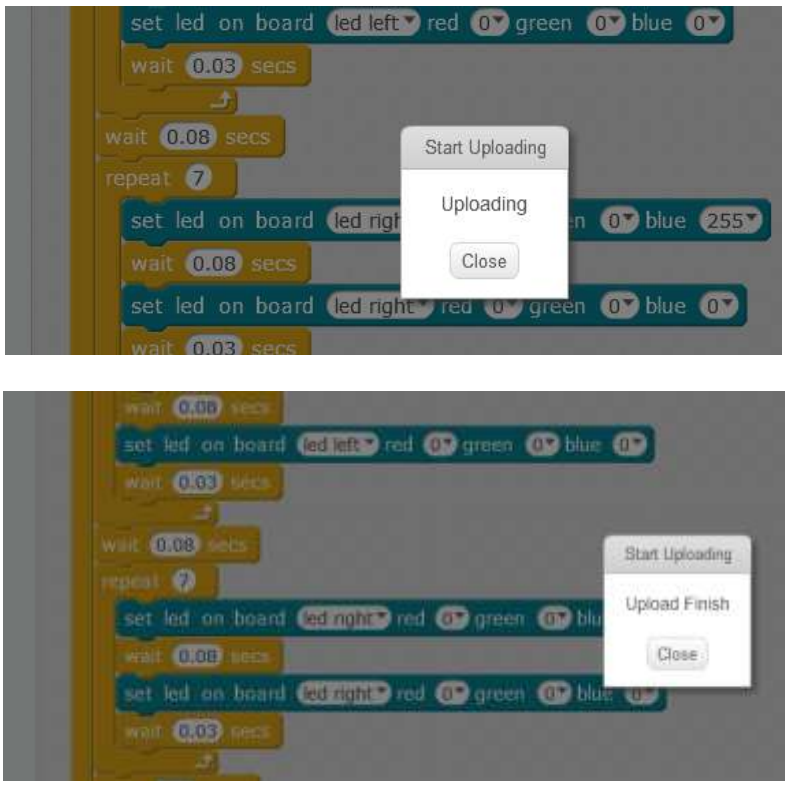


Рисунок 2.27 – Завантаження програми у пам’ять mBot

Після цього mBot відразу розпочне миготіти лівим та правим світлодіодом по черзі, відповідно, червоним і синім кольорами..

Керування адресними світлодіодами, вбудованими у плату mCore робота mBot у середовищі Arduino IDE

Запустіть середовище Arduino IDE та наберіть нижченаведену програму (скетч), що подана у прикладі 2.2.

```
#include <Adafruit_NeoPixel.h>  // під'єднуємо бібліотеку

// створюємо об'єкт strip із потрібними характеристиками
Adafruit_NeoPixel strip (2, 13, NEO_GRB + NEO_KHZ800);

void setup() {
  strip.begin();     // ініціалізуємо адресні світлодіоди
  strip.setBrightness(255);   // задаємо яскравість
                                 світлодіода (максимум 255)
}

void loop() {

  // блимаємо синім світлодіодом
  for (int i = 1; i < 7; i++) {
    strip.setPixelColor(0, strip.Color(255, 0, 0));
    strip.show();
    delay(80);
    strip.setPixelColor(0, strip.Color(0, 0, 0));
    strip.show();
    delay(30);
  }
  delay(80);
  // блимаємо синім світлодіодом
  for (int i = 1; i < 7; i++) {
    strip.setPixelColor(1, strip.Color(0, 0, 255));
    strip.show();
    delay(80);
    strip.setPixelColor(1, strip.Color(0, 0, 0));
    strip.show();
    delay(30);
  }
  delay(80);
}
```

Приклад 2.2 – Програма керування адресними світлодіодами

Далі покроково розберемося, як ця програма працює.

Під'єднуємо бібліотеку Adafruit_NeoPixel.h для адресних світлодіодів на платі mCore директивою: `#include <Adafruit_NeoPixel.h>`

Потім створюємо об'єкт strip із потрібними характеристиками:

```
Adafruit_NeoPixel strip (2, 13, NEO_GRB + NEO_KHZ800);
```

де вказують кількість світлодіодів (у нашому випадку два), фізичний порт, до якого їх під'єднано, палітру RGB, і частоту обміну даними 800 кГц.

Обов'язкова процедура `void setup()` виконується один раз під час увімкнення робота, або після того, як було натиснено кнопку Reset.

У нашому випадку вона містить `strip.begin()` – ініціалізацію адресних світлодіодів та `strip.setBrightness(255)` (де ми задаємо яскравість світло діодів: 0 – мінімальна яскравість (не світяться), 255 – максимальна яскравість).

Обов'язкова процедура `void loop()` – це нескінченний цикл, тобто, аналог блока «forever» в mBlock (Scratch). Вона містить усі команди для виконання певного алгоритму роботи, які розташовані між фігурними дужками { }.

Далі виконується цикл, у якому здійснює процедура миготіння червоним кольором першого чіпа адресного світлодіода WS-2812B 5050 RGB.

```
// блимаємо синім світлодіодом
  for (int i = 1; i < 7; i++) {
  }
```

Цикл for виконується у 3 кроки.

1. Оголошення змінної. Ініціалізується лічильник циклу `int i`. Ця частина виконується лише один раз коли цикл виконується уперше.

2. Умова. Якщо вона дорівнює false, то цикл завершує виконання. Якщо ж вона дорівнює true – то виконується тіло циклу.

3. Інкремент. Збільшення лічильника циклу на одиницю. Після цього виконується саме тіло циклу.

```
    strip.setPixelColor(0, strip.Color(255, 0, 0));
    strip.show();
    delay(80);
    strip.setPixelColor(0, strip.Color(0, 0, 0));
    strip.show();
    delay(30);
```

Після цього цикл повертається до кроку № 2. І так допоки умова циклу не дорівнюватиме false.

Пауза в Arduino IDE вимірюється у мілісекундах. В одній секунді 1000 мілісекунд, тому команди виглядають так:

```
delay(80); // чекати 2 секунди.
```

Після цього виконується цикл, у якому здійснює процедура миготіння синім кольором другого чіпа адресного світлодіода WS-2812B 5050 RGB.

```
for (int i = 1; i < 7; i++) {
    strip.setPixelColor(1, strip.Color(0, 0, 255));
    strip.show();
    delay(80);
    strip.setPixelColor(1, strip.Color(0, 0, 0));
    strip.show();
    delay(30);
  }
```

Далі задається пауза довжиною 80 мілісекунд – `delay(80);`

Після цього підпрограма `void loop()` почне виконувати повторно команди, які знаходяться між її фігурними дужками `{}`, починаючи з циклу, що здійснює функцію миготіння червоним світлодіодом. І так до нескінченності.

Текст програми містить коментарі, які починаються з символів «//». Таким чином ми можемо коментувати програму, щоб покращити їх розуміння. Ці коментарі не впливають на команди програми, але зручні для програмістів, оскільки використання коментарів у програмі вважається показником високого рівня програміста, тому не забувайте писати коментарі, вони Вам не раз допоможуть у майбутньому.

Зберегти дану програму можна, натиснувши на піктограму «Зберегти» у верхньому меню середовища розробки Arduino IDE (рис. 2.28).

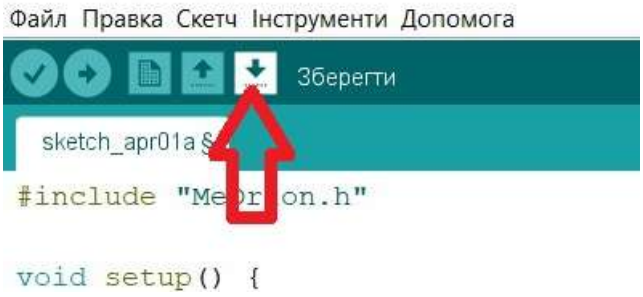


Рисунок 2.28 – Збереження програми в Arduino IDE

Завантаження програми у пам’ять мікроконтролера робота mBot із Arduino IDE

Перед завантаженням програми у пам’ять mBot перевірте з’єднання робота зі своїм комп’ютером та ввімкніть живлення робота. Коли будете готові до завантаження, клікніть по піктограмці «Вивантажити» (рис. 2.29).

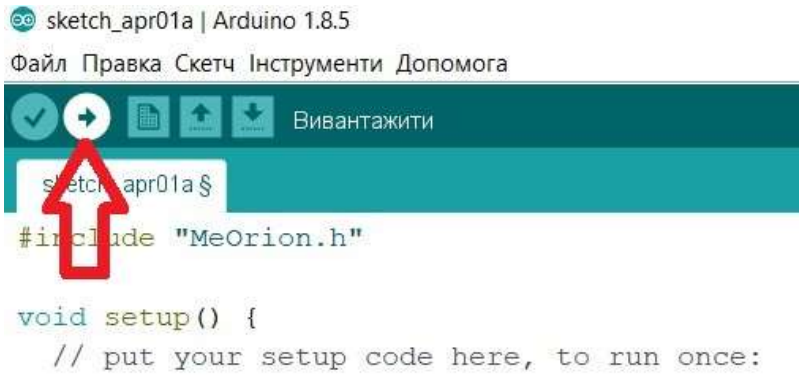


Рисунок 2.29 – Завантаження програми з Arduino IDE в mBot

У випадку успішного завантаження програми з Arduino IDE у пам’ять мікроконтролера робота mBot Ви побачите повідомлення «Вивантаження виконано» (рис. 2.30).

```
  delay(1000);                      // wait for a second
}
```

```
Вивантаження виконано.

Скетч використовує 928 байтів (2%) місця зберігання для програм. Мож
Глобальні змінні використовують 9 байтів (0%) динамічної пам'яті, і
```

Рисунок 2.30 – Вікно, що вказує на успішне завантаження програми в mBot

Якщо Ви допустили помилку, то повідомлення про це з'явиться у нижній частині вікна на чорному фоні, а завантаження програми буде перервано (рис. 2.31). У цьому випадку прочитайте повідомлення про помилку, виправте її та повторіть завантаження.

```
// the loop function runs over and over again forever
void loop() {
  digitalWrite(LED_BUILTIN, HIGH);   // turn the LED on (HIGH is the voltage
  delay(1000);                       // wait for a second
  digitalWrite(LED_BUILTIN, LOW);    // turn the LED off by making the volta
  delay(1000);                       // wait for a second
}
```

```
Проблема вивантаження в плату. Зверніться до http://www.arduino.cc/en/Guide/Troubleshooting#upload для пошуку рішення.

Проблема вивантаження в плату. Зверніться до http://www.arduino.cc/en/Guide/
```

Рисунок 2.31 – Вікно, що вказує на проблему
під час завантаження програми в mBot

Після успішного завантаження mBot відразу розпочне обертати колесами уперед-назад. Будьте готові до цього, тримайте робота в руках!

Тепер можна вимкнути живлення, витягнути USB-кабель та перенести робота на місце випробування.

2.4 Керування рухом робота mbot за допомогою двигунів постійного струму

Двигун (електрична машина постійного струму) є одним з найпростіших в експлуатації, завдяки чому його так часто застосовують у пристроях радіоелектроніки та робототехніки. Така популярність зумовлена простотою живлення та управління – для цього подаються два полюси від джерела живлення (негативний та позитивний), і при протіканні струму по обмотках відбувається обертання вала. У випадку зміни полярності двигун здійснює реверсивний рух (рис. 2.32) [49].

Для того, щоб керувати напрямком обертання двигуна постійного струму без перепідключення живлення до його контактів, використовують схему, яка називається Н-мостом. Н-міст (промовл. ейч міст) – це електронна схема, яка може керувати обертанням двигуна в обох напрямках. Н-мости використовуються у багатьох пристроях, найпоширенішим з яких є управління двигунами у роботах. Схема отримала назву «Н-міст» тому, що вона використовує чотири транзистори (VT1–VT4), підключені таким чином, що схема виглядає як буква «Н» (рис. 2.33).

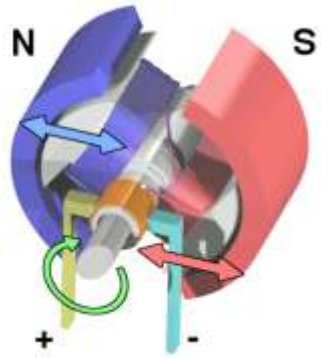
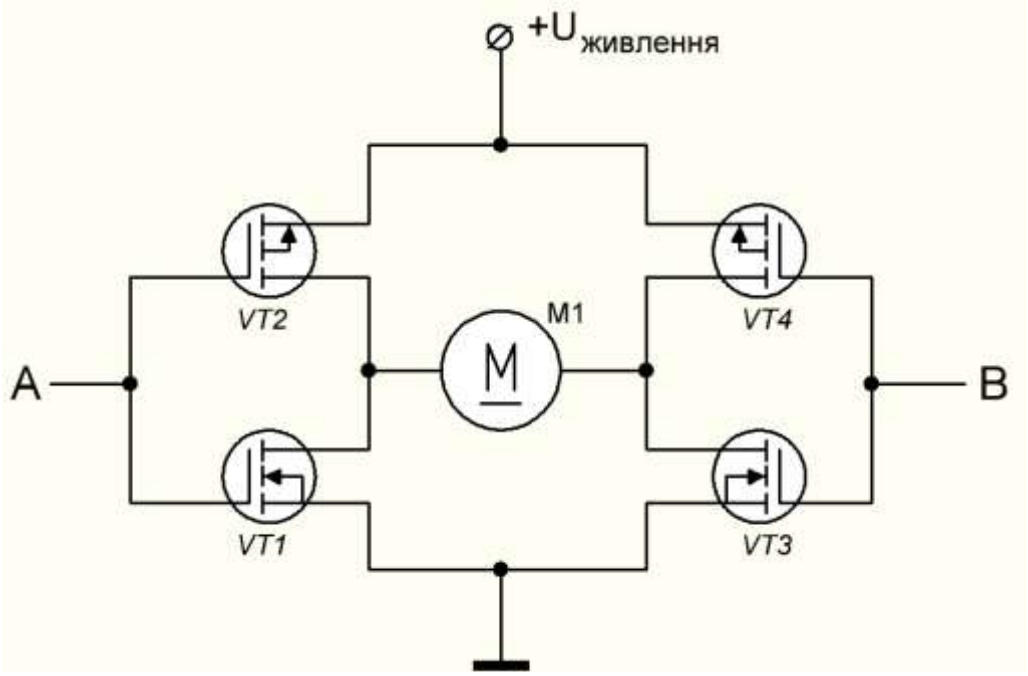


Рисунок 2.32 – Схематичне зображення двигуна постійного струму



	Стоп	Стоп	Вліво	Вправо
A	0	1	1	0
B	0	1	0	1

Рисунок 2.33 – Н-міст і таблиця сигналів керування двигуном

Для керування двигуном постійного струму в роботі mBot застосовується мікросхема Н-моста TB6612 (рис. 2.34) [50].

TB6612 – це двоканальний драйвер двигунів, який ідеально підходить для керування за допомогою мікроконтролера двома малопотужними елек-

тродвигунами постійного струму, або для керування біполярним крововим двигуном. Драйвер оснований на MOSFET H-мостах, які забезпечують більші струми двигунам і менші для живлення логіки.

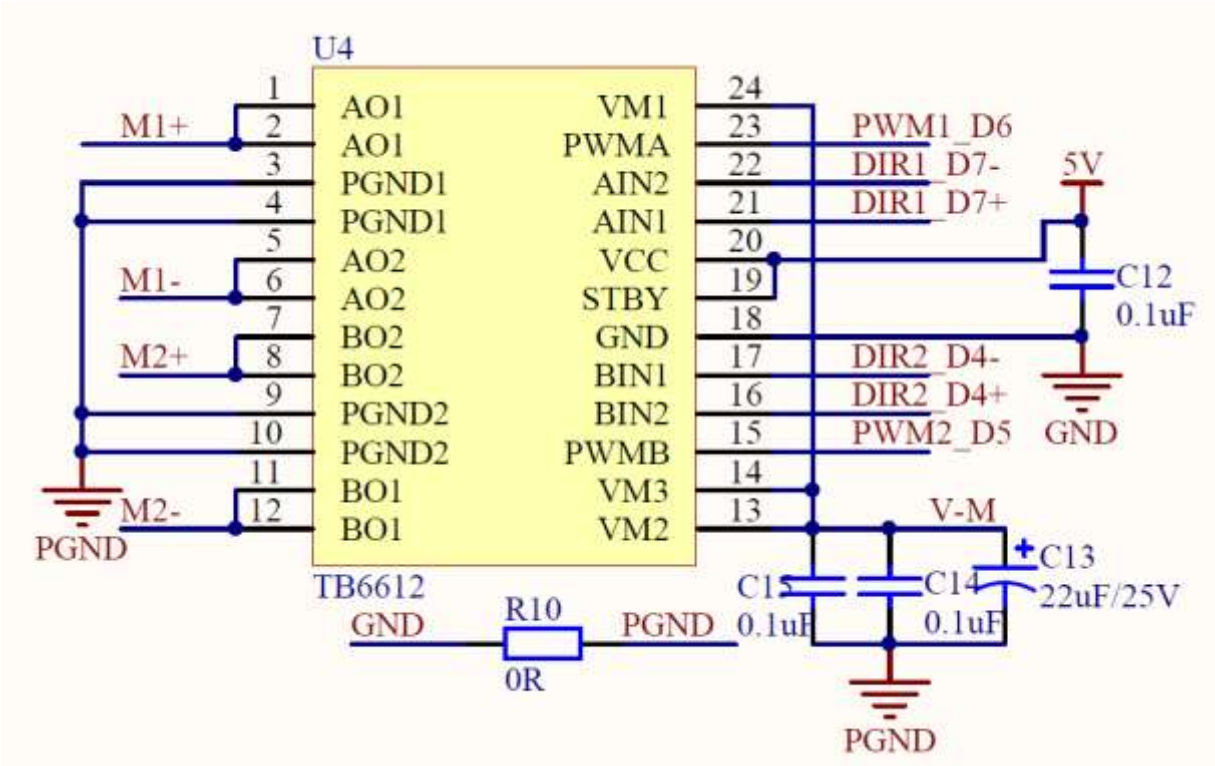


Рисунок 2.34 – Схема підключення мікросхеми H-моста TB6612

Усі виводи керування мають внутрішню «підтяжку» до землі. Кожен із обох каналів має два контакти: керування напрямом і регулювання швидкістю обертання, на який надходить сигнал ШІМ із частотою до 100 кГц.

Широтно-імпульсна модуляція – це операція отримання аналогового значення, що змінюється, за допомогою цифрових пристроїв. Пристрої використовуються для отримання прямокутних імпульсів – сигналу, який постійно перемикається між максимальним і мінімальним значеннями. Для отримання різних аналогових величин змінюється ширина імпульсу. При досить швидкій зміні періодів увімкнення-вимкнення можна подавати постійний сигнал між 0 і напругою живлення на двигун, тим самим керуючи швидкістю його обертання [51].

На графіку (рис. 2.35) вертикальні лінії позначають постійні часові періоди. Тривалість періоду обернено пропорційна частоті ШІМ. Тобто, якщо частота ШІМ становить 500 Гц, то вертикальні лінії відзначатимуть інтервали тривалістю 2 мілісекунди кожен. Виклик функції analogWrite() з масштабом 0 – 255 означає, що значення analogWrite(255) буде відповідати 100% робочому циклу (постійне увімкнення живлення $U_ж$), а значення analogWrite(127) – 50% робочому циклу.

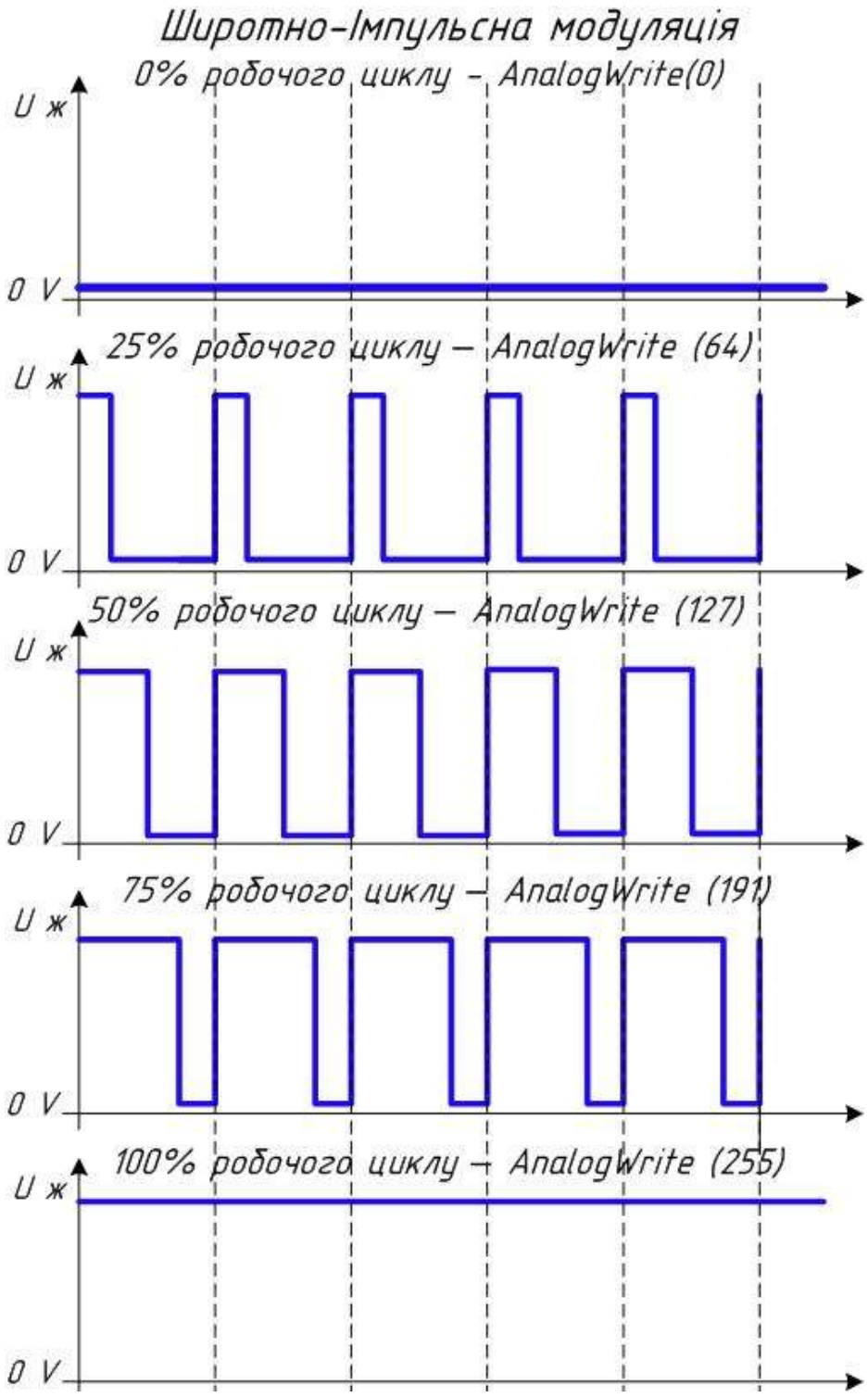


Рисунок 2.35 – Діаграми ШІМ з різною частотою

Програмуємо рух робота mBot уперед-назад у mBlock

Запустіть програму mBlock і перевірте налаштування.

Використовуйте розташовані справа-зверху в палітрі скриптів (Scripts) блоки з розділів «Контроль» (Control) і «Робот» (Robots) і, перетягуючи їх на робоче поле у центрі mBlock, побудуйте, наведену на рис. 2.36 програму.

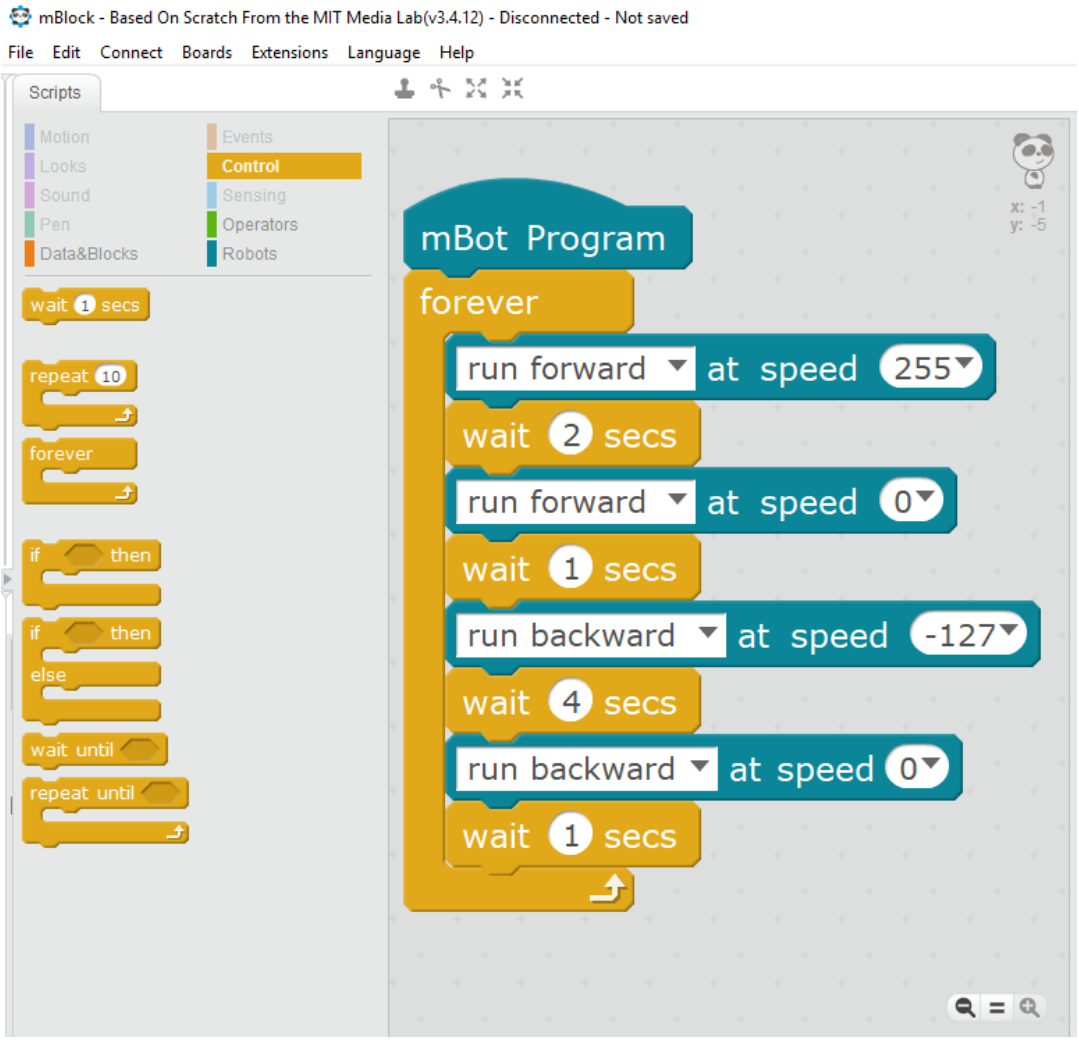


Рисунок 2.36 – Програма керування рухом робота

Далі покроково розглянемо, як ця програма працює.

1. Блок «`mBot Program`» – це заголовок програми для роботи mBot.

2. Блок «`forever`» – це нескінченний цикл, усередині якого всі блоки з командами виконуються нескінченну кількість разів, поки ввімкнене живлення робота.

3. Блок «`run forward at speed 255`» вмикає обертання двигунів робота mBot так, щоб він рухався вперед на швидкості 255. Це максимальна швидкість для двигунів.

4. Блок «`wait 2 sec`» потрібен для того, щоб робот рухався вперед 2 секунди.

Програма у цей час нічого не буде робити, просто чекатиме поки пройде 2 секунди, тому робот mBlock за цей час подолає деяку відстань, рухаючись уперед.

5. Блок «`run forward at speed 0`» вимикає обертання двигунів робота mBot так, щоб він не рухався, тобто його швидкість дорівнювала 0.

6. Блок «`wait 1 sec`» потрібен для того, щоб робот не рухався протягом 1 секунди.

7. Блок «`run backward at speed 127`» – це команда для роботи mBot рухатися назад (заднім ходом) на швидкості 127.

8. Блок «`wait 4 sec`» потрібен для того, щоб робот рухався у зворотному напряму протягом 4 секунд.

9. Блок «`run forward at speed 0`» вимикає обертання двигунів робота mBot так, щоб він не рухався, тобто його швидкість дорівнювала 0.

10. Блок «`wait 1 sec`» потрібен для того, щоб робот не рухався протягом 1 секунди.

11. Після цього циклу «`forever`» повторює усі команди, починаючи з блока «`run backward at speed 255`» у пункті 3, і так нескінченну кількість разів.

Збережіть створену програму обравши у верхньому меню File → Save Project As (див. рис. 2.26).

Завантаження програми у пам'ять робота з mBlock

Перед завантаженням програми у пам'ять mBot перевірте з'єднання робота з ПК і ввімкніть живлення робота.

Увага, після успішного завантаження програми робот відразу почне обертати колесами вперед-назад. Будьте готові до цього, тримаючи робота в руках!

Коли будете готові до завантаження, натисніть кнопку «Upload to Arduino» і зачекайте допоки програма не завантажиться у робот mBot. Після закінчення з'явиться повідомлення про успішне завантаження «Upload Finish» (див. рис. 2.27)

Після цього mBot відразу розпочне повертати колесами уперед-назад! Тепер можна вимкнути живлення, відімкнути USB-кабель від роботи та перенести робота на місце випробування.

Програмуємо рух робота mBot уперед-назад в Arduino IDE

Запустіть середовище Arduino IDE та введіть у ньому поданий у прикладі 2.3 скетч.

```
uint8_t Speed_motor = 255; // Швидкість двигунів рух вперед

void setup()  {
  pinMode(4, OUTPUT);      // настроювання порта 4 на вихід
  pinMode(7, OUTPUT);      // настроювання порта 7 на вихід
  pinMode(5, OUTPUT);      // настроювання порта 5 на вихід
  pinMode(6, OUTPUT);      // настроювання порта 6 на вихід
}

void loop() {
  // Рух робота mBot вперед
  digitalWrite(4, HIGH);       // Обертання правого двигуна за
годинниковою стрілкою
```

```
  analogWrite (5, Speed_motor);// швидкість обертання лівого
двигуна (ШІМ=255)
  digitalWrite(7, LOW);        // Обертання лівого двигуна
проти годинникової стрілки
  analogWrite (6, Speed_motor);// швидкість обертання правого
двигуна (ШІМ=255)
  delay(2000);                 // Очікувати 2с., рух вперед
  // Зупинка робота mBot
  analogWrite (5, 0);  // Зупинка правого двигуна (ШІМ=0)
  analogWrite (6, 0);  // Зупинка лівого двигуна (ШІМ=0)
  delay(100);          // Очікувати 1с., робот не рухається
  // Рух робота mBot назад
  digitalWrite(4, LOW);        // Обертання правого двигуна
проти годинникової стрілки
  analogWrite (5, Speed_motor/2); //швидкість обертання лівого
двигуна (ШІМ=127)
  digitalWrite(7, HIGH);       // Обертання лівого двигуна за
годинниковою стрілкою
  analogWrite (6, Speed_motor/2); // швидкість обертання пра-
вого двигуна (ШІМ=127)
  delay(4000);                 // Очікувати 2с., рух назад
  // Зупинка робота mBot
  analogWrite (5, 0); // Зупинка правого двигуна (ШІМ=0)
  analogWrite (6, 0); // Зупинка лівого двигуна (ШІМ=0)
  delay(100);         // Очікувати 1с., рух не здійснюється
}
```

Приклад 2.3 – Програма керування рухом mBot уперед-назад в Arduino IDE

Далі детально проаналізуємо роботу програми. Значення швидкості двигунів 255 будемо зберігати у числовій змінній `Speed_motor`, оскільки це зручно для корегування програми у майбутньому:

```
uint8_t motorSpeed = 255;
```

Обов'язкова процедура `void setup()` виконується один раз під час увімкнення робота, або після того, як було натиснено кнопку Reset. У нашому випадку вона містить `pinMode()`. Функція `pinMode()` в Arduino IDE встановлює режим роботи заданого піна, на вхід чи вихід. Цифровий пін може бути у двох станах. У режимі входу пін зчитує напругу від 0 до 5 Вольт, а у режимі виходу на пін можна подавати напругу у тому ж діапазоні.

```
void setup()  {
  pinMode(4, OUTPUT);
  pinMode(7, OUTPUT);
  pinMode(5, OUTPUT);
  pinMode(6, OUTPUT);
}
```

У нашому випадку піни 4–7 налаштовані на вихід.

Обов'язкова процедура `void loop()` – це нескінченний цикл, тобто аналог блока «`forever`» у mBlock (Scratch). Вона містить усі команди для руху робота, які розташовані між фігурними дужками `{}`.

Під час руху робота потрібно врахувати, що лівий двигун зміщений щодо правого на 180 градусів, відповідно, для того, щоб робот mBot рухався вперед, його лівий двигун має обертатись проти годинникової стрілки, а правий – за.

```
// Рух робота mBot вперед
  digitalWrite(4, HIGH);
  analogWrite (5, Speed_motor);
  digitalWrite(7, LOW);
  analogWrite (6, Speed_motor);
  delay(2000);
```

Нагадаємо, що пауза в Arduino IDE вимірюється у мілісекундах, тому команди виглядатимуть так:

```
delay(2000); // чекати 2 сек. поки робот рухається вперед
```

Для того, щоб робот зупинився, потрібно задати такі команди:

```
  // Зупинка робота mBot
  analogWrite (5, 0);  // Зупинка правого двигуна (ШІМ=0)
  analogWrite (6, 0);  // Зупинка лівого двигуна (ШІМ=0)
  delay(100);         // Очікувати 1сек., робот не рухається
```

Для початку руху робота назад нам достатньо у командах керування двигунами змінити значення стану відповідних пінів на протилежний та задати швидкість їх обертання. У нашому випадку швидкість руху робота у зворотному напрямку вдвічі менша, а час удвічі більший:

```
// Рух робота mBot назад
  digitalWrite(4, LOW);
  analogWrite (5, Speed_motor/2);
  digitalWrite(7, HIGH);
  analogWrite (6, Speed_motor/2);
  delay(4000);
```

І ще раз застосуємо команди для зупинки робота після того, як він завершив рух «заднім ходом»:

```
// Зупинка робота mBot
  analogWrite (5, 0);  // Зупинка правого двигуна (ШІМ=0)
  analogWrite (6, 0);  // Зупинка лівого двигуна (ШІМ=0)
  delay(100);         // Очікувати 1сек., робот не рухається
```

Після цього підпрограма `void loop()` буде виконувати повторно команди, які знаходяться між її фігурними дужками `{}`, починаючи з руху робота вперед і так до нескінченності.

Не забудьте зберегти дану програму.

Програмуємо рух робота mBot уперед-назад в Arduino IDE з використанням бібліотеки Makeblock

Нагадаємо, що для полегшення програмування робота mBot в Arduino IDE доцільно використовувати бібліотеку Makeblock.

Запустіть середовище Arduino IDE та введіть скетч, наведений у прикладі 2.4

```
#include "MeOrion.h"      //під'єднуємо бібліотеку Makeblock
  // об'являємо двигуни
MeDCMotor motorLeft(M1);  //лівий двигун
MeDCMotor motorRight(M2); //правий двигун
uint8_t motorSpeed = 255; // Задаємо швидкість двигунів 255

void setup() {
}

void loop() {
  // Рух робота mBot вперед
motorLeft.run(-motorSpeed);  // Обертання лівого двигуна
motorRight.run(motorSpeed);  // Обертання правого двигуна
delay(2000);                 // Очікувати 2с., рух вперед
motorLeft.stop();            // Зупинка лівого двигуна
motorRight.stop();           // Зупинка правого двигуна
delay(100);          // Очікувати 1с., робот не рухається
  // Рух робота mBot назад
motorLeft.run(motorSpeed/2);  // Обертання лівого двигуна
motorRight.run(-motorSpeed/2);// Обертання правого двигуна
delay(4000);                  // Очікувати 4с., рух назад
motorLeft.stop();             // Зупинка лівого двигуна
motorRight.stop();            // Зупинка правого двигуна
delay(100);          // Очікувати 1с., робот не рухається
}
```

Приклад 2.4 – Програма керування рухом робота mBot уперед-назад

Далі детально розберемося, як ця програма працює.

Під'єднуємо бібліотеку makeblock для плати mCore директивою `#include "MeOrion.h"`.

Потім створюємо об'єкти – лівий та правий двигуни, вказуючи у дужках порти, до яких вони під'єднані на платі mCore:

```
MeDCMotor motorLeft(M1);       //лівий двигун
MeDCMotor motorRight(M2);      //правий двигун
```

Значення швидкості двигунів 255 зберігатимемо у числовій змінній `motorSpeed: uint8_t motorSpeed = 255;`

Обов'язкова процедура `void loop()` – це нескінченний цикл, тобто аналог блока «`forever`» у mBlock (Scratch). Вона містить усі команди для руху робота, які розташовані між фігурними дужками `{}`. Але нам не потрібно, щоб вона містила команди тому вона порожня:

```
void setup()
{
}
```

Під час руху робота потрібно врахувати, що лівий двигун зміщений щодо правого на 180 градусів, відповідно, для того, щоб робот mBot рухався вперед, його лівий двигун має обертатись проти годинникової стрілки `motorLeft.run(-255)` (числове значення швидкості зі знаком мінус), а правий – за `motorRight.run(255)` (числове значення швидкості зі знаком плюс):

```
// Рух робота mBot вперед
motorLeft.run(-motorSpeed);
motorRight.run(motorSpeed);
```

Пауза у 2 секунди

```
delay(2000);
```

Для того, щоб робот зупинився нам потрібно задати такі команди:

```
motorLeft.stop();
motorRight.stop();
delay(100);
```

Для того, щоб робот рухався назад, нам достатньо у командах керування двигунами змінити знак числового значення швидкості на протилежний та задати швидкість їх обертання. У нашому випадку швидкість руху робота у зворотному напрямку вдвічі менша, а час – удвічі більший:

```
// Рух робота mBot назад
motorLeft.run(motorSpeed / 2);
motorRight.run(-motorSpeed / 2);
delay(4000);
```

І ще раз застосуємо команди для зупинки робота, після того як він завершив рух «заднім ходом»:

```
motorLeft.stop();
motorRight.stop();
delay(100);
```

Після цього підпрограма `void loop()` почне виконувати повторно команди, які знаходяться між її фігурними дужками `{}`, починаючи з руху робота вперед, і так нескінченну кількість разів.

Завантаження програми у пам’ять мікроконтролера робота mBot із Arduino IDE

Перед завантаженням програми у пам’ять mBot перевірте з’єднання робота з ПК та ввімкніть живлення робота.

Коли будете готові до завантаження, клікніть по піктограмці «Вивантажити» (див. рис. 2.29). У випадку успішного завантаження програми із Arduino IDE у пам’ять мікроконтролера робота mBot ви будете спостерігати повідомлення «Вивантаження виконано» (див. рис. 2.30).

Якщо Ви допустили будь-яку помилку, то повідомлення про це з’явиться у нижній частині вікна на чорному фоні, а завантаження програми буде перервано (див. рис. 2.31). У цьому випадку прочитайте повідомлення про помилку, виправте її та повторіть завантаження.

Нагадаємо, що після успішного завантаження mBot відразу розпочне обертати колесами вперед-назад. Тепер можна вимкнути живлення, від’єднати USB-кабель від робота та перенести останнього на місце випробування.

2.5 Керування серводвигуном за допомогою Arduino сумісної плати mcore робота mBot

Серводвигун – це такий вид приводу, який може точно керувати параметрами руху (рис. 2.37). Іншими словами, це двигун, який може повертати свій вал на певний кут або підтримувати безперервне обертання з точним періодом.

Рисунок 2.37 – Загальний вигляд серводвигуна

Схема роботи серводвигуна заснована на використанні зворотного зв'язку (контур із замкненою схемою, в якому сигнали на вході та виході не узгоджені). Як серводвигун може виступати будь-який тип механічного приводу, у складі якого є сенсор і блок керування, що автоматично підтримує всі встановлені параметри на сенсорі. Конструкція серводвигуна складається з двигуна, датчика позиціонування та системи керування. Основним завданням таких пристроїв є реалізація у сфері сервомеханізмів. Також серводвигуни нерідко використовуються у таких сферах, як обробка матеріалів, виробництво транспортного обладнання, обробка деревини, виготовлення металевих листів, виробництво будматеріалів та інші.

У робототехніці серводвигуни часто використовується для найпростіших механічних дій:

- Повернути далекомір або інші датчики на певний кут для того, щоб виміряти відстань у вузькому секторі огляду робота.
- Зробити невеликий рух кінцівкою чи головою.
- Для створення роботів-маніпуляторів.
- Для реалізації механізму рульового керування.
- Відкрити або закрити дросель, заслінку або інший механізм.

Звичайно, сфера застосування серводвигунів у реальних проєктах набагато ширша, а тут наведено лише найпопулярніші приклади.

Схема та типи серводвигунів

Принцип роботи серводвигуна ґрунтується на зворотному зв’язку з одним або кількома системними сигналами [52]. Вихідний показник подається на вхід, де порівнюється його значення з задавальною дією і виконуються необхідні дії (наприклад, вимикається двигун). Найпростішим варіантом реалізації є змінний резистор (потенціометр), який обертається валом: у випадку зміни параметрів резистора змінюються параметри струму, що живить двигун (рис. 2.38).

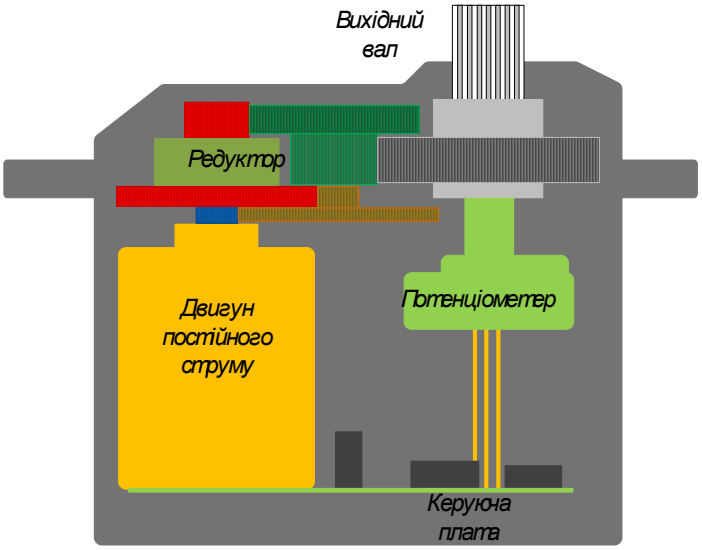


Рисунок 2.38 – Двигун постійного струму, керований потенціометром

У реальних серводвигунах механізм керування набагато складніший і використовує вбудовані мікросхеми – контролери. Залежно від типу використовуваного механізму зворотного зв’язку виділяють *аналогові* та *цифрові* серводвигуни. Перші використовують щось подібне до потенціометра, другі – контролери.

Уся схема керування серводвигуна знаходиться всередині корпусу, сигнали керування та живлення подаються, як правило, по трьох провідниках: земля, живлення та сигнал керування.

Серводвигуни безперервного обертання 360, 180 та 270 градусів

Виділяють два основних види серводвигунів – з безперервним обертанням та з фіксованим кутом (найчастіше, 180º або 270º). Відмінність серводвигуна з фіксованим обертанням полягає у наявності механічних елементів конструкції, які можуть блокувати рух вала поза заданими параметрами кутів. Так, досягши кута 180º, вал вплине на обмежувач, а той віддасть команду на вимкнення двигуна. Серводвигуни безперервного обертання таких обмежувачів не мають.

Крутний момент і швидкість повороту

Крутний момент – векторна фізична величина, що дорівнює добутку радіус-вектора, проведеного від осі обертання до точки прикладання сили, на вектор цієї сили. Характеризує обертальну дію сили на тверде тіло. Ця

характеристика показує, наскільки важкий вантаж серводвигун здатний утримати у спокої на важелі заданої довжини. Якщо момент серводвигуна, дорівнює 5 кг×см, то це означає, що серводвигун утримає в горизонтальному положенні вантаж, закріплений на важелі довжиною 1 см, вага якого становить 5 кг. Або, що еквівалентно, важіль довжиною 5 см, до якого підвісили 1 кг.

Швидкість серводвигуна вимірюється інтервалом часу, який потрібний важелю серводвигуна, щоб повернутися на 60°. Характеристика 0,1 с. / 60° означає, що серводвигун повертається на 60° за 0,1 с. З неї нескладно обчислити швидкість у більш звичній величині, обертах за хвилину, але так склалося, що для опису серводвигуна найчастіше використовують саме таку одиницю.

Іноді доводиться шукати компроміс між цими двома характеристиками, оскільки, якщо ми хочемо надійний серводвигун, що втримує велику вагу, то ми маємо бути готові, що ця потужна установка буде повільно повертатися. А якщо ми хочемо дуже швидкий привід, його відносно легко вивести з положення рівноваги. Під час використання того самого двигуна баланс визначає конфігурація шестерень у редукторі.

Матеріали редукторів серводвигунів

У більшості серводвигунів сполучною ланкою між валом та зовнішніми елементами є редуктор, тому дуже важливо, з якого матеріалу зроблені шестерні редуктора. Найбільш доступних варіантів два: пластмасові (рис. 2.39, а.) або металеві (рис. 2.39, б). У дорожчих моделях можна знайти елементи з карбону та навіть титану.

Пластмасові варіанти, звичайно, дешевші, простіші у виробництві і часто використовуються у недорогих моделях серводвигунів. Для навчальних проєктів, коли серводвигун виконує кілька простих рухів, це не критично. Але у серйозних проєктах використання пластмаси неможливе через дуже швидке зношення таких шестерень під навантаженням.

Металеві шестерні надійніші, але це, безумовно, позначається як на ціні, так і на вазі моделі. Економні виробники можуть зробити частину деталей пластмасовими, а частину металевими, це також потрібно мати на увазі. І, звичайно, що у найдешевших моделях навіть наявність металевої шестерні не є гарантією якості.

а) б)

Рисунок 2.39 – Шестерні: а) пластмасові; б) металеві

Титанові або карбонові шестерні – найкращий варіант, якщо Ви не обмежені бюджетом. Легкі та надійні, такі серводвигуни активно використовуються для створення моделей автомобілів, дронів та літаків.

Переваги серводвигунів

Широке використання серводвигунів пов'язане з тим, що вони характеризуються стабільною роботою, високою стійкістю до перешкод, малими габаритами та широким діапазоном контролю швидкості. Важливими особливостями серводвигунів є здатність збільшувати потужність та забезпечувати зворотний інформаційний зв'язок. З цього випливає, що при прямому напрямку контур є передавачем енергії, а при зворотному – передавачем інформації, яка використовується для покращення точності керування.

Відмінності серво- та звичайного двигунів

Вмикаючи або вимикаючи звичайний електричний двигун, ми можемо сформувати обертальний рух та змусити рухатись колеса або інші предмети, прикріплені до вала. Цей рух буде безперервним, але щоб зрозуміти, на який кут повернувся вал, чи скільки обертів він зробив, потрібно встановлювати додаткові зовнішні елементи – енкодери. Серводвигун вже містить все необхідне для отримання інформації про поточні параметри обертання і може самостійно вимикатися, коли вал повернеться на потрібний кут.

Відмінності серво- та крокового двигунів

Важливою відмінністю серводвигуна від крокового двигуна є можливість працювати з великими прискореннями та при змінному навантаженні. Також серводвигуни мають більш високу потужність. Крокові двигуни не мають зворотного зв'язку, тому може спостерігатися ефект втрати кроків, у серводвигуна втрати кроків унеможливлені, оскільки всі порушення будуть зафіксовані та виправлені. Однак серводвигуни дорожчі, ніж крокові, мають складнішу систему підключення та керування і вимагають кваліфікованішого обслуговування. Важливо відзначити, що крокові двигуни та серводвигуни не є безпосередніми конкурентами – кожен із них має свою певну сферу застосування.

Керування серводвигуном

Вирішальне значення під час управління серводвигунами виконує сигнал керування [53]. Він являє собою імпульси постійної частоти та змінної ширини. Довжина імпульсу – це один із найважливіших параметрів, що визначає положення серводвигуна. Цю довжину можна встановити у програмі вручну шляхом підбору через кут, або використовуючи команди бібліотеки. Для кожної марки пристрою довжина імпульсу може бути різною.

Коли сигнал потрапляє до схеми керування, генератор подає свій імпульс, тривалість якого визначається за допомогою потенціометра. В іншій частині схеми відбувається порівняння тривалості поданого сигналу та си-

гналу з генератора. Якщо ці сигнали різні за тривалістю, вмикається електродвигун, напрямок обертання якого визначається тим, який з імпульсів коротший. У випадку рівності довжини імпульсів двигун зупиняється.

Стандартна частота, з якою подаються імпульси, дорівнює 50 Гц, тобто 1 імпульс за 20 мілісекунд (рис. 2.40). За таких значень тривалість становить 1520 мікросекунд і серводвигун займає середнє положення (рис. 2.41).

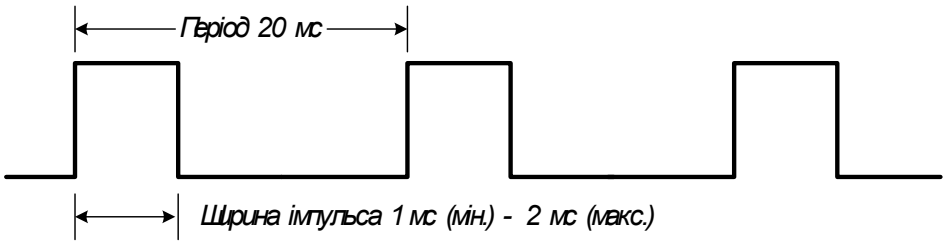


Рисунок 2.40 – Імпульси керування серводвигуном стандартної частоти

Зміна довжини імпульсу приводить до повороту серводвигуна, а саме: зі збільшенням тривалості поворот здійснюється за годинниковою стрілкою, при зменшенні – проти. Є межі тривалості, у бібліотеці Servo на 0° встановлено значення імпульсу 544 мкс (нижня межа), а на 180° – 2400 мкс (верхня).

Важливо враховувати, що на конкретному пристрої налаштування можуть відрізнятися від загальноприйнятих значень. У деяких пристроях середнє положення та ширина імпульсу може дорівнювати 760 мкс. Усі прийняті значення також можуть незначно відрізнятися через похибку, яка може бути допущена під час виробництва пристрою.

Спосіб керування приводом часто помилково називають PWM/ШІМ [54]. Але це не зовсім коректно. Управління безпосередньо залежить саме від довжини імпульсу, їх частота не така важлива. Коректна робота буде забезпечена і на частоті 40 Гц, і на частоті 60 Гц, вклад внесе лише суттєве зменшення або збільшення частоти. При різкому спаді частоти серводвигун почне працювати ривками, а при її збільшенні вище 100 Гц пристрій може перегрітися.

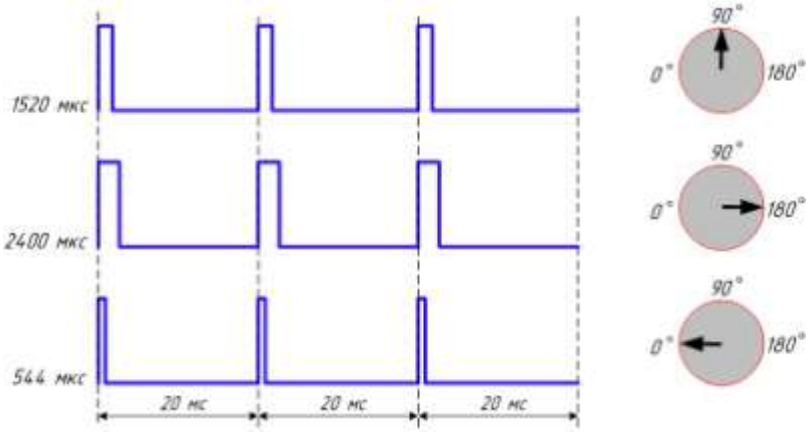


Рисунок 2.41 – Деякі значення тривалостей імпульсів
та відповідних кутів повороту севродвигуна

За внутрішнім інтерфейсом можна виділити аналогові та цифрові серводвигуни. Зовнішніх відмінностей між ними немає – усі відмінності лише у внутрішній електроніці. Аналоговий серводвигун усередині містить спеціальну мікросхему, а цифровий – мікропроцесор, який приймає та аналізує імпульси.

Під час отримання сигналу аналоговий серводвигун приймає рішення, змінити чи не змінити положення, й, за необхідності, подає на двигун сигнал із частотою 50 Гц. За час реакції (20 мс) можуть відбутися зовнішні дії, що змінять положення серводвигуна, і пристрій не встигне зреагувати (рис. 2.42).

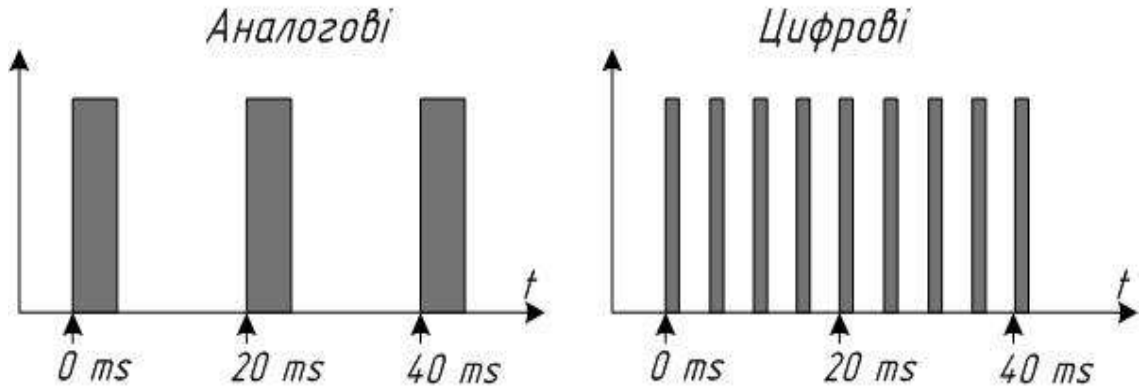


Рисунок 2.42 – Імпульси керування аналоговим
та цифровим серводвигуном

Цифровий серводвигун використовує процесор, який подає та обробляє сигнали з більшою частотою – від 200 Гц, тому він може швидше відреагувати на зовнішні дії, швидше розвивати потрібну швидкість та крутний момент.

Отже, цифровий серводвигун здатний краще утримувати задане положення. При цьому для роботи цифрових серводвигунів потрібно більше електроенергії, що підвищує їхню вартість. Великий внесок у ціну робить і складність їхнього виробництва. Висока вартість – єдиний недолік цифрових серводвигунів, у технічному плані вони наразі набагато кращі за аналогові.

Програмування керування серводвигуном SG 90 робота mBot у mBlock

Запустіть програму mBlock і перевірте налаштування. Використовуйте розташовані справа-зверху в палітрі скриптів (Scripts) блоки з розділів «Контроль» (Control) і «Робот» (Robots) і, перетягуючи їх на робоче поле у центрі mBlock, побудуйте програму керування серводвигуном.

Ми можемо налаштувати серводвигун так, щоб він повернувся під певним кутом. Для цього будуємо програму відповідно до рис. 2.43.

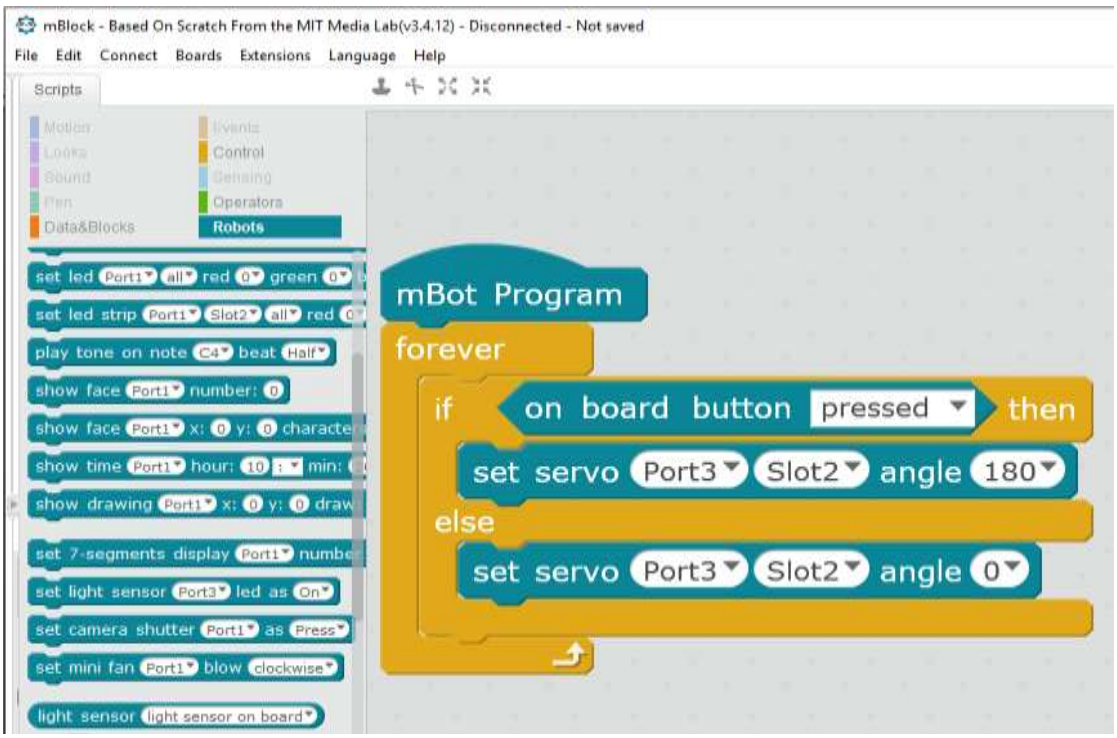


Рисунок 2.43 – Програма керування серводвигуном SG 90 у mBlock

Далі детально розглянемо цю програму.

1. Блок «`mBot Program`» – заголовок програми для робота mBot.

2. Блок «`forever`» – це нескінченний цикл, усередині якого всі блоки з командами виконуються нескінченну кількість разів, поки ввімкнене живлення р*о*бота.

3. Блок «`if… then…else`» – умовний оператор, що перевіряє чи досягнено істинність умови: натиснена кнопка на платі mCore «`on board button pressed`». Якщо умова істинна (тобто кнопка натиснена) – виконується блок 4. У протилежному вмпадку виконується блок 5.

4. Блок «`set servo Port3 Slot2 angle 180`» – вмикає обертання серводвигуна на кут 180 градусів. Це максимальний кут повороту серводвигуна SG 90.

5. Блок «`set servo Port3 Slot2 angle 0`» – вмикає обертання серводвигуна на кут 0 градусів. Це мінімальний кут повороту серводвигуна SG 90.

6. Після цього циклу «`forever`» повторює всі команди, починаючи з блока «`if… then…else`» у пункті 3, і так – нескінченну кількість разів.

Збережіть створену програму обравши у верхньому меню File→Save Project As (див. рис. 2.26).

Завантаження програми у пам’ять робота з mBlock

Перед завантаженням програми у пам’ять mBot перевірте з’єднання роботи з персональним комп’ютером і ввімкніть живлення робота.

Натискайте кнопку «Upload to Arduino» і зачекайте, поки програма не завантажиться у пам’ять мікроконтролера робота mBot. Після закінчення з’явиться повідомлення про успішне завантаження «Upload Finish» (див. рис. 2.27).

MBot відразу розташує вихідний вал серводвигуна у положення, що відповідає відмітці у 0 градусів і очікуватиме допоки не буде натиснено кнопку на платі.

Програмування керування серводвигуном SG 90 робота mBot у середовищі Arduino IDE

Запустіть середовище Arduino IDE та наберіть скетч, наведений у прикладі 2.5.

```
#include <MeMCore.h>  //під’єднуємо бібліотеку Makeblock
Servo servo;        // оголошуємо змінну servo типа "servo"
MePort port(3);   //оголошуємо порт mCore для підкл. серво

void setup() {
  servo.attach(port.pin1());// прив’язуємо серво до порту
                            A3 (mCore port3 pin1)
  pinMode(A7, INPUT);   // налаштування порту A7 на вхід
}
void loop() {
  if (analogRead(A7) > 50) {// умова натиснення кнопки
    delay(50);
    servo.write(180); // встановлюємо кут повороту 180°
  } else {
    delay(50);
    servo.write(0);   // встановлюємо кут повороту 0°
  }
}
```

Приклад 2.5 – Програма керування серводвигуном SG 90 робота mBot у середовищі Arduino IDE

Далі детально розглянемо функціонування програми.

Під’єднуємо бібліотеку makeblock для плати mCore директивою

```
#include <MeMCore.h>
```

Оголошуємо змінну `Servo` тип `servo`;

Потім створюємо об’єкт – серводвигун, вказуючи у дужках порт, до якого він під’єднано на платі mCore:

```
MePort port(3);
```

Обов’язкова процедура `void setap()` – виконується один раз під час увімкнення робота або після того, як було натиснено кнопку Reset. У нашому випадку `servo.attach` під’єднує серводвигун до вказаного порту на платі mCore, з якого здійснюється керування приводом.

```
void setup()
{
servo.attach(port.pin1());
}
```

Функція `pinMode()` в Arduino IDE встановлює режим роботи заданого піна як входу чи виходу. Аналоговий пін може бути лише в одному стані – INPUT.

```
pinMode(A7, INPUT);
```

Обов'язкова процедура `void loop()` – це нескінченний цикл, тобто аналог блока «`forever`» в mBlock (Scratch). Вона містить усі команди для руху робота, які розташовані між фігурними дужками { }.

Наступна частина коду програма перевіряє, чи значення на `analogRead(A7)` перевищує показник у 50 одиниць, і якщо так, то здійснюється команда встановити кут 180º на вихідному валу серводвигуна. Інакше кажучи, якщо вираз у круглих дужках істинний, виконуються оператори всередині фігурних дужок. Якщо ні – програма пропускає цей код і виконує код, що знаходиться у фігурних дужках оператора `else`, тобто встановити кут 0º на вихідному валу серводвигуна.

```
if (analogRead(A7) > 100) {
    delay(50);
    servo.write(180);
  } else {
    delay(50);
    servo.write(0);
  }
```

Функція `analogRead()` зчитує значення з зазначеного аналогового входу з 10-бітовим аналого-цифровим перетворювачем (АЦП). Напругу, що подана на аналоговий вхід (зазвичай, від 0 до 5 Вольт) буде перетворено на значення від 0 до 1023, це 1024 кроки з роздільною здатністю 0.0049 Вольт.

Для того, щоб розрізняти натиснена кнопка чи ні, ми умовно приймаємо значення «LOW» у межах від 0 до 100 кроків, тобто від 0 до 0,49 Вольта включно, а значення «HIGH» від 0,49 до 5 Вольт.

Зчитування значення аналогового входу займає приблизно 100 мікросекунд (0.0001 секунд), тобто. максимальна частота зчитування приблизно 10000 разів на секунду.

Після досягнення останньої команди підпрограма `void loop()` почне виконувати повторно команди, які знаходяться між її фігурними дужками `{}`, починаючи з руху робота вперед і так нескінченну кількість разів.

Завантаження програми у пам'ять мікроконтролера робота mBot **із Arduino IDE**

Перед завантаженням програми у пам'ять mBot перевірте з'єднання робота з ПК та ввімкніть живлення робота. Далі клікніть по піктограмці «Вивантажити» (див. рис. 2.29). У випадку успішного завантаження програми з Arduino IDE у пам'ять мікроконтролера робота mBot Ви будете спостерігати повідомлення «Вивантаження виконано» (див. рис. 2.30). Як-

що ж Ви допустили якусь помилку, завантаження програми буде перервано. У цьому випадку прочитайте повідомлення про помилку, виправте її та повторіть завантаження (див. рис. 2.31).

Після успішного завантаження mBot відразу розпочне реагувати на натискання кнопки розміщеної на платі mCore.

2.6 Керування роботом mBot за допомогою IR пульта дистанційного керування

Більшість сучасної побутової електронної апаратури має пульт дистанційного керування, що використовує інфрачервоне (ІЧ) випромінювання (англ. IR – Infrared) як спосіб передачі інформації. Такий ІЧ пульт (рис. 2.44) входить і до складу плати mCore робота mBot [55].

Рисунок 2.44 – ІЧ пульт дистанційного керування mBot

Розглянемо основи роботи ІЧ пульта дистанційного керування.

Інфрачервоне, або теплове випромінювання – це електромагнітне випромінювання, яке випромінює будь-яке нагріте до певної температури тіло. ІЧ діапазон лежить у найближчій до видимого світла області спектра, у його довгохвильовій частині, і займає область приблизно від 750 нм до 1000 мкм.

Оптичні властивості речовин в інфрачервоному випромінюванні відрізняються від їх властивостей у світлі. Наприклад, деяке скло непрозоре для інфрачервоних променів, а парафін, на відміну від видимого світла, прозорий для ІЧ випромінювання та використовується для виготовлення ІЧ лінз. Для його реєстрації використовують теплові та фотоелектричні приймачі та спеціальні фотоматеріали. Джерелом ІЧ променів, крім нагрітих тіл, найчастіше використовуються твердотільні випромінювачі: інфрачервоні світлодіоди, ІЧ лазери. Деякі особливості інфрачервоного випромінювання роблять його зручним для застосування у пристроях передачі даних:

- ІЧ твердотільні випромінювачі (ІЧ світлодіоди) компактні, практично безінерційні;
- ІЧ промені не відволікають увагу людини через свою невидимість;
- Незважаючи на поширеність ІЧ променів та високий рівень «фону», джерел імпульсних завад в ІЧ спектрі мало;
- ІЧ випромінювання низької потужності не впливає на стан здоров’я людини;
- ІЧ промені добре відбиваються від більшості матеріалів (стін, меблів тощо);
- ІЧ випромінювання не проникає крізь стіни та не заважає роботі інших аналогічних пристроїв.

Усе це дозволяє успішно використовувати спосіб ІЧ керування у багатьох пристроях.

Пульт ІЧ керування під час натискання кнопки випромінює кодоване посилання, а приймач, встановлений у керованому пристрої, приймає його та виконує передбачені дії. Для того, щоб передати логічну послідовність, пульт формує імпульсний пакет ІЧ променів, інформація в якому модулюється або кодується тривалістю або фазою складових пакетів імпульсів.

Сучасні інфрачервоні світлодіоди мають ефективність 100–200 мВт випромінюваної енергії за струму 50 мА. Допустимий середній струм не має перевищувати 10–20 мА. Спектр світлодіодів для ІЧ пультів більшості побутової апаратури має максимум в області 940 нм.

Розглянемо принцип роботи ІЧ-приймача.

ІЧ-приймач на платі mCore здатний приймати та обробляти інфрачервоний сигнал у вигляді імпульсів заданої тривалості та частоти. Зазвичай ІЧ-приймач конструктивно виконаний у чорному корпусі, має три виводи. Його назва може звучати як TSOP – Temic Semiconductors Opto electronics Photo modules.

Спрощена схема TSOP-приймача наведена на рисунку 2.45. Як вихідний елемент усередині TSOP використовується звичайний NPN транзистор. У неактивному стані транзистор закритий, і на виході (вивід V_o) є слабкий рівень високої напруги (логічна «1»). Із появою у чутливій зоні TSOP інфрачервоного випромінювання з «основною» частотою цей транзистор відкривається і вихід приймає низький рівень сигналу (логічний «0»).

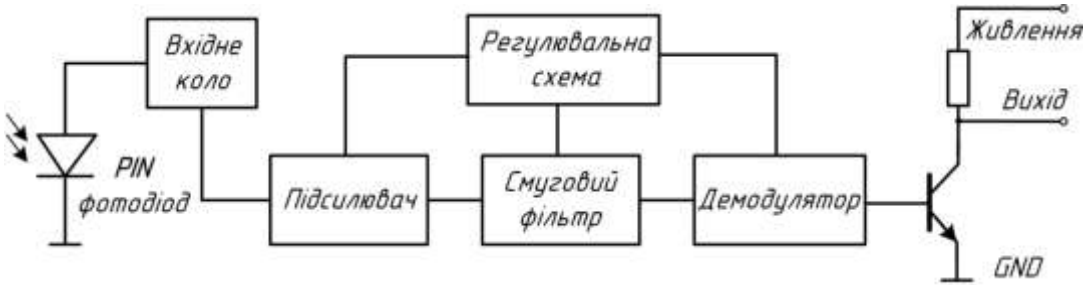


Рисунок 2.45 – Структурна схема TSOP приймача

«Головна» частота – це частота імпульсів інфрачервоного випромінювання (світла), яку відфільтровує внутрішній демодулятор TSOP. Ця частота, зазвичай, дорівнює 36, 38, 40 кГц, але може бути й інша.

У нашому випадку на платі mCore встановлено ІЧ приймач TSOP4838, де число 48 характеризує його типорозмір, а число 38 – частоту в кілогерцах (рис. 2.46), у нашому випадку це 38 кГц [56, 57].

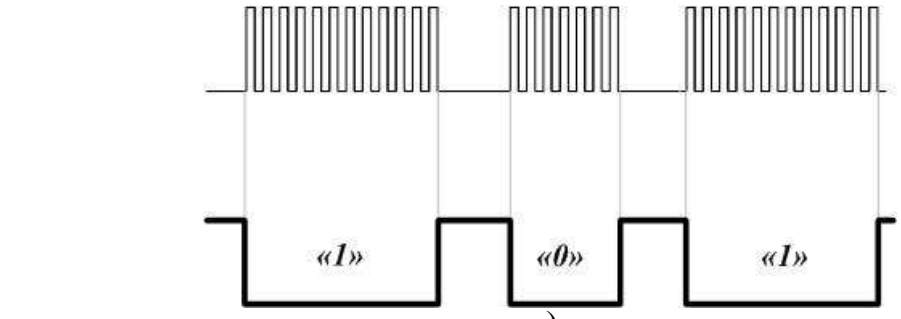
TSOP4838 TSOP1738 SPS440-38

Рисунок 2.46 – ІЧ-приймачі TSOP4838, TSOP1638 та SPS440-38

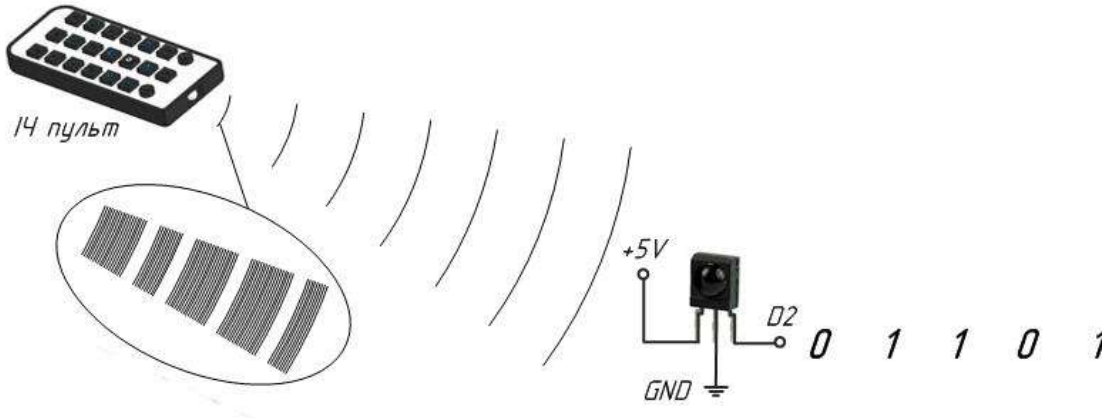
Для підвищення завадостійкості ІЧ каналу зв’язку застосовується модульована передача ІЧ світла [58]. Характеристики модуляції для завадозахисної передачі наведені у технічній специфікації (англ. Datasheet) на конкретний TSOP-приймач. Але у більшості випадків достатньо дотримуватися простих правил:

- мінімальна кількість імпульсів у пачці – 15;
- максимальна кількість імпульсів у пачці – 50;
- мінімальний час між пачками – 15·T, де T – період «основної» частоти TSOP-приймача;
- частота імпульсів у пачці має відповідати основній частоті TSOP-приймача;
- світлодіод має бути з довжиною хвилі 950 nm.

Регулюючи у деяких межах довжину пачки імпульсів, можна передавати бінарні сигнали. Довгий імпульс на виході приймача TSOP може означати «одиницю», а короткий – «нуль» (рис. 2.47). Таким чином, при дотриманні правил модуляції дальність передачі цифрових сигналів у межах прямої видимості між світлодіодом і TSOP-приймачем може досягати 10–20 метрів. Швидкість передачі невелика, близько 1200 біт за секунду, залежно від використовуваного TSOP-приймача.



а)



б)

Рисунок 2.47 – а) принцип передачі імпульсів;
б) принцип роботи ІЧ приймача

Програмуємо керування роботом mBot за допомогою IR пульта дистанційного керування у середовищі mBlock

Запустіть програму mBlock і перевірте налаштування. Використовуйте розташовані справа-зверху в палітрі скриптів (Scripts) блоки з розділів «Контроль» (Control) і «Робот» (Robots) і, перетягуючи їх на робоче поле у центрі mBlock, побудуйте програму, наведену на рис. 2.48. Проаналізуємо функціонування програми.

1. Блок «`mBot Program`» – заголовок програми для роботи mBot.

2. Блок «`forever`» – це нескінченний цикл, усередині якого всі блоки з командами виконуються нескінченну кількість разів, допоки увімкнене живлення робота.

3. Блок «`if…then…else`» – умовний оператор, перевіряє чи досягнено істинність умови: натиснута кнопка на пульті керування платою mCore «↑». Якщо умова істинна, тобто кнопка натиснута, – виконується блок 4, якщо ж ні –виконується блок 5.

4. Блок «`run forward at speed 255`» – вмикає обертання двигунів, забезпечуючи повертання ліворуч зі швидкістю 255 одиниць. Це максимальна швидкість обертання двигунів.

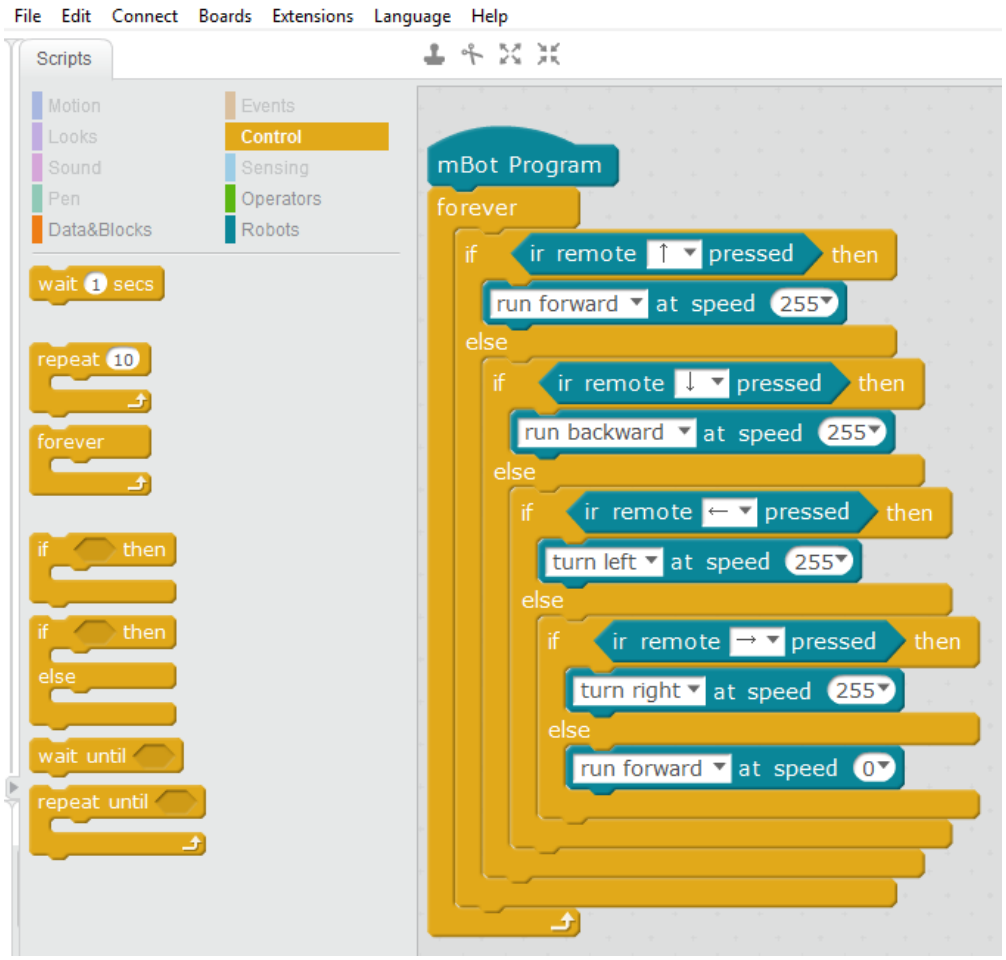


Рисунок 2.48 – Програма керування mBot
за допомогою IR пульта у середовищі mBlock

5. Блок «`if…then…else`» – перевіряє чи натиснута кнопка на пульті керування платою mCore «↓». Якщо так – виконується блок 6, а якщо ні – блок 7.

6. Блок «`run backward at speed 255`» – вмикає обертання двигунів, забезпечуючи рух назад зі швидкістю 255 одиниць.

7. Блок «`if…then…else`» – перевіряє чи натиснута кнопка на пульті керування платою mCore «←». Якщо так – виконується блок 8, а якщо ні – блок 9.

8. Блок «`run left at speed 255`» – вмикає обертання двигунів, забезпечуючи рух ліворуч зі швидкістю 255 одиниць.

9. Блок «`if…then…else`» – перевіряє чи натиснута кнопка на пульті керування платою mCore «→». Якщо так – виконується блок 10, а якщо ні – блок 11.

10. Блок «`run right at speed 255`» – вмикає обертання двигунів, забезпечуючи рух праворуч зі швидкістю 255 одиниць.

11. Блок «`run forward at speed 0`» – вимикає обертання двигунів.

Збережіть створену програму (див. рис. 2.26).

Завантаження програми у пам'ять робота з mBlock

Перед завантаженням програми у пам'ять mBot перевірте з'єднання робота з ПК та увімкніть живлення робота. Коли будете готові до завантаження, натисніть кнопку «Upload to Arduino» і зачекайте допоки програма не завантажиться в mBot. Після закінчення з'явиться повідомлення про успішне завантаження «Upload Finish» (див. рис. 2.27).

Під час натисканні кнопок, що задіяні у коді програми, на пульті дистанційного керування mBot відразу розпочне рухатись у ту чи іншу сторону, залежно від прописаної у коді команди.

Програмуємо керування роботом mBot за допомогою IR пульта дистанційного керування у середовищі Arduino IDE

Потрібно вказати, що ми можемо запрограмувати керування роботом mBot за допомогою ІЧ пульта будь-якого побутового пристрою. Для цього нам спочатку потрібно дізнатись коди тієї чи іншої кнопки на такому пульті у десятковому, бінарному або HEX форматі.

Для цього потрібно запустити середовище Arduino IDE та ввести програму, наведену у прикладі 2.6.

```
#include <IRremote.h>  //підєднуємо бібліотеку IRremote.h
IRrecv irrecv(2);      // вказуємо пін, на який під'єднано
                          IR приймач
decode_results results; // присвоюємо змінній results
                        значення сигналу отримане від пульта
void setup() {
  Serial.begin(9600);           // Задаємо швидкість обміну
                                   даними по COM порту
  irrecv.enableIRIn();          // запускаємо IR приймач
}
void loop() {
  if ( irrecv.decode( &results )) { // Якщо дані прийшли
```

```
    Serial.println(results.value);  // Виводимо в COM порт
                                       отримані дані
    Serial.println(results.value, HEX); // Виводимо в COM
                              порт отримані дані у HEX форматі
Serial.println(results.value, BIN); // Виводимо в COM порт
                             отримані дані у бінарному форматі
    irrecv.resume();           // приймаємо наступну команду
  }
}
```

Приклад 2.6 – Програма визначення кодів кнопок на довільному IR пульті керування

Визначившись із кнопками на пульті, які нам зручні для використання, та за допомогою програми, наведеної у прикладі 2.6, натискаємо відповідну кнопку на пульті та отримуємо її значення у тому чи іншому форматі даних у COM порті.

У нашому випадку:

- рух вперед – 16712445;
- рух назад – 16750695;
- рух ліворуч – 16769055;
- рух праворуч – 16748655;

За замовчуванням використовуємо десятковий формат даних.

Після цього ми за допомогою середовища Arduino IDE завантажуємо в mBot програму (приклад 2.7), яка за допомогою нашого ІЧ пульта буде керувати рухом робота mBot,

```
#include <IRremote.h>     //підєднуємо бібліотеку IRremote.h
IRrecv irrecv(2);         //вказуємо пін, на який підєднано
                            IR приймач
decode_results results;  // присвоюємо змінній results
                         значення сигналу отримане від пульта
uint8_t Speed_motor = 255;       // Швидкість двигунів

void setup() {
  Serial.begin(9600);         // Задаємо швидкість обміну
                                 даними по COM порту
  irrecv.enableIRIn();        // запускаємо IR приймач
  pinMode(4, OUTPUT);         // налаштування порта 4 на вихід
  pinMode(7, OUTPUT);         // налаштування порта 7 на вихід
  pinMode(5, OUTPUT);         // налаштування порта 5 на вихід
  pinMode(6, OUTPUT);         // налаштування порта 6 на вихід
}

void loop() {
  if ( irrecv.decode( &results )) {  // якщо дані прийшли
   if (results.value == 16712445) {
      // Рухатись вперед
      digitalWrite(4, HIGH);   // Обертання правого двигуна
                                  за годинниковою стрілкою
      analogWrite (5, Speed_motor);  // швидкість обертання
                                     лівого двигуна (ШІМ=255)
```

```
      digitalWrite(7, LOW);     // Обертання лівого двигуна
                                проти годинниковою стрілкою
      analogWrite (6, Speed_motor);  // швидкість обертання
                                  правого двигуна (ШІМ=255)
    }
    if (results.value == 16750695) {
     // Рухатись назад
      digitalWrite(4, LOW);     // Обертання правого двигуна
                                 проти годинникової стрілки
      analogWrite (5, Speed_motor); //швидкість обертання
                                  лівого двигуна (ШІМ=127)
      digitalWrite(7, HIGH);     // Обертання лівого двигуна
                                  за годинниковою стрілкою
      analogWrite (6, Speed_motor);   // швидкість обертання
                                  правого двигуна (ШІМ=127)
    }
    if (results.value == 16769055) {
      // Рухатись ліворуч
      digitalWrite(4, LOW);     // Обертання правого двигуна
                                проти годинниковою стрілкою
      analogWrite (5, Speed_motor/2); //швидкість обертання
                                  лівого двигуна (ШІМ=127)
      digitalWrite(7, LOW);      // Обертання лівого двигуна
                                  за годинниковою стрілкою
      analogWrite (6, Speed_motor/2); // швидкість обертання
                                  правого двигуна (ШІМ=127)
    }
    if (results.value == 16748655) {
      // Рухатись праворуч
      digitalWrite(4, HIGH);    // Обертання правого двигуна
                                 проти годинникової стрілки
      analogWrite (5, Speed_motor/2); //швидкість обертання
                                  лівого двигуна (ШІМ=127)
      digitalWrite(7, HIGH);     // Обертання лівого двигуна
                                  за годинниковою стрілкою
      analogWrite (6, Speed_motor/2); // швидкість обертання
                                  правого двигуна (ШІМ=127)
    }

      else {
      analogWrite (5, 0); // Зупинка правого двигуна (ШІМ=0)
      analogWrite (6, 0); // Зупинка лівого двигуна (ШІМ=0)
    }
  }

  irrecv.resume();          // приймаємо наступну команду
}
```

Приклад 2.7 – Програма керування роботом mBot
за допомогою IR пульта у середовищі Arduino IDE

Не забудьте зберегти цю програму.

Завантаження програми у пам’ять мікроконтролера робота mBot із Arduino IDE

Перед завантаженням програми у пам’ять mBot перевірте з’єднання робота з ПК та ввімкніть живлення робота. Далі клікніть по піктограмці «Вивантажити» (див. рис. 2.29). У випадку успішного завантаження програми з Arduino IDE у пам’ять мікроконтролера робота mBot Ви будете спостерігати повідомлення «Вивантаження виконано» (див. рис. 2.30). Якщо ж Ви допустили якусь помилку, завантаження програми буде перервано. У цьому випадку прочитайте повідомлення про помилку, виправте її та повторіть завантаження (див. рис. 2.31).

Після успішного завантаження програми при натисненні на відповідну запрограмовану кнопку на пульті mBot відразу почне рухатись згідно з натиснутою кнопкою: вперед, назад, вліво чи вправо.

2.7 Застосування світлодіодної матриці Me LED MATRIX (8×16) у роботі mBot

Світлодіодна матриця Me LED MATRIX (8×16) містить 128 світлодіодів синього кольору, розташованих у 8 рядків та 16 стовпців (рис. 2.49). Отримавши дані з основної плати, можна керувати показом цифр, літер або символів на світлодіодній матриці. На роз’ємі є синій ідентифікатор. Це означає, що даний модуль можна під’єднати до порту з синім ID на Makeblock mCore [59].

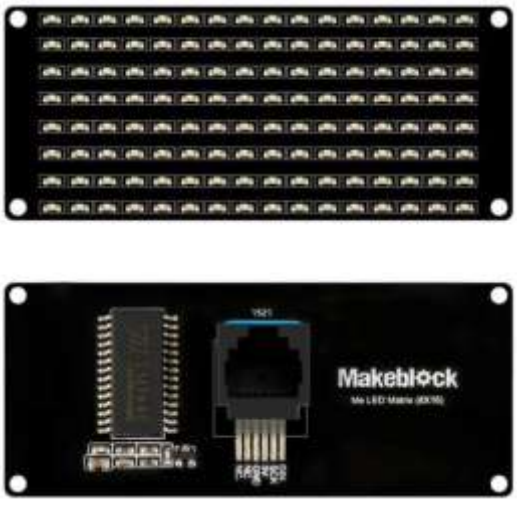


Рисунок 2.49 – Вигляд світлодіодної матриці Me LED MATRIX (8×16)

Технічні характеристики світлодіодної матриці Me LED MATRIX (8×16):

- Робоча напруга: 5 В (постійної напруги)
- Режим сигналу: зв’язок I^2C
- Розмір модуля: 73 × 31 × 15 мм (Д × Ш × В)

Порт Me LED Matrix (8 × 16) має чотири контакти з функціями, наведеними у таблиці 2.2.

Таблиця 2.2 – Функції контактів світлодіодної матриці Me LED Matrix (8 × 16)

№	Пін	Функція
1	DIN	Data
2	SCK	Clock
3	5 V	Power supply
4	GND	Grounding

Підключення здійснюється за допомогою роз'єму RJ25. Оскільки порт LED Matrix (8 × 16) має Blue ID, Вам потрібно підключити порт із синім ID на Makeblock. Ви можете підключити матрицю до будь-якого з портів № 1, 2, 3 чи 4 плати mCore, як наведено на рис. 2.50.

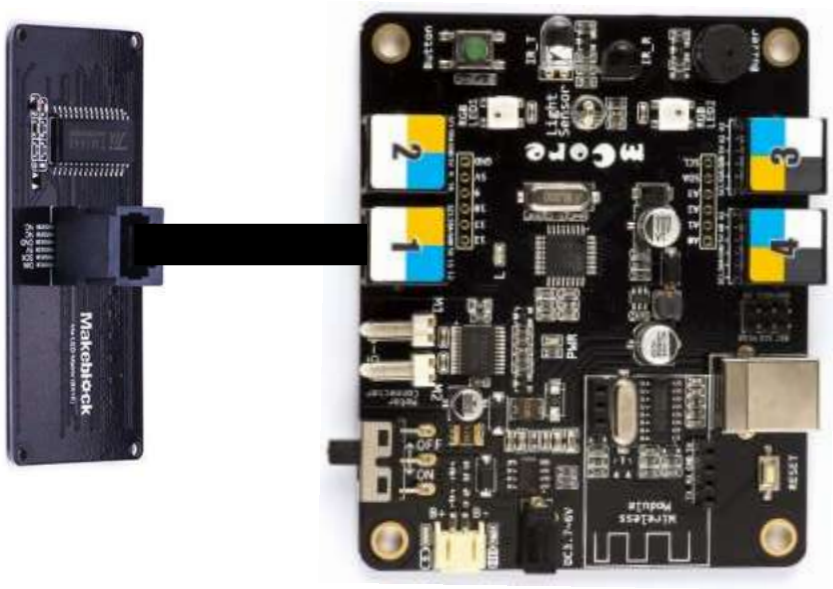


Рисунок 2.50 – Під'єднання світлодіодної матриці Me LED MATRIX (8×16) до першого порта плати mCore

Під час використання середовища Arduino IDE для написання коду програми, потрібно викликати бібліотеку Makeblock-Library-master для керування цією світлодіодною матрицею.

Далі розглянемо, як влаштована матриця Me LED MATRIX (8×16). Як згадувалось раніше, ця матриця має 8 рядків та 16 стовпців. Кожен світлодіод індексується від 0 до 7 по вертикалі та від 0 до 15 по горизонталі. Для кращого розуміння наведемо рисунок 2.51

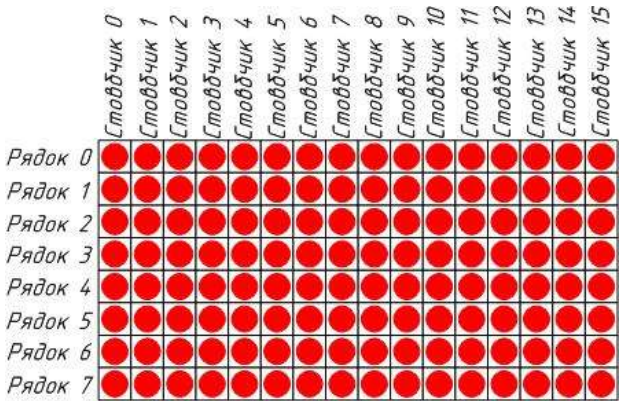


Рисунок 2.51 – Розташування світлодіодів
світлодіодної матриці Me LED MATRIX (8×16)

Якщо Вам потрібно щось відобразити на матриці, потрібно ввімкнути певні світлодіоди. Світлодіоди вмикаються за координатами по горизонталі та по вертикалі.

Наприклад, якщо необхідно відобразити смайлик, Вам потрібно засвітити, наприклад, світлодіоди, як наведено на рис. 2.52.

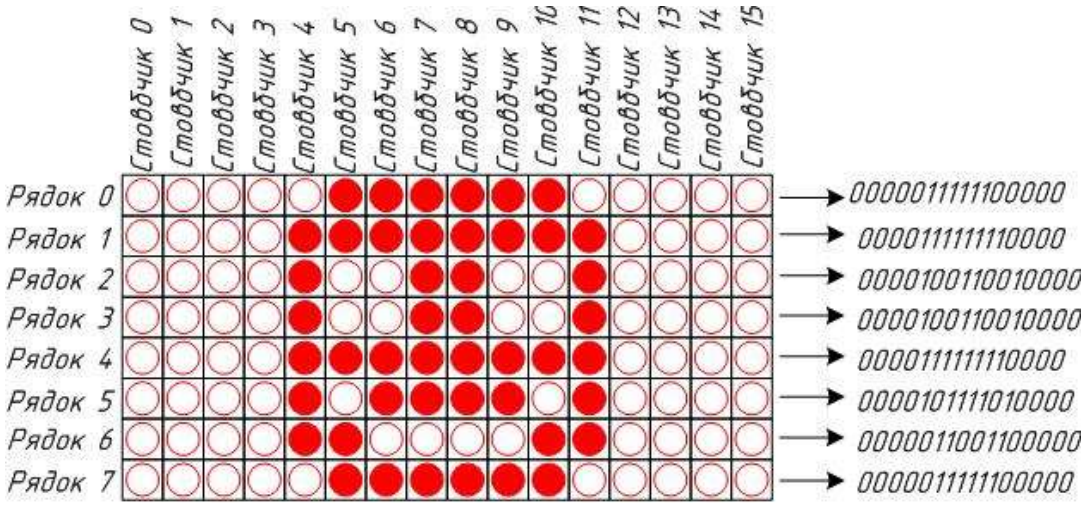


Рисунок 2.52 – Відображення смайлика
на світлодіодній матриці Me LED MATRIX (8×16)

Програмуємо виведення на модуль Me LED MATRIX анімації у середовищі mBlock

Запустіть програму mBlock і перевірте налаштування. Використовуйте розташовані справа-зверху у палітрі скриптів (Scripts) блоки з розділів «Контроль» (Control) і «Робот» (Robots) і, перетягуючи їх на робоче поле у центрі mBlock, побудуйте програму, наведену на рис. 2.53.

Після чого натисніть на пусте поле «draw» блока «show drawing» (рис. 2.54). З’явиться робоче поле «Face Panel», в якому створюємо будь-яке зображення розмірами 8×16 пікселів та зберігаємо у середовищі розробки mBlock, натиснувши на кнопку «Add To Favourete» (рис. 2.55).

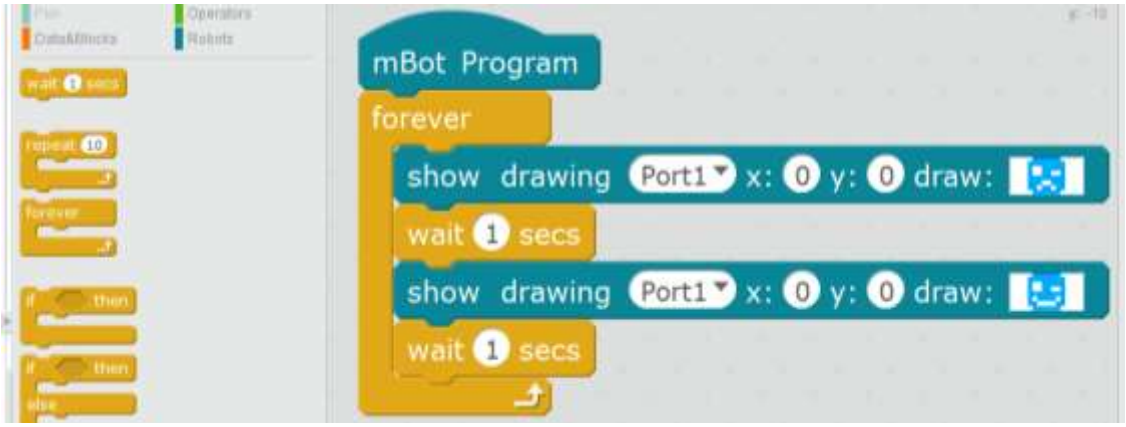


Рисунок 2.53 – Програма виведення на модуль
Me LED MATRIX анімації у середовищі mBlock

Рисунок 2.54 – Поле «draw» блока «show drawing»

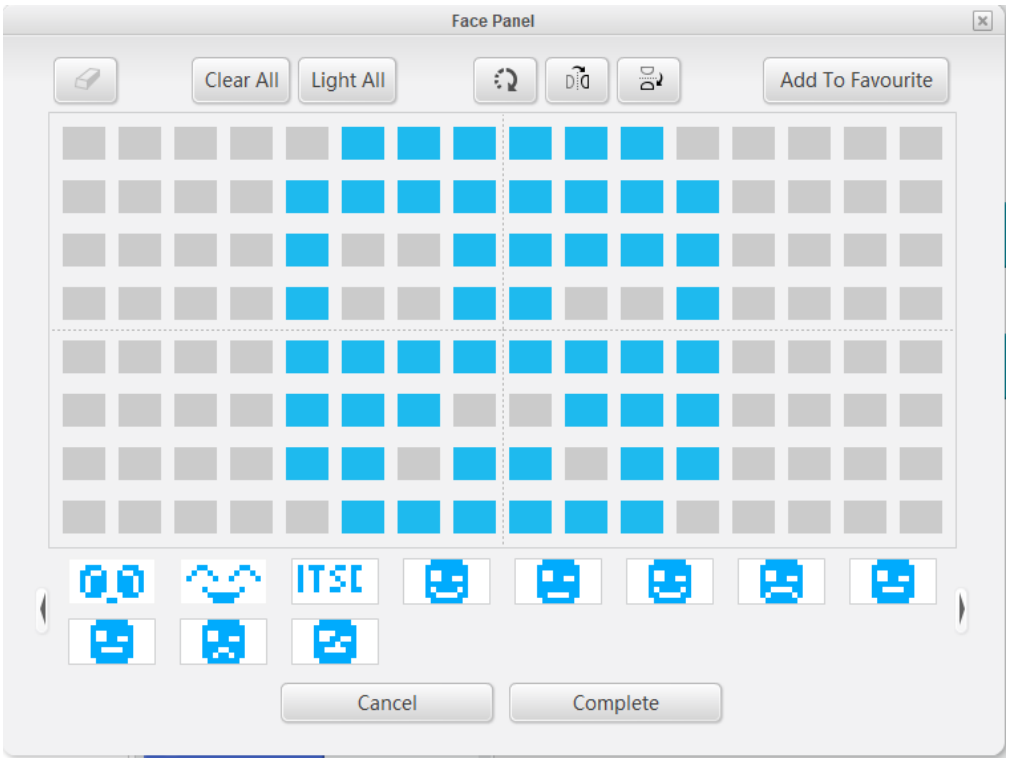


Рисунок 2.55 – Вікно робочого поля «Face Panel»

Подальше натискання на кнопку «Complite» додасть створену чи обрану піктограму до блока «show drawing».

Далі детально розглянемо, як ця програма працює.

1. Блок «mBot Program» – заголовок програми для роботи mBot.

2. Блок «forever» – це нескінченний цикл, усередині якого всі блоки з командами виконуються нескінченну кількість разів, допоки ввімкнено живлення робота.

3. Блок «show drawing» – відображає обрану у полі «draw» піктограму за заданими координатами x = 0, y = 0, у нашому випадку з початку системи координат світлодіодної матриці.

4. Блок «wait 1 secs» потрібен для того, щоб зображення протягом 1 секунди відображалося на матриці.

5. Далі програма виводить іншу піктограму, що задана у блоці «show drawing» аналогічно пункту 3.

6. Після цього циклу «forever» повторює усі команди, починаючи з блока «show drawing» у пункті 3, і так нескінченну кількість разів, у результаті чого у нас з'являється анімація «робот, що моргає».

Не забудьте зберегти створену програму (див. рис. 2.26).

Завантаження програми у пам'ять робота з mBlock

Перед завантаженням програми в пам'яті mBot перевірте з'єднання роботи з персональним комп'ютером і ввімкніть живлення робота.

Коли будете готові до завантаження, натисніть кнопку «Upload to Arduino» і зачекайте кілька хвилин, поки програма не завантажиться у пам'ять робота mBot. Після закінчення з'явиться повідомлення про успішне завантаження «Upload Finish» (див. рис. 2.27). І на світлодіодному модулі Me LED MATRIX робота mBot відразу розпочнеться анімація.

Програмуємо виведення на світлодіодний модуль Me LED MATRIX анімації у середовищі Arduino IDE з використанням бібліотеки TM16xxMatrix.h

Для програмування модуля Me LED MATRIX робота mBot в Arduino IDE доцільно використовувати бібліотеку TM16xxMatrix.h. Її можна завантажити за посиланням https://github.com/maxint-rd/TM16xx.git.

Після завантаження бібліотеки для свого проєкту, запаковану в ZIP-архів, її можна встановити через середовище Arduino IDE. Для встановлення потрібно перейти в меню Скетч → Підключити бібліотеку → Додати бібліотеку .ZIP і вибрати ZIP-архів на комп'ютері (див. рис. 2.14). Файл буде автоматично розпаковано та переміщено у директорію з бібліотеками.

Далі запустіть середовище Arduino IDE та введіть скетч, наведений у прикладі 2.8.

```
#include <TM1640.h>          //під'єднуємо бібліотеку TM1640
#include <TM16xxMatrix.h>    //під'єднуємо бібліотеку
                               Me LED MATRIX
TM1640 Me_LED_MATRIX  (12, 11); // вказуємо піни для
 під'єднання модуля до плати mCore: DIN=12/MOSI, CLK=11/SCK
```

```
TM16xxMatrix matrix(&Me_LED_MATRIX, 16, 8);  // оголошуємо
 обєкт Me_LED_MATRIX, кількість стовбпців, кількість рядків
char data[8][16] = { // декларуємо масив data розміром 8x16
  { 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0,},
  { 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0,},
  { 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0,},
  { 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0,},
  { 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0,},
  { 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0,},
  { 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0,},
  { 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0,},
};
char data1[8][16] = {//декларуємо масив data1 розміром 8x16
  { 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0,},
  { 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0,},
  { 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0,},
  { 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0,},
  { 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0,},
  { 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0,},
  { 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0,},
  { 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0,},
};
void setup()
{
  Me_LED_MATRIX.clearDisplay();        // стираємо дисплей
}
void loop() {
  for ( int  i = -1; i < 16; i++ ) {    1 // цикл перебору
                                          елементів рядків
    for ( int j = -1; j < 8 ; j++ ) {     // цикл перебору
                                         елементів стовпців
      matrix.setPixel(i, j, data[j][i]);  // засвітити і-й
                                         піксел в j-у рядку
    }
  }
  delay(1000);                              // очікуємо 1с.
  for ( int  i = -1; i < 16; i++ ) {        // цикл перебору
                                           елементів рядків
    for ( int j = -1; j < 8 ; j++ ) {       // цикл перебору
                                          елементів стовпців

      matrix.setPixel(i, j, data1[j][i]);  // засвітити і-й
                                         піксел в j-у рядку
    }
  }
  delay(1000);                               // очікуємо 1с.
}
```

Приклад 2.8 – Програма виведення на світлодіодний модуль Me LED MATRIX анімації у середовищі Arduino IDE з використанням бібліотеки TM16xxMatrix.h

Далі детально проаналізуємо, як ця програма працює.

Під'єднуємо бібліотеку Me LED MATRIX для плати mCore:

```
#include <TM1640.h>
#include <TM16xxMatrix.h>
```

Вказуємо піни для під'єднання модуля до плати mCore: DIN=12/MOSI, CLK=11/SCK:

```
TM1640 Me_LED_MATRIX (12, 11);
```

Потім створюємо об'єкт – світлодіодну матрицю Me_LED_MATRIX та вказуємо її розміри: кількість стовпців, кількість рядків:

```
TM16xxMatrix matrix(&Me_LED_MATRIX, 16, 8);
```

Декларуємо два масиви data та data1 розміром 8×16

```
char data[8][16] = {
  { 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0,},
  { 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0,},
  { 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0,},
  { 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0,},
  { 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0,},
  { 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0,},
  { 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0,},
  { 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0,},
};
char data1[8][16] = {
  { 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0,},
  { 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0,},
  { 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0,},
  { 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0,},
  { 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0,},
  { 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0,},
  { 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0,},
  { 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0,},
};
```

Обов'язкова процедура `void setup()` виконується один раз під час включення робота або після того, як було натиснено кнопку Reset. У даній процедурі ми здійснюємо стирання можливих значень елементів матриці, що залишились у пам'яті мікроконтролера:

```
void setup()
{
Me_LED_MATRIX.clearDisplay();
}
```

Обов'язкова процедура `void loop()` – це нескінченний цикл, тобто, аналог блока «`forever`» у mBlock (Scratch). Вона містить усі команди для руху робота, які розташовані між фігурними дужками `{}`.

Далі програма використовує вкладну конструкцію циклу `for` для перебору та виведення елементів двовимірного масиву, що приймають значен-

ня «false» – 0, або «true» – 1. Перший цикл перебирає рядки масиву `data` (змінна `i`), другий, вкладений – стовпці (змінна `j`) масиву.

```
    for ( int  i = -1; i < 16; i++ ) {
        for ( int j = -1; j < 8 ; j++ ) {
matrix.setPixel(i, j, data[j][i]);
        }
      }
```

Після чого здійснюється виведення значення елементів `i`, `j` масиву `data` на світлодіодну матрицю з тими ж координатами `i`, `j` за допомогою функції

```
matrix.setPixel(i, j, data[j][i]);
```

Аналогічно здійснюється виведення на світлодіодну матрицю другого масиву – `data1`. У результаті чого ми спостерігаємо почергову зміну зображення піктограм масиву `data` та `data1` на світлодіодній матриці Me LED MATRIX з паузою в 1 секунду, яку забезпечує функція

```
delay(1000);
```

Після цього підпрограма `void loop()` почне виконувати повторно команди, які знаходяться між її фігурними дужками {}, починаючи з руху робота вперед, і так нескінченну кількість разів.

Завантаження програми у пам'ять мікроконтролера робота mBot із Arduino IDE

Перед завантаженням програми у пам'ять mBot перевірте з'єднання робота з ПК та ввімкніть живлення робота. Далі клікніть по піктограмці «Вивантаження» (див. рис. 2.29). У випадку успішного завантаження програми з Arduino IDE у пам'ять мікроконтролера робота mBot Ви будете спостерігати повідомлення «Вивантаження виконано» (див. рис. 2.30). Якщо ж Ви допустили якусь помилку, завантаження програми буде перервано. У цьому випадку прочитайте повідомлення про помилку, виправте її та повторіть завантаження (див. рис. 2.31).

Після успішного завантаження програми на світлодіодній матриці Me LED MATRIX робота mBot відразу розпочнеться показ анімації.

2.8 Керування рухом робота mBot за допомогою ультразвукового модуля ULTRASONIC SENSOR

Ультразвуковий далекомір – пристрій, призначений для визначення відстані від сенсора до об'єкта. В основі принципу вимірювання покладено ехолокацію, як у дельфінів або кажанів. Сенсор складається з передавача, що генерує ультразвукові хвилі, приймача, який «слухає» відлуння та певної електронної схеми, призначеної для нормальної роботи модуля [60].

Ультразвуковий далекомір наведено на рис. 2.56, де передавач (transmitter) та приймач (receiver), позначені T і R відповідно.

Характеристики сенсора:

- діапазон вимірювання відстані: 0,03 м–4 м;
- частота ультразвуку: 40 kHz;
- кут огляду: 30°;
- інтерфейс: 2 логічні TTL лінії;
- вихідна інформація: імпульс 0,15–25 mS;
- напруга живлення V_{cc} = 5 В;
- струм споживання в активному режимі 15 мА.

Рисунок 2.56 – Загальний вигляд модуля Ultrasonic Sensor V3.0 від Makeblock

Спрощено принцип роботи даного сенсора можна подати так. Далекомір генерує звукові хвилі на частоті 40 кГц. Після того, як ці хвилі відбиваються від об’єкта і повертаються на приймач, сенсор видає інформацію про час, витрачений на проходження хвилі від сенсора до об’єкта і назад. Наочно цей процес наведено на рисунку 2.57.

Ультразвуковий сигнал поширюється широконаправленою хвилею 30°. Напрямок поширення ультразвукового сигналу з передавача наведено на рисунку 2.58. Найефективніший кут виміру 15°. Сторонні об’єкти, які потрапляють під цей кут вимірювання, можуть «збивати» показання сенсора [61].

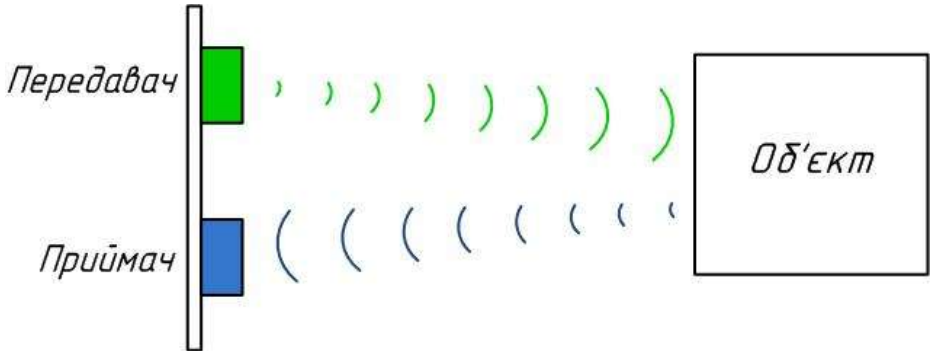


Рисунок 2.57 – Рух ультразвукового сигналу від передавача до приймача

На показники ультразвукових далекомірів не впливають засвічення від сонця або колір об’єктів, як це відбувається з інфрачервоними сенсорами. Ультразвукова хвиля відображатиметься практично від будь-яких поверхонь, навіть прозорих, але можуть виникнути труднощі з визначенням відстані до пухнастих або дрібних предметів. Також на показники впливає кут падіння хвилі. Якщо сенсор спрямований перпендикулярно до об’єкта, то вимірювання будуть найточнішими. Якщо ж кут падіння буде занадто ве-

ликим, то хвиля, відбившись від об’єкта, не потрапить у приймач, що призведе до недостовірного виміру (рис. 2.59).

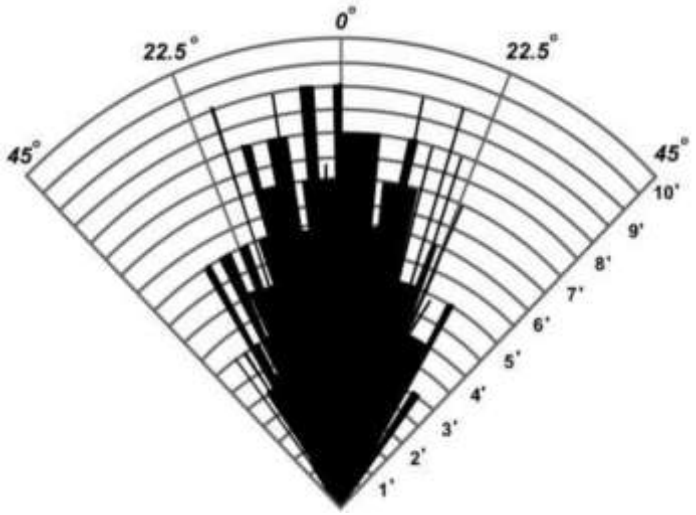


Рисунок 2.58 – Діаграма спрямованості ультразвукової хвилі

Далі перейдемо до розрахунку відстані від датчика до об’єкта. Сам датчик нічого не розраховує самостійно, лише видає імпульс певної тривалості. Усі розрахунки потрібно проводити за допомогою мікроконтролера.

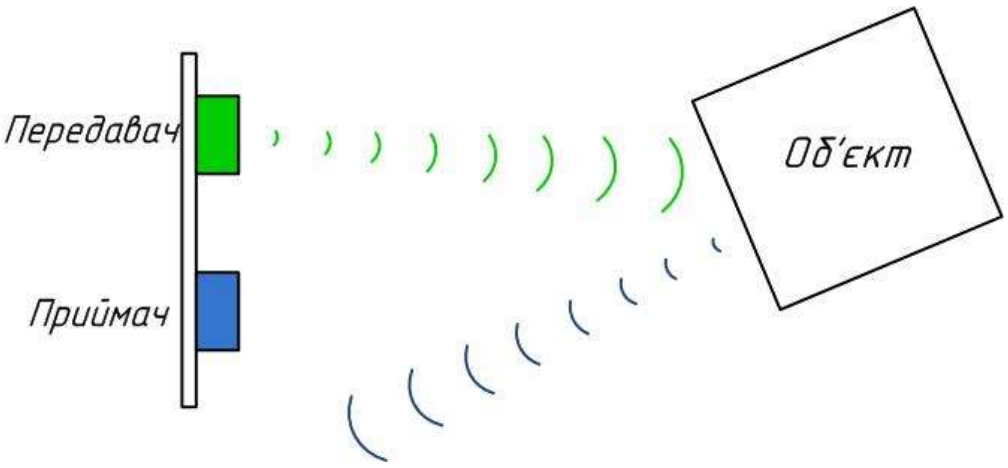


Рисунок 2.59 – Рух ультразвукового сигналу від передавача до приймача під великим кутом

Розрахунок відстані відбувається на підставі отриманого часу і обчислюється за формулою

$$S = v \cdot t; \quad t = \frac{T}{2} \Rightarrow \; S = \frac{v \cdot T}{2},$$

де v – швидкість звуку (340 м/с); t – час руху хвилі від сенсора до об’єкта; T – час руху хвилі від сенсора до об’єкта і назад.

Ділення на два потрібно, оскільки сигнал проходить відстань до об’єкта і назад, а нам потрібно визначити лише відстань до об’єкта.

Вищенаведеної формули цілком достатньо для коректного вимірювання відстані. Однак, якщо є потреба покращити точність вимірювання, потрібно врахувати низку факторів. По-перше, бажано враховувати температуру навколишнього середовища, де здійснюються вимірювання. Це зумовлено тим, що швидкість звуку в газах збільшується з підвищенням температури. Під час підвищення температури повітря на 1 °С швидкість звуку в ньому збільшується на 0,6 м/с. Графік залежності швидкості звуку від температури наведено на рис. 2.60, а у табл. 2.3 наведено деякі проміжні значення.

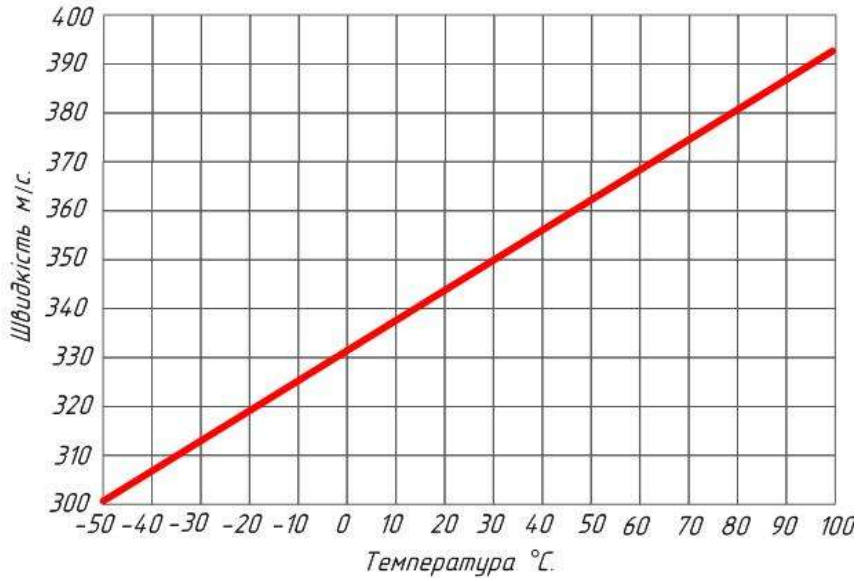


Рисунок 2.60 – Залежність швидкості звуку від температури

Таблиця 2.3 – Швидкість звуку за певної температури

t, °С	-20	-10	0	10	20	30
v, м/с	318,8	325,1	331,5	337,3	343,1	348,9

Щоб мати змогу оперативно підлаштовуватися під зміну температури навколишнього середовища, можна додати до системи вимірювання температурний сенсор (наприклад, DHT11 або DHT22) та використовувати його результати вимірів. На деяких далекомірах передбачені температурні сенсори.

Робота датчика відбувається у такий спосіб. Спочатку виводи «Echo» та «Trig» встановлені в «0». Для того, щоб запустити процес вимірювання, потрібно подати на вивід «Trig» імпульс тривалістю 10 мкс. Повторне вимірювання можна робити не раніше, ніж через 50 мс. Після цього на передавач надходять 8 коротких імпульсів для генерації ультразвукової хвилі. Після завершення генерації серії звукових хвиль вивід «Echo» встановлюється в «1». Після відображення серії хвиль від об’єкта та повернення на приймач, вивід «Echo» встановлюється у «0». У результаті отримаємо імпульс, тривалість якого дорівнює часу, витраченому на проходження звуку

від датчика до об’єкта і назад. Тривалість цього імпульсу лежить у межах від 118 мкс до 24 мс. Знаючи цей час, можна визначити відстань. На рисунку 2.61 наведено часову діаграму сигналів далекоміра.

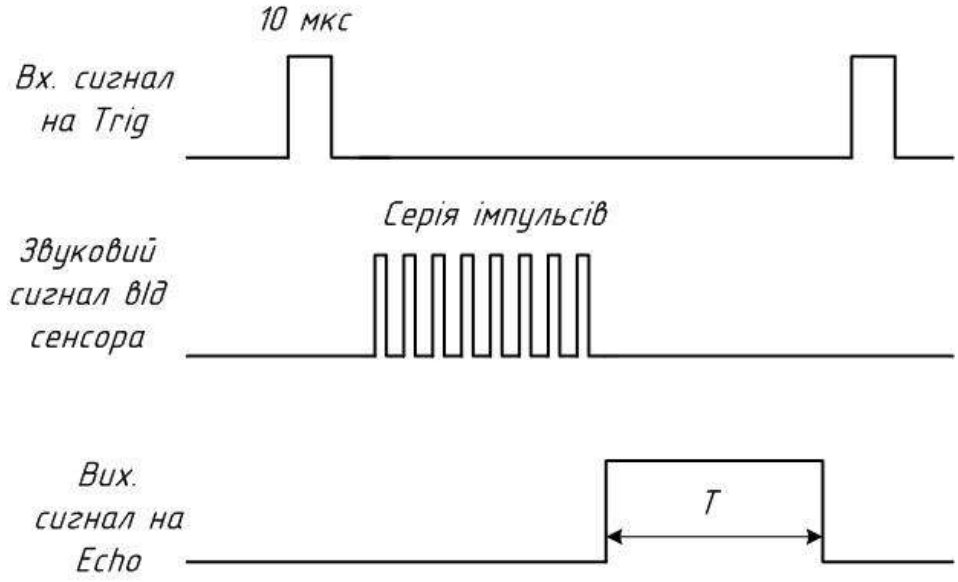


Рисунок 2.61 – Часова діаграма сигналів далекоміра

Програмуємо рух робота mBot з огинанням перешкод, використовуючи сенсор відстані Me Ultrasonic Sensor V3.0 у середовищі mBlock

Запустіть програму mBlock та перевірте налаштування. Використовуйте розташовані справа-зверху у палітрі скриптів (Scripts) блоки з розділів «Контроль» (Control) і «Робот» (Robots) і, перетягуючи їх на робоче поле у центрі mBlock, побудуйте програму, наведену на рис. 2.62.

Далі проаналізуємо роботу програми.

1. Блок «`mBot Program`» – заголовок програми для роботи mBot.

2. Блок «`forever`» – це нескінченний цикл, усередині якого всі блоки з командами виконуються нескінченну кількість разів, допоки ввімкнене живлення робота.

3. Блок «`if…then…else`» – умовний оператор перевіряє чи досягнено істинність умови: сигнал зчитаний з порту 4 «`ultrasonic sensor Port 4 distance < 20`».

Якщо умова істинна, виконується блок 4, а якщо ні – блок 5.

4. Блок «`turn left at speed 125`» – вмикає обертання двигунів робота mBot у протилежних напрямках так, щоб він повертався вліво на швидкості 125. Це половина від максимальної швидкість для двигунів.

5. Блок «`run forward at speed 255`» – команда для робота mBot рухатись уперед на швидкості 255.

6. Після цього циклу «`forever`» повторює усі команди, починаючи з блока «`run backward at speed 255`» у пункті 3, і так нескінченну кількість разів.

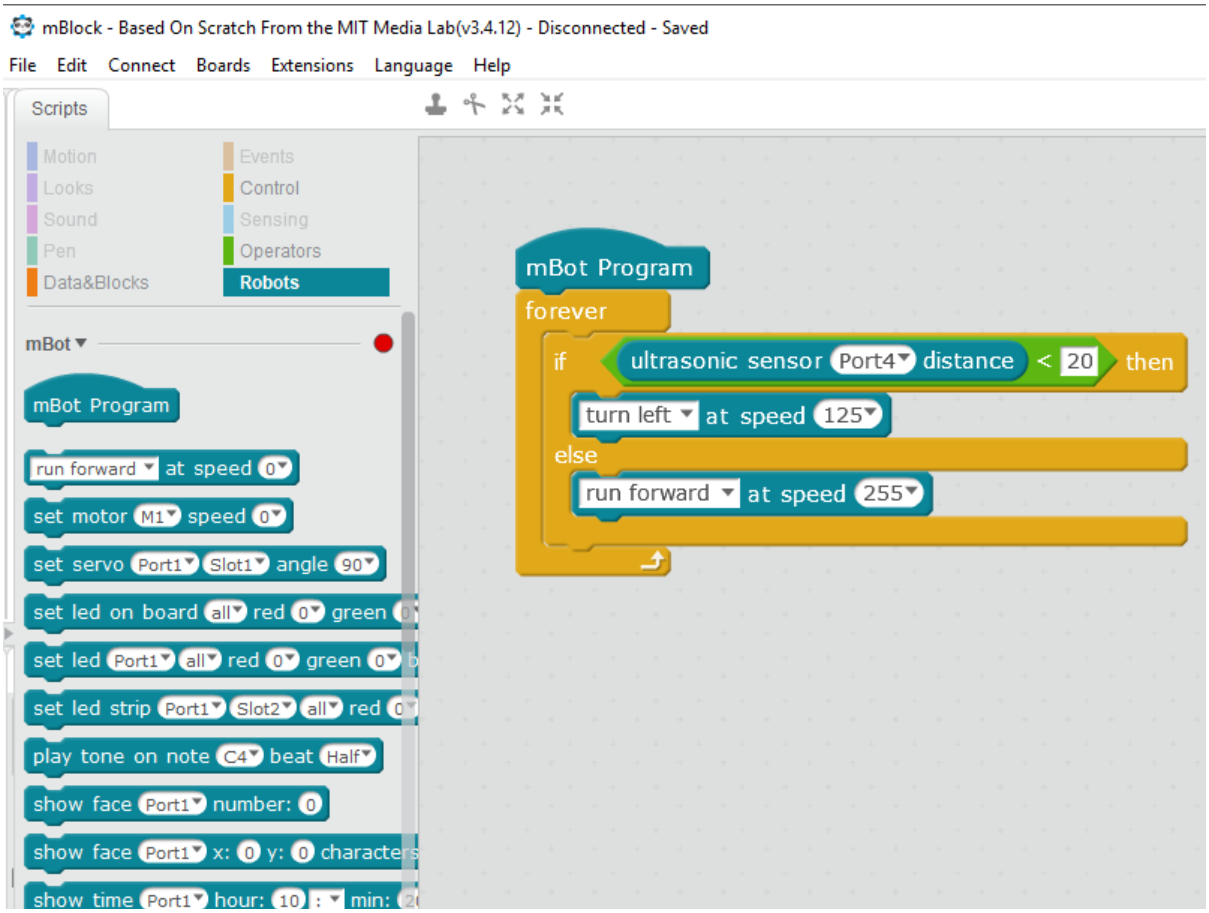


Рисунок 2.62 – Програма керування рухом робота mBot
з огинанням перешкод, із сенсором відстані Me Ultrasonic Sensor V3.0

Збережіть створену програму, обравши в верхньому меню File→Save Project As (див рис. 2.26).

Завантаження програми у пам'ять робота з mBlock

Перед завантаженням програми у пам'ять робота mBot перевірте з'єднання роботи з ПК і ввімкніть живлення робота.

Увага! Після успішного завантаження програми робот відразу почне обертати колесами. Будьте готові до цього, тримайте робота у руках.

Коли будете готові до завантаження, натисніть кнопку «Upload to Arduino» і зачекайте допоки програма не завантажиться в mBot. Після закінчення з'явиться повідомлення про успішне завантаження «Upload Finish» (див рис. 2.27), і mBot відразу розпочне повертати колесами вперед-назад! Тепер можна вимкнути живлення, від'єднати USB-кабель від робота та перенести робота на місце випробування.

Програмуємо рух робота mBot з огинанням завад, використовуючи сенсор відстані Me Ultrasonic Sensor V3.0 у середовищі Arduino IDE з використанням бібліотеки Makeblock

Для полегшення програмування робота mBot в Arduino IDE доцільно використовувати бібліотеку Makeblock. Завантажити її можна за посиланням: https://github.com/Makeblock-official/mBlock.

Після завантаження бібліотеки для свого проєкту, запаковану в ZIP-архів, її можна встановити через середовище Arduino IDE. Для встановлення потрібно перейти в меню: Скетч → Підключити бібліотеку → Додати бібліотеку .ZIP і вибрати ZIP-архів на ПК (див. рис. 2.14). Файл буде автоматично розпакований та переміщений у директорію з бібліотеками.

Запустіть середовище Arduino IDE та введіть скетч, наведений у прикладі 2.9.

```
#include <MeMCore.h>     //під'єднуємо бібліотеку Makeblock

 // оголошуємо двигуни
MeDCMotor motorLeft(M1);        //лівий двигун
MeDCMotor motorRight(M2);       //правий двигун

// оголошуємо ультразвуковий сенсор
MeUltrasonicSensor ultrasonic(1);  //під'єднання до 1 порту

uint8_t motorSpeed = 255;           // Задаємо максимальну
                                     швидкість двигунів 255

void setup() {
  }
void loop() {
  if ((ultrasonic.distanceCm()) < (20)) {
   motorLeft.run(motorSpeed/2); //Обертання лівого двигуна
   motorRight.run(motorSpeed/2);//Обертання правого двигуна

  } else {
    motorLeft.run(-motorSpeed); //Обертання лівого двигуна
    motorRight.run(motorSpeed); //Обертання правого двигуна
  }
}
```

Приклад 2.9 – Програма керування рухом робота mBot з огинанням перешкод із сенсором відстані Me Ultrasonic Sensor V3.0 в Arduino IDE з використанням бібліотеки Makeblock

Далі детально проаналізуємо роботу програми.

Під'єднуємо бібліотеку makeblock для плати mCore директивою

```
#include <MeMCore.h>
```

Потім створюємо об'єкти – лівий та правий двигуни та сенсор відстані, вказуючи у дужках порти, до яких вони підключені на платі mCore:

```
MeDCMotor motorLeft(M1);
MeDCMotor motorRight(M2);
MeUltrasonicSensor ultrasonic(1);
```

Значення швидкості двигунів 255 будемо для зручності зберігати у числовій змінній `Speed_motor`

```
uint8_t motorSpeed = 255;
```

Обов'язкова процедура `void setup()` виконується один раз під час увімкнення робота або після того, як було натиснуто кнопку Reset. Але нам не потрібно, щоб вона містила команди, тому вона порожня

```
void setup()
{
}
```

Обов'язкова процедура `void loop()` – це нескінченний цикл, тобто аналог блока «`forever`» в mBlock (Scratch). Вона містить усі команди для руху робота, які розташовані між фігурними дужками `{}`.

Далі програма перевіряє, чи значення `ultrasonic.distanceCm()` менше 20 чи ні. Якщо так, то виконується поворот ліворуч. Інакше кажучи, якщо вираз у круглих дужках істинний – виконуються оператори всередині фігурних дужок; якщо ж ні – програма пропускає цей код і виконує код, що знаходиться у фігурних дужках оператора `else`.

Дужки після оператора `if` можуть бути опущені. Якщо так зроблено, тільки наступний рядок (позначений точкою з комою) стає оператором, який виконується в операторі `if`.

```
if ((ultrasonic.distanceCm()) < (20)) {
    //Повертаємо ліворуч
  }
else {
    //Рухаємось прямо
}
```

Як вже було розглянуто вище, рух робота вперед та поворот ліворуч здійснюються за допомогою такого коду:

```
//Рух робота mBot вперед
   motorLeft.run(-motorSpeed);
   motorRight.run(motorSpeed);
//Поворот ліворуч
   motorLeft.run(motorSpeed/2);
   motorRight.run(motorSpeed/2);
```

Після цього підпрограма `void loop()` почне виконувати повторно команди, які знаходяться між її фігурними дужками `{}`, починаючи з руху робота вперед, і так безліч кількість разів.

Завантаження програми у пам'ять мікроконтролера робота mBot із Arduino IDE

Перед завантаженням програми у пам'ять mBot перевірте з'єднання робота з ПК та ввімкніть живлення робота. Далі клікніть по піктограмці «Вивантаження» (див. рис. 2.29). У випадку успішного завантаження програми з Arduino IDE у пам'ять мікроконтролера робота mBot Ви будете

спостерігати повідомлення «Вивантаження виконано» (див. рис. 2.30). Якщо ж Ви допустили якусь помилку, завантаження програми буде перервано. У цьому випадку прочитайте повідомлення про помилку, виправте її та повторіть завантаження (див. рис. 2.31).

Після успішного завантаження mBot відразу розпочне обертати колесами вперед-назад! Тепер можна вимкнути живлення, відімкнути USB-кабель від роботи та перенести робота на місце випробування.

2.9 Контрольні питання

1. Поясніть, для чого встановлюються пристрої «USB SerialConverter» та «USB SerialPort».
2. Поясніть призначення методу voidsetup().
3. Поясніть призначення методу voidloop().
4. Наведіть основні характеристики плати mCore.
5. Опишіть принцип роботи аналогового порту.
6. Опишіть принцип роботи ультразвукового далекоміра.
7. Поясніть, з якою метою встановлюються Н-міст.
8. Поясніть, що таке широтно-імпульсна модуляція. Наведіть відповідну діаграму.
9. Опишіть загальний принцип керування двоканальним драйвером двигунів TB6612.
10. Опишіть принцип роботи двигуна постійного струму.
11. Опишіть принцип роботи крокового двигуна.
12. Наведіть відомі Вам види крокових двигунів. Чим вони відрізняються?
13. Поясніть, як відбувається керування кроковим двигуном.
14. Поясніть, що таке сервопривід, які різновиди сервоприводів існують і чим вони відрізняються.
15. Поясніть, як відбувається керування сервоприводом сигналами керування.
16. Поясніть, до яких виводів підключається сервопривід до платформи Arduino.
17. Назвіть основні команди Arduino, що використовуються для керування сервоприводом.
18. Поясніть, що таке SMT (Surface-mount techology) світлодіод.
19. Поясніть, що таке світлодіод з інтелектуальним керуванням.
20. Поясніть, як відбувається вимірювання відстані за допомогою ультразвукового далекоміра.
21. Поясніть, що таке діаграма спрямованості ультразвукової хвилі.

3 ОСНОВИ ПРОГРАМУВАННЯ РОБОТІВ

3.1 Вступ до програмування роботів мовою С

Іноді створюється враження, що всі завдання можуть бути вирішені за допомогою готових комп'ютерних програм. У низці випадків це дійсно так, але, як показує досвід, завжди є завдання, які не вирішуються (або погано вирішуються) стандартними засобами. У цих випадках доводиться писати власну програму, яка робить усе так, як Ви цього хочете.

Два етапи створення програм

Програма мовою С (Сі), як і у більшості сучасних мов програмування, створюється у два етапи [62]:

1) трансляція – переклад тексту програми на машинні коди;

2) компонування – складання частин програми та приєднання стандартних функцій.

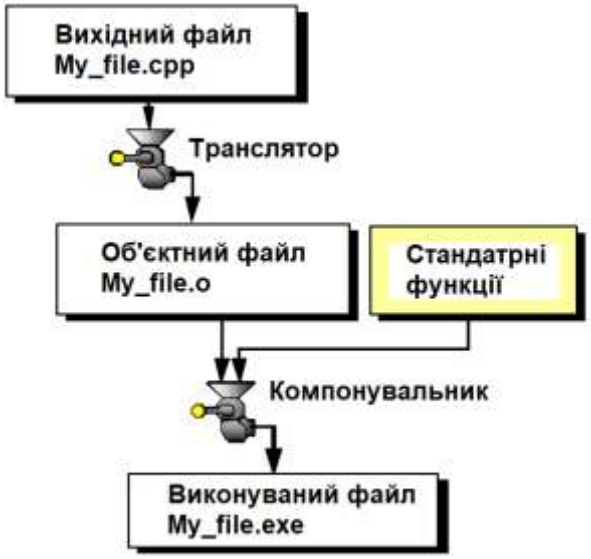


Рисунок 3.1 – Два етапи створення програм

Чому це все не прийнято робити за один крок? Для найпростіших програм це справді було б простіше, але для складних проєктів двоступінчастий процес має явні переваги:

- зазвичай складна програма розбивається на кілька окремих частин (модулів), які налаштовуються окремо й, найчастіше, різними людьми, тому на завершення залишається лише зібрати готові модулі у єдиний проєкт [63];

- при виправленні в одному модулі не треба знову транслювати (переводити в машинні коди) усі інші (це можуть бути десятки тисяч рядків);

- під час компонування у багатьох системах можна під'єднувати

модулі, написані іншими мовами, наприклад, асемблером (у машинних кодах).

Транслятори мови С називаються **компіляторами**: вони переводять (транслюють) відразу всю програму в машинний код, а не транслюють рядок за рядком під час виконання, як це роблять **інтерпретатори**. Такий підхід дозволяє значно прискорити виконання програми та не ставити інтерпретатор на кожен комп'ютер, де програма виконуватиметься [64].

Початковий файл програми мовою С має розширення ***.c** або ***.cpp** (розширення***.cpp** говорить про те, що у програмі можуть бути використані можливості мови С++). Це звичайний текстовий файл, в який можна записати текст програми в будь-якому текстовому редакторі, наприклад, у Блокноті [64].

Транслятор переводить вихідний файл (вірніше, записану у ньому програму) в машинні коди і будує так званий об'єктний файл з тим самим ім'ям і розширенням ***.o**. Хоча у ньому вже записано машинний код, об'єктний файл ще не можна запускати на ПК, оскільки в ньому не вистачає стандартних функцій (наприклад, для введення та виведення даних) [65].

Компонувальник під'єднує стандартні функції, що зберігаються у бібліотеках (вони мають розширення ***.a**). У результаті виходить один файл з розширенням ***.exe**, який і є готовою виконуваною програмою [66].

Найпростіша програма мовою С

Така програма складається усього з восьми символів (приклад 3.1).

```
main()
{
}
```

Приклад 3.1 – Найпростіша програма мовою С

Основна програма завжди називається ім'ям main (будьте уважні – С розрізняє великі та маленькі літери, а усі стандартні оператори С записуються маленькими літерами) [67]. Порожні дужки означають, що main не має аргументів. Фігурні дужки позначають початок і кінець основної програми – оскільки усередині нічого немає, наша програма нічого не робить, вона просто відповідає правилам мови С, її можна скомпілювати й отримати exe-файл [68].

Виведення тексту на екран

Складемо тепер програму, яка робить щось корисне, наприклад, виводить на екран слово «Привіт» (приклад 3.2).

```
#include <stdio.h>
main()
{
printf("Привіт!");
}
```

Приклад 3.2 – Програма мовою С, що друкує слово «Привіт!»

У першому рядку програми здійснюється підключення функцій стандартного введення та виведення, опис яких знаходиться у файлі stdio.h. У четвертому рядку виконується виклик функції виведення на екран.

Нижче наведемо кілька важливих тез.

- Для того, щоб використовувати стандартні функції, потрібно сказати транслятору, що є функція з таким ім'ям та перерахувати тип її аргументів – тоді він зможе визначити, чи правильно ми її використовуємо. Це означає, що треба додати до програми опис цієї функції. Описи стандартних функцій С розташовуються у заголовних файлах з розширенням *.h (у каталозі C:\Dev-Cpp\include) [69].
- Для під'єднання заголовних файлів використовується директива (команда) препроцесора #include, після якої у кутових дужках ставиться ім'я файлу. Усередині кутових дужок не має бути пропусків. Для підключення ще кожного нового заголовка файлу треба використовувати нову команду #include [64].
- Для відображення інформації на екрані використовується функція printf. У найпростішому випадку вона приймає єдиний аргумент – рядок у лапках, який треба вивести на екран [65].
- Кожен оператор мови С закінчується крапкою з комою [69].

3.2 Арифметичні вирази

Арифметичні вирази, що розташовуються у правій частині оператора присвоєння, можуть містити:

- цілі та речові числа (у речових числах ціла та дробова частини розділяються крапкою, а не комою, як це прийнято в математиці);
- знаки арифметичних дій: «+» (додавання), «-» (віднімання), «*» (множення), «/» (ділення), «%» (залишок від ділення);
- виклики стандартних функцій: abs(i) – модуль цілого числа i, fabs(x) – модуль дійсного числа x, sqrt(x) – квадратний корінь із дійсного числа x, pow(x,y) – обчислює x у степені y;
- круглі дужки для зміни порядку дій.

Особливості арифметичних операцій

Під час виконання ділення цілого числа на ціле залишок відкидається, таким чином, 7/4 дорівнюватиме 1. Якщо ж треба отримати дійсне число з залишком, ділене або дільник треба перетворити у дійсну форму (приклад 3.3).

```
int i, n;
float x;
i = 7;
x = i/4;         // x=1, ділиться ціле на ціле
x = i/4.;       // x=1.75, ділиться ціле на дробове
x =(float)i/4; // x=1.75, ділиться дробове на ціле
n = 7./4.;      // n=1, результат записується у цілу змінну
```

Приклад 3.3 – Різні варіанти ділення чисел

Найбільші складнощі з усіх дій викликає обчислення залишку. Якщо треба обчислити залишок від ділення змінної a на змінну b і результат записати у змінну zalyshok, то оператор присвоювання виглядає так:

```
zalyshok = a%b
```

Пріоритет арифметичних операцій

У мовах програмування арифметичні висловлення записуються в один рядок, тому потрібно знати пріоритет операцій, тобто послідовність їх виконання. Спочатку виконуються операції у дужках, потім виклики функцій, потім операції множення, ділення, залишку від ділення, а потім додавання та віднімання, зліва направо [62].

Наприклад,

```
t = (a + 7*b) * fabs (c + d) – (4 * b – c);
      2   1   5  4      3     8     6    7
```

Для зміни порядку виконання операцій використовують круглі дужки. Так, вираз

$$s = \frac{2x+4}{(3x-7z)(8z-4)} - \frac{5x}{x+z+1}$$

у комп'ютерному вигляді запишеться як

```
s = (2*x + 4) / ((3*x – 7*z) * (8*z – 4)) – 5*x / (x + z + 1);
```

Оператори присвоєння

У програмуванні часто використовуються дещо дивні оператори присвоювання, наприклад: `i=i+1;`. Якщо вважати це рівнянням, воно безглуздо з погляду математики. Проте, з погляду інформатики, цей оператор слугує для збільшення значення змінної і на одиницю. Буквально це означає: взяти старе значення змінної і, додати до нього одиницю та записати результат у ту саму змінну і.

Інкремент та декремент

У мові С визначено два спеціальні оператори швидкого збільшення на одиницю `i++;` або `++i;` (інкременту), що рівносильно оператору присвоювання `i=i+1;` а також швидкого зменшення на одиницю `i--;` або `--i;` (декременту), що рівносильно оператору присвоювання `i=i-1;`

Між першою та другою формами цих операторів є певна різниця, але тільки тоді, коли вони входять до складу складніших операторів чи умов.

Скорочений запис арифметичних виразів

Якщо ми хочемо змінити значення певної змінної (отримати її старе значення, щось із ним зробити і записати результат у цю ж змінну), то зручно використовувати скорочений запис арифметичних виразів, наведений у табл. 3.1

Формати виведення даних

Цілі числа

Першим параметром під час виклику функцій **scanf** і **printf** має бути символьний рядок, що визначає формат введення або виведення даних. Для функції **scanf**, яка виконує введення даних, достатньо просто вказати один із форматів **%d, %f** або **%c** для введення цілого числа, дійсного числа або символа, відповідно. У той же час форматний рядок у функції **printf** дозволяє керувати виведенням на екран, а саме: вказати розмір поля, яке відводиться для цього числа.

Таблиця 3.1 – Скорочений та повний записи арифметичних виразів

Скорочений запис	Повний запис
`x += a;`	`x = x + a;`
`x -= a;`	`x = x - a;`
`x *= a;`	`x = x * a;`
`x /= a;`	`x = x / a;`
`x %= a;`	`x = x % a;`

У таблиці 3.2 наведено приклади форматування під час виведення цілого числа 1234. Для того, щоб побачити поле, яке відводиться для числа, воно обмежене ліворуч і праворуч дужками.

Таблиця 3.2 – Приклади форматування під час виведення цілого числа 1234

Приклад виведення	Результат	Коментар
`printf("[%d]", 1234);`	`[1234]`	Мінімально можливе поле.
`printf("[%6d]",1234);`	`[  1234]`	6 позицій, вирівнювання праворуч.
`printf("[%-6d]",1234);`	`[1234  ]`	6 позицій, вирівнювання вліво.
`printf("[%2d]",1234);`	`[1234]`	Число не вміщується у задані 2 позиції, тому ділянка виведення розширюється.

Для виведення символів використовуються такі самі прийоми форматування, але формат **%d** замінюється на **%c.**

Дійсні числа

Для виведення (і введення) дійсних чисел можуть використовуватися три формати: **%f, %e** та **%g**. У таблиці 3.3 наведено приклади використання формату **%f.**

Формат **%e** застосовується у наукових розрахунках для виведення дуже великих або дуже маленьких чисел, наприклад, розмір атома або відстань до Сонця. З поданням числа у так званому стандартному вигляді (з виділеною мантисою та порядком). Наприклад, число **123.45** може бути записано

у стандартному вигляді як 1.2345×10^2. Тут **1.2345** – мантиса (вона завжди знаходиться в інтервалі від 1 до 10), а 2 порядок (мантиса множиться на 10 у цьому степені). При виведенні формату **%e** також можна задати число позицій, що відводяться для виведення числа, та число цифр у дробовій частині мантиси. Порядок завжди вказується у вигляді двох цифр, перед якими стоїть буква **e** та знак порядку (плюс або мінус). У таблиці 3.4 наведено приклади використання формату **%e.**

Таблиця 3.3 – Приклади використання формату **%f**

Приклад виведення	Результат	Коментар
`printf("[%f]",123.45);`	`[123.450000]`	Мінімально можливе поле, 6 знаків у дробовій частині.
`printf("[%9.3f]",123.45);`	`[ 123.450]`	Усього 9 позицій, їх 3 – для дробової частини, вирівнювання вправо.
`printf("[%-9.3f]",123.45);`	`[123.450 ]`	Усього 9 позицій, їх 3 – для дробової частини, вирівнювання вліво.
`printf("[%6.4f]",123.45);`	`[123.4500]`	Число не поміщається у задані 6 позицій (4 цифри в дробовій частині), тому поле виведення розширюється.

Таблиця 3.4 – Приклади використання формату **%e**

Приклад виведення	Результат	Коментар
`printf("[%e]",123.45);`	`[1.234500e+02]`	Мінімально можливе поле, 6 знаків у дробовій частині мантиси.
`printf("[%12.3e]",123.45);`	`[ 1.234e+02]`	Усього 12 позицій, з них 3 – для дробової частини мантиси, вирівнювання вправо.
`printf("[%-12.3e]",123.45);`	`[1.234e+02 ]`	Усього 12 позицій, з них 3 – для дробової частини мантиси, вирівнювання вліво.
`printf("[%6.2e]",123.45);`	`[1.23e+02]`	Число не вміститься у заданих 6 позицій (2 цифри у дробовій частині мантиси), тому поле виведення розширюється.

Формат **%g** застосовується для того, щоб видалити зайві нулі в кінці дробової частини числа та автоматично вибрати формат (у стандартному

вигляді або з фіксованою комою). Для дуже великих чи дуже маленьких чисел вибирається формат із плаваючою комою (у стандартному вигляді). У цьому форматі можна задати загальну кількість позицій на число та кількість дійсних цифр. Приклади використання формату **%e** наведено у таблиці 3.5.

Таблиця 3.5 – Приклади використання формату **%e.**

Приклад виведення	Результат	Коментар

`printf("[%g]",12345);`
`printf("[%g]",123.45);`
`printf("[%g]",0.000012345);`
	`[12345]`
	`[123.45]`
	`[1.2345e-05]`
		Мінімально можливе поле не більше 6 цифр.

`printf("[%10.3g]",12345);`
`printf("[%10.3g]",123.45);`
`printf("[%10.3g]",0.000012345);`
	`[  1.23e+04]`
	`[       123]`
	`[  1.23e-05]`
		Усього 10 позицій, з них 3 цифри, вирівнювання вправо. Щоб зробити вирівнювання вліво, використовують формат `"%-10.3g"`.

3.3 Типи даних і змінні

Для обробки даних їх потрібно зберігати у пам’яті. І до цих даних треба певним чином звертатися. Наприклад, люди звертаються один до одного за іменем, Аналогічний спосіб використовується у програмуванні: кожній комірці пам’яті (або групі комірок) дається своє власне ім’я. Використовуючи його, можна прочитати інформацію з комірки і записати туди нову інформацію [63].

Змінна – це комірка у пам’яті комп’ютера, яка має ім’я та зберігає певне значення. Значення змінної може змінюватись під час виконання програми. Після запису в комірку нового значення старе стирається [64].

З точки зору комп’ютера усі дані в пам’яті – це числа (набори нулів та одиниць). З цілими та дробовими числами працюють по-різному. Тому у кожній мові програмування є різні типи даних, для обробки яких використовуються різні способи [67]. Наприклад,

- цілі змінні – тип int (англ. integer – цілий), займають 4 байти у пам’яті;
- дійсні змінні, які можуть мати дробову частину (тип float – з англ. floating point – плаваюча кома), займають 4 байти у пам’яті;
- символи (тип char – з англ. character – символ), займають 1 байт у пам’яті.

Будь-яку змінну, що Ви будете використовувати у програмі, потрібно оголошувати – сказати комп’ютеру, щоб він виділив для неї комірку пам’яті потрібного розміру та присвоїв їй ім’я. Змінні зазвичай оголошуються на початку програми. Для оголошення треба написати назву типу змінних (int, float або char), а потім через кому імена усіх змінних, що ого-

лошуються. За бажанням можна відразу записати у нову комірку потрібне значення, як показано нижче у прикладах. Якщо змінній не надається жодного значення, то в ній знаходиться «сміття», тобто те, що там було раніше.

```
int a;        // виділить пам'ять під цілу змінну a
float b, c; // дві дійсних змінних a і b
int An12=4, Fl6, An225; // три цілих змінних,
                        // в An12 відразу записується число 4
float x=3.26, y, z;    // три дійсних змінних,
                       // в x відразу записується 4.56
char c, c2='U', m;      // три символьних змінних,
                       // в c2 відразу записується символ 'A'
```

Приклад 3.4 – Варіанти оголошення змінних

Нехай нам потрібно вивести на екран суму двох цілих чисел, введених з клавіатури. Така програма наведена у прикладі 3.5.

```
#include <stdio.h>
main()
{
int a, b, c; // оголошення змінних
printf ("Введіть 2 цілі числа \n");  // підказка ввести
                                      // 2 цілих числа
scanf ("%d%d", &a, &b);              // введення даних
c = a + b;                           // обчислення суми
printf ("Результат: %d+%d=%d\n",a,b,c); //виведення результату
getchar();
   }
```

Приклад 3.5 – Програма обчислення суми двох цілих чисел

Проаналізуємо дану програму. Отже, програма містить 4 частини:

- оголошення змінних;
- введення вихідних даних;
- обробка даних (обчислення);
- виведення результату.

Перед введенням даних потрібно вивести на екран підказку (інакше комп'ютер чекатиме на введення даних, а користувач не знатиме, що від нього хоче машина).

Символи `\n` у функції `printf` позначають перехід на початок нового рядка.

Для введення даних використовують функцію `scanf`. Формат введення – це рядок у лапках, де перераховано один або кілька форматів введення даних:

`%d` – введення цілого числа (змінна типу int);

`%f` – введення дійсного числа (змінна типу float);

`%c` – введення одного символу (змінна типу char).

Після формату введення через кому перераховуються адреси комірок пам'яті, в які треба записати введені значення (відчуйте різницю: **`a`** – значення змінної **`a`**, тоді, як **`&a`** – адреса змінної **`a`**). Зрозуміло, що кількість форматів у рядку має дорівнювати кількості адрес змінних у списку. Крім того, тип змінних має збігатися з зазначеним: наприклад, якщо **`a`** та **`b`** – цілі змінні, то виклики функцій, наведені у прикладі 3.6, є помилкові.

- `scanf("%d%d", &a);` (тут немає куди записувати друге введене число).
- `scanf("%d%d", &a, &b, &c);` (не задано формат для змінної c).
- `scanf("%f%f", &a, &b);` (не можна вводити цілі змінні за дійсним форматом).

Приклад 3.6 – Помилкові виклики функцій введення

Для обчислень використовують оператор присвоєння, у якому праворуч від знака «дорівнює» є арифметичний вираз (`a+b`), який треба обчислити, а ліворуч – ім'я змінної (`c`), в яку потрібно записати результат.

Для виведення чисел і значень змінних на екран використовують функцію `printf`. Вміст дужок під час виклику функції `printf` дуже схожий на функцію `scanf`: спочатку йде символьний рядок (формат виведення), в якому можна використовувати спеціальні символи (**`%d`** – виведення цілого числа; **`%f`** – дійсного, **`%c`** – одного символа, **`%s`** – символьного рядка, **`\n`** – перехід на початок нового рядка. Решта символів (крім деяких інших спеціальних команд) просто виводяться на екран. У нашій команді «`printf ("Результат: %d+%d=%d\n",a,b,c);`» Слово у лапках (Результат:) буде виведене без змін, а замість `%d+%d=%d` буде виведено значення введених раніше цілих `a, b` та обчисленої суми `(c)` у форматі `a+b=c`. Тобто, через кому після формату виведення (`%d+%d=%d\n",`) потрібно ще навести список чисел або змінних, значення яких треба вивести (у нашому випадку це `a,b,c`).

Доцільно зауважити, що через кому після формату виведення ми можемо задавати не просто список чисел або змінних, значення яких треба вивести, а й вирази, за якими потрібно виконувати обчислення, наприклад,

```
printf ( "Результат: %d + %d = %d \n", a, 45, a*4);
```

Не забувайте так само і для функції `scanf`, потрібно стежити за збігом типів та кількості змінних та форматів виведення.

3.4 Алгоритми розгалуження

У найпростіших програмах (лінійних) усі команди виконуються одна за одною послідовно. Однак часто треба вибрати той чи інший варіант дій залежно від деякої умови: якщо вона виконується, діяти одним способом, а якщо не виконується – іншим. Для цього використовують алгоритми, що розгалужуються, які в мовах програмування подані у вигляді умовних операторів [66]. У мові С є два види умовних операторів [64]:

- оператор `if – else` для вибору з двох варіантів
- оператор множинного вибору `switch` для вибору кількох варіантів.

Умовний оператор `if – else`

Завдання. Ввести з клавіатури два дійсних числа і визначити найбільше з них.

Отже, нам потрібно вивести один із двох варіантів відповіді: якщо перше число більше за друге, то вивести на екран його, якщо ні – то друге число. Нижче показано два варіанти розв'язання цього завдання: у першому (приклад 3.7) результат відразу виводиться на екран, а у другому (приклад 3.8) найбільше з двох чисел спочатку записується у третю змінну `Max`.

```
#include <stdio.h>
main()
{
float A, B;
printf("Введіть A, B: ");
scanf ("%f%f", &A, &B);
if (A > B)
{
printf ("Найбільше %f", A);
}
else
{
printf ("Найбільше %f", B);
}
getchar();
}
```

Приклад 3.7 – Перший спосіб визначення найбільшого з двох чисел

```
#include <stdio.h>
main()
{
float A, B, Max;
printf("Введіть A, B: ");
scanf ("%f%f", &A, &B);
if ( A > B ) // перевірка умови
{
Max = A; // блок «Якщо умова виконується»
}
else
```

```
    {
    Max = B; // блок «Якщо умова не виконується»
    }
    printf ("Найбільше %f", Max);
    getchar();
    }
```

Приклад 3.8 – Другий спосіб визначення найбільшого з двох чисел

У загальному плані умовний оператор має вигляд, який наведено у прикладі 3.9

```
if ( умова ) // заголовок з умовою
{
... // блок «якщо» – виконується, якщо умова в дужках дійсна
}
else
{
... // блок «інакше» – виконуються, якщо умова в дужках хибна
}
```

Приклад 3.9 – Загальний вигляд умовного оператора

Цей запис є єдиним оператором, тому між дужкою, що завершує блок «якщо» і словом `else` не можуть знаходитися жодні оператори.

- Після слова `else` ніколи НЕ ставиться умова – блок «інакше» виконується тоді, коли основна умова, зазначена у дужках після `if`, хибна.
- Якщо у блоці «якщо» або в блоці «інакше» лише один оператор, то фігурні дужки можна не ставити.
- В умові можна використовувати знаки таких логічних відношень: **>** (більше), **<** (менше), **>= (**більше або дорівнює**), <=** (менше або дорівнює), **=** (дорівнює), **!=** (не дорівнює).
- У мові С будь-яке число, не рівне нулю, означає справжню умову, а нуль – хибну.
- Якщо у блоці «інакше» не треба нічого робити (наприклад: «якщо у продажу є кава, купи каву», а якщо ні…), то весь блок «інакше» можна опустити та використати скорочену форму умовного оператора (приклад 3.10).

```
 if ( умова )
    {
    ...   // що робити, якщо умова дійсна
    }
```

Приклад 3.10 – Скорочена форма умовного оператора

Використовуючи скорочену форму умовного оператора, вирішення попереднього завдання могло б виглядати, як наведено у прикладі 3.11.

```
#include
<stdio.h>
```

```
main()
{
Float A, B, Max;
printf("Введіть A, B: ");
scanf ("%f%f", &A, &B); Max = A;
if (B > A)
Max = B;
printf ("Найбільше %f",Max);
getchar();
}
```

Приклад 3.11 – Визначення найбільшого з двох чисел із використанням скороченої форми умовного оператора

До блоків «якщо» та «інакше» можуть входити будь-які інші оператори, також інші вкладені умовні оператори, при цьому оператор `else` відносять до найближчого попереднього `if` (приклад 3.12).

```
if (A > 100000)
  if (A > 1000000)
    printf ("У вас дуже багато грошей.");
  else
    printf ("У вас достатньо грошей.");
else
 printf ("У вас замало грошей.");
```

Приклад 3.12 – Вкладені умовні оператори

Зауважимо, що для того, щоб легше розібратися у програмі, всі блоки «якщо» та «інакше» (разом з дужками, що їх обмежують) зсуваються вправо на 2–3 символи (запис «драбинкою»).

Складні умови

Найпростіші умови складаються з одного відношення (більше, менше тощо). Бувають випадки, коли треба написати умову, в якій об'єднуються два чи більше найпростіших відношень. Наприклад, фірма відбирає співробітників віком від 25 до 45 років (включно). Тоді найпростіша програма могла б мати такий вигляд, що наведено у прикладі 3.13.

```
#include
<stdio.h>
main()
{
int age;
printf ("\n Введіть Ваш вік: ");
scanf ("%d", &age);
if (age <= 25  & age <= 45)// складна умова
  printf ("Ви нам підходите.");
else
  printf ("Вибачте, Ви нам не підходите.");
getchar();
}
```

Приклад 3.13 – Реалізація складної умови

Складна умова складається з двох або більшої кількості простих відношень, які поєднуються за допомогою знаків логічних операцій «І», «АБО», «НІ» [68].

Операція «І» передбачає одночасне виконання двох (або більшої кількості) умов

```
умова_1 && умова_2
```

Цю операцію можна описати таблицею істинності 3.6

Таблиця 3.6 – Таблиця істинності операції «І»

умова_1	**умова_2**	**умова_1 && умова_2**
Хибно (0)	Хибно (0)	Хибно (0)
Хибно (0)	Істинно (1)	Хибно (0)
Істинно (1)	Хибно (0)	Хибно (0)
Істинно (1)	Істинно (1)	Істинно (1)

Операція «АБО» передбачає потрібно виконання хоча б однієї з двох умов (або обох відразу)

```
умова_1 || умова_2
```

Цю операцію можна описати таблицею істинності 3.7

Таблиця 3.7 – Таблиця істинності операції «АБО»

умова_1	**умова_2**	**умова_1 || умова_2**
Хибно (0)	Хибно (0)	Хибно (0)
Хибно (0)	Істинно (1)	Істинно (1)
Істинно (1)	Хибно (0)	Істинно (1)
Істинно (1)	Істинно (1)	Істинно (1)

У складних умовах іноді використовується операція «НІ» – заперечення умови

```
!умова
```

Наприклад, нижченаведені дві умови рівносильні

```
A > B!          (A <= B)
```

Порядок виконання (пріоритет) логічних операцій та відношень:

1. Операції в дужках, потім
2. Операція «НІ», потім
3. Логічні відносини >, <, >=, <=, ==, !=, потім
4. Операція «І», потім
5. Операція «АБО».

Для змінення порядку дій використовуються круглі дужки.

Оператор вибору switch (множинний вибір)

У випадку, якщо потрібно вибрати один з кількох варіантів залежно від значення деякої цілої або символьної змінної, можна використовувати кі-

лька вкладених операторів `if`, але зручніше використовувати спеціальний оператор `switch`.

Наприклад, нехай нам потрібно скласти програму, яка вводить з клавіатури українську літеру та виводить на екран назву тварини на цю літеру (приклад 3.14).

```
#include
<stdio.h>
main()
{
char c;
printf("\n Введіть першу літеру:");
scanf("%c", &c); //очікуємо введення літери
switch (c)       // заголовок оператору вибору
{
 case 'а': printf("\nАнтилопа");
 break; case 'б': printf("\nБорсук");
 break; case 'в': printf("\nВовк"); break;
 default: printf("\nНе знаю я таких!"); // за замовчуванням
}
getchar();
}
```

Приклад 3.14 – Реалізація множинного вибору

- Оператор множинного вибору `switch` складається з заголовка та тіла оператора у фігурних дужках.
- У заголовку після ключового слова `switch` у круглих дужках записано ім’я змінної (цілої або символьної). Залежно від значення цієї змінної робиться вибір між кількома варіантами.
- Кожному варіанту відповідає мітка `case`, після якої стоїть одне з можливих значень цієї змінної та двокрапка; якщо значення змінної збігається з однією з міток, то програма переходить на цю мітку та виконує всі нижченаведені оператори.
- Оператор `break` служить для виходу з тіла оператора `switch`. Якщо прибрати всі оператори `break`, то, наприклад, при натисненні на літеру «а» буде надруковано

```
Антилопа
Барсук
Вовк
Не знаю таких!
```

- Якщо значення змінної не збігається з жодною з позначок, програма переходить на позначку `default` (за замовчуванням, тобто якщо нічого іншого не вказано).
- Можна ставити дві позначки на один оператор. Наприклад, для того щоб програма реагувала як на великі, так і на малі літери, треба в тілі оператора `switch` написати, як наведено у прикладі 3.15, і т. д.

```
case 'а':
case 'А':
 printf("\nАнтилопа"); break;
case 'б':
case 'Б':
 printf("\nБарсук"); break;
```

Приклад 3.15 – Реалізації множинного вибору

3.5 Циклічні програми

Часто буває так, що нам потрібно кілька разів виконати якусь дію. Наприклад, потрібно вивести на екран рядок символів 100 разів. Звичайно, можна написати 100 разів оператор printf, але програма значно збільшиться. У таких випадках доцільно використовувати цикли. Цикл – це послідовність команд, яка виконується кілька разів [62].

У мові С є кілька видів циклів [64].

Цикл з відомою кількістю кроків (for)

Часто ми заздалегідь знаємо (або можемо розрахувати) скільки разів нам треба виконати якусь операцію. У деяких мовах програмування для цього використовується цикл `repeat` – «повтори задану кількість разів». Даний цикл виконується так. У пам'яті виділяється комірка і записується число повторень. Коли програма виконує тіло циклу один раз, вміст цієї комірки (лічильник) зменшується на 1. Виконання циклу закінчується, коли у цій комірці буде 0.

У мові С циклу `repeat` немає, а є цикл `for`. Він не приховує комірку-лічильник, а потребує її явного оголошення (виділення під неї пам'яті), і навіть дозволяє використовувати її значення у тілі циклу. У прикладі 3.16 наведено програму, яка друкує вітання 100 разів.

```
#include <stdio.h>
main()
{
int i; // оголошення змінної циклу
for ( i = 1; i <= 100; i ++ ) // заголовок циклу
{ // початок циклу (відкриваюча дужка)
printf ("Привіт"); // тіло циклу
}    //   кінець циклу (закриваюча скобка)
getchar();
}
```

Приклад 3.16 – Реалізація множинного вибору

- Цикл `for` використовується тоді, коли кількість повторень циклу заздалегідь відома або може бути розрахована.
- Цикл `for` складається з заголовка та тіла циклу.
- У заголовку після слова `for` у круглих дужках записуються через крапку з комою три вирази:

- початкові значення: оператори присвоєння, які виконуються один раз перед виконанням циклу;

- умова, в якій виконується наступний крок циклу; якщо умова не виконується, робота циклу закінчується; якщо умова не виконується на самому початку, цикл не виконується жодного разу (кажуть, що це цикл із передумовою, тобто умова перевіряється перед виконанням циклу);

- дії наприкінці кожного кроку циклу (переважно це оператори присвоювання).

- У кожній частині заголовка може бути кілька операторів, розділених комами. Приклади заголовків:

```
for (i = 0; i < 10; i ++) { ... }
for (i = 0, x = 1.; i < 10; i += 2, x *= 0.1){ ... }
```

- Тіло циклу розташовується у фігурних дужках; якщо у тілі циклу є лише один оператор, дужки можна не ставити.

- У тіло циклу можуть входити будь-які інші оператори, зокрема інші цикли (такий прийом називається «вкладені цикли»).

- Для того, щоб легше розібратися у програмі, все тіло циклу і дужки, що обмежують його, зсуваються вправо на 2–3 символи (запис «драбинкою»).

Напишемо програму, яка вводить з клавіатури натуральне число N і виводить на екран квадрати усіх цілих чисел від 1 до N (приклад 3.17) у такому вигляді

```
Квадрат числа 1 дорівнює 1
Квадрат числа 2 дорівнює 4
```

```
#include <stdio.h>
main()
{
int i, N; // i – змінна циклу
printf ("Введіть число N: "); // підсказка
scanf ("%d", &N)              // введення N з клавіатури
for (i = 1; i <= N; i ++)     // цикл: для всіх i від 1 до N
{
printf ("Квадрат числа %d рівний %d\n", i, i*i);
}
getchar();
}
```

Приклад 3.17 – Програма виведення квадратів усіх цілих чисел від 1 до N

Ми оголосили дві змінні: N – максимальне число, `i` – допоміжна змінна, яка в циклі послідовно приймає всі значення від `1` до N. Для введення значення N ми надрукували на екрані підказку і використовували функцію `scanf` з форматом `%d` (введення цілого числа).

При вході у цикл виконується оператор `i = 1`, а потім змінна `i` з кожним кроком збільшується на одиницю (`i++`). Цикл виконується наразі умо-

вою `i <= N`. У тілі циклу єдиний оператор виведення друкує на екрані саме число та його квадрат за заданим форматом. Для піднесення до квадрата або інший невисокий степень краще використовувати множення.

Цикл із умовою (while)

Дуже часто заздалегідь неможливо сказати, скільки разів треба виконати певну операцію, але можна визначити умову, за якої вона має закінчуватися. Таке завдання українською може виглядати так: роби цю роботу поки вона не буде завершена (пиляти колоду, доки вона не буде розпиляна; йди вперед, доки не дійдеш до університету). Слово «поки» англійською мовою записується як `while`, так само називається ще один вид циклу [65].

Нехай наша задача – це ввести натуральне число та визначити, скільки у ньому цифр.

Для розв'язання даної задачі зазвичай застосовується такий алгоритм. Число ділиться на 10 і відкидається залишок, так продовжується, допоки результат ділення не дорівнює нулю. За допомогою спеціальної змінної (вона називається лічильником) рахуємо скільки разів виконувалося ділення, – стільки цифр і було у числі. Зрозуміло, що не можна заздалегідь визначити, скільки разів треба поділити число, тому треба використовувати цикл з умовою (приклад 3.18).

```
#include <stdio.h>
main()
{
int N;                              // число, яке обробляємо
int count=0;                        // змінна-лічільник
printf ("\nВведіть число N: ");     // підказка
scanf ("%d", &N);                   // введення N з клавіатури
while (N > 0)                       // заголовок циклу «поки N>0»
{                                   // початок циклу
N /= 10;                            // відсікаємо останню цифру
count ++;                           // збільшуємо лічильник цифр
}                                   // кінец циклу
printf ("У цьому числі %d цифр\n", count );
getchar();
}
```

Приклад 3.18 – Визначення кількості цифр натурального числа

Цикл `while` використовується тоді, коли кількість повторень циклу заздалегідь невідома і не може бути обчислена.

- Цикл `while` складається з заголовка і тіла циклу.

- У заголовку після слова `while` у круглих дужках записується умова, за якої цикл продовжує виконуватися. Коли ця умова порушується (стає хибною), цикл закінчується.

- В умові можна використовувати знаки таких логічних відношень та операцій: **>** (більше), **<** (менше), **>= (**більше або дорівнює**), <=** (менше або дорівнює), **==** (дорівнює), **!=** (не дорівнює).

- Якщо умова не виконується на самому початку, то цикл не виконується жодного разу (це цикл із передумовою).

- Якщо умова ніколи не стає хибною (неправильною), то цикл ніколи не завершується – у такому разі кажуть, що програма «зациклилася» – це серйозна логічна помилка.

- У мові С будь-яке число, що не дорівнює нулю, позначає істинну умову, а нуль – хибну:

```
while ( 1 ){ ... } // нескінченний цикл
while ( 0 ){ ... } // цикл не виконається жодного разу
```

- Тіло циклу вміщується у фігурні дужки, якщо у тілі циклу стоїть лише один оператор, дужки можна не ставити.

- У тіло циклу можуть входити будь-які інші оператори, зокрема й інші цикли (такий прийом називається «вкладені цикли»).

Насамкінець зауважимо, що мовою С заміна циклу for на while і навпаки можлива завжди.

Цикл із постумовою (do – while)

Існують випадки, коли треба виконати цикл хоча б раз, а потім на кожному кроці робити перевірку деякої умови і закінчити цикл, коли дана умова стане хибною. Для цього використовується цикл із постумовою (тобто умова перевіряється не на початку, а у кінці циклу) [68]. Не рекомендується застосовувати його занадто часто, оскільки він нагадує таку ситуацію: стрибнув у басейн, і тільки потім подивився, чи є у ньому вода. Розглянемо випадок, коли його використання доцільне.

Нехай нам потрібно ввести натуральне число і знайти суму його цифр. При цьому організувати введення числа потрібно так, щоб не можна було ввести від'ємне число або нуль.

Фактично кожна програма має забезпечувати захист від неправильного введення даних (іноді такий захист називають «захистом від дурня» – fool proof). Оскільки користувач може вводити дані неправильно скільки завгодно разів, то треба використовувати цикл з умовою. З іншого боку, один раз обов'язково треба ввести число, тому потрібен цикл із постумовою.

На відміну від попередньої програми, тепер треба під час кожного ділення визначати залишок (остання цифра числа дорівнює залишку від ділення його на 10) і підсумовувати усі залишки у спеціальній змінній (приклад 3.19).

```
#include <stdio.h>
main()
{
int N, sum;       // sum - сума цифр числа
sum = 0;          // на початку сума = 0
do {// початок циклу
   printf ("\nВведіть натуральне число:");
   scanf ("%d", &N);
```

```
   }
while (N <= 0);  // умова циклу «поки N <= 0»
   while (N > 0) {
   sum += N % 10;
   N /= 10;
   }
printf ("Сумма цифр числа = %d\n", sum);
getchar();
}
```

Приклад 3.19 – Програма обчислення суми цифр натурального числа

Цикл do – while використовується тоді, коли кількість повторень циклу заздалегідь невідома і не може бути обчислена [62].

- Цикл складається з заголовка do, тіла циклу та завершальної умови.
- Умова записується в круглих дужках після слова while, цикл продовжує виконуватися доки умова є правильною; коли умова стає неправильною, цикл закінчується.
- Умова перевіряється тільки наприкінці чергового кроку циклу (це цикл із постумовою), отже, цикл завжди виконується хоча б один раз.
- Якщо умова ніколи не стає хибною (неправильною), то цикл ніколи не завершується, у такому разі кажуть, що програма «зациклилася» – це серйозна логічна помилка.
- Тіло циклу вкладається у фігурні дужки, якщо у тілі циклу стоїть лише один оператор, дужки можна не ставити.
- У тіло циклу можуть входити будь-які інші оператори, зокрема й інші цикли (такий прийом називається «вкладені цикли»).

Достроковий вихід із циклу

Іноді потрібно вийти з циклу і перейти до наступного оператора, не чекаючи закінчення чергового кроку циклу. Для цього використовують спеціальний оператор `break`. Можна також сказати комп'ютеру, що треба завершити поточний крок циклу і відразу перейти до нового кроку (не виходячи з циклу). Для цього застосовують оператор `continue`.

Нехай нам потрібно написати програму, що обчислює частку і залишок від ділення двох введених цілих чисел. Програма має працювати у циклі, тобто запитувати значення діленого та дільника, виводити результат, знову запитувати дані тощо. Якщо обидва числа дорівнюють нулю, потрібно вийти з циклу і завершити роботу програми. Також потрібно передбачити повідомлення про помилку, якщо друге число дорівнює нулю, а перше – ні.

Особливість цієї задачі полягає в тому, що при вході у цикл ми не можемо визначити, чи треба буде виконати до кінця черговий крок. Необхідна інформація надходить лише під час введення даних із клавіатури. Тому тут використовується нескінченний цикл `while(1){...}` (нагадаємо, що у мові С 1 вважається істинною умовою). Вийти з такого циклу можна тільки за допомогою спеціального оператора `break` (приклад 3.20).

Водночас, якщо друге число дорівнює 0, то частину циклу, що залишилася, виконувати не потрібно. Для цього слугує оператор `continue`.

```
#include <stdio.h>
main()
{
int A, B;
while (1) // нескінченний цикл, вихід тільки по break!
 {
 printf ("\nВведіть ділене і дільник:");
 scanf ("%d%d", &A, &B);
 if (A == 0 && B == 0) break; // вихід із циклу
 if (B == 0) {
   printf ("Ділення на нуль");
   continue; // достроковий перехід до нового кроку циклу
   }
 printf ("Частка %d, залишок %d", A/B, A%B);
 }
getchar();
}
```

Приклад 3.20 – Програма обчислення частки і залишку від ділення двох цілих чисел

- Якщо тільки усередині циклу можна визначити, чи треба робити обчислення у тілі циклу і чи треба продовжувати цикл (наприклад, при введенні вихідних даних), часто використовують нескінченний цикл, усередині якого є оператор виходу `break`:

```
while ( 1 ) {
...
if ( треба вийти ) break;
...
}
```

- За допомогою оператора `break` можна достроково виходити з будь-яких циклів: for, while, do-while.

- Для того, щоб достроково завершити поточний крок циклу й одразу перейти до наступного кроку, використовують оператор `continue`.

3.6 Масиви

Основні поняття

Основне призначення комп'ютерів не обчислення, як багато хто вважає, а обробка великих обсягів даних. Під час розміщення великої кількості даних у пам'яті виникає така проблема: треба навчитися звертатися до кожної комірки з даними окремо. При цьому досить складно дати кожній комірці власне ім'я і при цьому не заплутатися. Викручуються з цієї ситуації так: дають ім'я не комірці, а групі комірок, де кожна комірка має номер. Така область пам'яті називається масивом.

Масив – це група комірок пам’яті однакового типу, що розташовані поруч і мають спільне ім’я. Кожна комірка у групі має унікальний номер [62].

Під час роботи з масивами треба навчитися вирішувати три завдання:

- виділяти пам’ять потрібного розміру під масив;
- записувати дані у потрібну комірку;
- читати дані з комірки.

Оголошення масиву

Для того, щоб використати масив, його треба оголосити – виділити місце у пам’яті. Тип масиву визначається типом елементів, що входять до нього. Масиви можуть бути різних типів – `int`, `float`, `char`, і т. д. Масив оголошують так само, як і звичайні змінні, але після імені масиву в квадратних дужках записується його розмір:

```
int A(10], B[20]; // 2 масиви на 10 і 20 цілих чисел
float C[12];      // масив із 12 дійсних чисел
```

Під час оголошення масиву можна відразу заповнити його початковими значеннями, перераховуючи їх усередині фігурних дужок:

```
intA[4] = {2, 3, 12, 76}
```

Якщо у списку у фігурних дужках записано менше чисел, ніж елементів у масиві, то елементи, що залишилися, заповнюються нулями. Якщо чисел більше, ніж треба, транслятор повідомляє про помилку. Наприклад,

```
int A[4] = {2}; // останні три елементи дорівнюють 0
```

Для підвищення універсальності програми розмір масиву краще визначати через константу. В цьому випадку для переробки програми для масиву іншого розміру треба лише змінити значення цієї константи:

```
const int N = 20; // Константа
main()
{
int A[N]; // розмір масиву задано через константу
…
}
```

У таблиці 3.8 наведено приклади правильного та неправильного оголошення масиву.

Таблиця 3.8 – Приклади правильного та неправильного оголошення масиву

Правильно		неправильно	
`int A[20];`	розмір масиву вказано явно	`int A[];`	розмір масиву невідомий
`const int N=20;` `int A[N];`	розмір масиву – постійна величина	`int N=20;` `int A[N];`	розмір масиву не може бути змінною

Звертання до елемента масиву

Кожен елемент масиву має власний порядковий номер. Для того, щоб звернутися до елемента масиву, треба написати ім'я масиву, а потім у квадратних дужках номер потрібного елемента. Потрібно запам'ятати одне важливе правило: елементи масивів у мові С нумеруються з нуля. Таким чином, якщо у масиві 10 елементів, він містить елементи:

```
A[0], A[1], A[2], …, A[9]
```

Номер елемента масиву також називається його індексом. Ось приклади звернення до масиву:

```
x = (A [3] + 5) * A [1]; // прочитати значення A[3] та A[1]
A [0] = x + 6; // записати нове значення в A[0]
```

У мові С не контролюється вихід за межі масиву, тобто формально Ви можете записати щось в елемент з неіснуючим індексом, наприклад A[345] або в A[-12]. Однак при цьому Ви стираєте певну комірку пам'яті, що не належить масиву, тому наслідки такого кроку непередбачувані, наприклад, у ряді випадків програма «зависає».

Введення з клавіатури та виведення на екран

Як же ввести дані у масив? Існує багато способів залежно від Вашого завдання:

- елементи масиву вводяться з клавіатури вручну;
- масив заповнюється випадковими числами (наприклад для моделювання випадкових процесів);
- елементи масиву зчитуються з файлу;
- елементи масиву надходять через порт із зовнішнього пристрою (наприклад, сканера, модему тощо);
- масив заповнюється у процесі обчислень.

Нехай нам потрібно ввести з клавіатури масив із 10 елементів, помножити усі елементи на 2 та вивести отриманий масив на екран (приклад 3.21).

На жаль, неможливо просто сказати комп'ютеру: «Введи масив А». Ми маємо кожен елемент прочитати окремо, наприклад, викликавши функцію введення `scanf`).

Введення з клавіатури застосовується у найпростіших програмах, коли обсяг інформації, що вводиться, невеликий. Для введення масиву використовуватимемо цикл for. Нагадаємо, що масив треба попередньо оголосити, тобто виділити під нього пам'ять.

Вводити можна стільки елементів масиву, скільки комірок пам'яті виділено. Пам'ятайте, що елементи масиву нумеруються з нуля, тому якщо масив має 10 елементів, то останній елемент має номер 9. Якщо намагатися записувати в 10-й елемент, відбудеться вихід за межі масиву, і програма може працювати неправильно (а, можливо, й зависне). При введенні маси-

ву бажано видати на екран загальну підказку для введення всього масиву та підказки для кожного елемента.

Для множення елементів масиву на 2 треба знову використовувати цикл, у якому за один раз обробляється 1 елемент масиву.

Виведення масиву на екран виконується також у циклі for. Елементи виводяться по одному. Якщо у кінці рядка формату в операторі printf поставити пропуск, то елементи масиву буде надруковано у рядок, а якщо використати символ \n – то у стовпець.

```
# include <stdio.h>
const int N = 10; // розмір масиву
main()
{
int i, A[N];   // оголошення масиву
printf ("Введіть масив A\n"); // підказка для введення
for (i = 0; i < N; i++) {     // цикл по всіх елементах //
  printf ("Введіть A[%d]> ", i); //підказка для введення A[i]
  scanf ("%d", &A[i]); // Введення A [i]
  }
for (i = 0; i < N; i++) // цикл по усіх елементах
  A[i] = A[i] * 2;  // помножити A(i) на 2
printf ("\nРезультат:\n");
for (i = 0; i < N; i++) // цикл по усіх елементах
  printf("%d", A[i]);  // вивести A[i]
}
```

Приклад 3.21 – Програма роботи з масивом

Заповнення випадковими числами

Цей прийом використовується для моделювання випадкових процесів, наприклад, броунівського руху частинок. Нехай потрібно заповнити масив рівномірно розподіленими випадковими числами в інтервалі [a, b]. Оскільки для цілих та дійсних чисел способи обчислення випадкового числа у заданому інтервалі відрізняються, розглянемо обидва варіанти. Тут і далі передбачається, що на початку програми є рядок

```
const int N = 10;
```

Опис функції-датчика випадкових чисел знаходиться у заголовку `stdlib.h`. Зручно також додати до своєї програми функцію `random`:

```
int random (int N) {return rand()%N;},
```

яка видає випадкові числа з рівномірним розподілом в інтервалі [0, N–1]. Для отримання випадкових чисел з рівномірним розподілом в інтервалі [a, b] треба використовувати формулу

```
k = random(b-a+1) + a;
```

Для дійсних чисел формула дещо інша

```
x = rand() * (b - a) / RAND.MAX + a;
```

Тут константа RAND_MAX – це максимальне випадкове число, яке видає стандартна функція rand.

У прикладі 3.22 масив A заповнюється випадковими цілими числами в інтервалі [-5, 10], а масив X – випадковими *дійсними* числами у тому ж інтервалі.

```
# include <stdlib.h>
const int N=10;
main()
{
int i, A[N], a=-5, b=10;
float X[N];
for (i=0; i<N; i++)
  A[i] = random(b-a+1) + a;
for (i=0; i<N; i++)
  X[i] = (float)rand()*(ba)/RAND_MAX + a;
// тут ми робимо потрібні дії з масивами
)
```

Приклад 3.22 – Фрагмент програми заповнення масивів випадковими числами

Можливо, у цьому прикладі не зовсім зрозуміло, навіщо перед викликом функції rand є слово (float). Це пов'язано з тим, що у нас a і b – цілі числа. Результат функції rand – теж ціле число. Тут можливі дві проблеми:

- під час множення результату функції rand на ba може вийти дуже велике число, яке не поміститься у змінну типу int;
- у мові С під час ділення цілого числа на ціле залишок відкидається, тому результат буде неправильним.

Розглянемо ще одне завдання. Нехай нам потрібно заповнити масив випадковими цілими числами в інтервалі [-10, 15], помножити усі елементи на 2 і вивести на екран вихідний масив та результат (приклад 3.23).

```
include <stdio.h>
include <stdlib.h>
const int N = 10;
int random (int N) {return rand() %N;}
main()
{
int i, A[N);
for (i=0; i<N; i++)
A[i] = random(26)-10; // Заповнення масиву випадковими числами
printf("n  Вихідний  масив:\n");  //Виведення  вихідного  масиву
for (i=0; i<N; i++)
  printf("%d", A[i]);
for (i=0; i<N; i++)
  A[i] = A[i]*2; //помножити усі елементи на 2
Printf("n Результат:\n);
  for (i=0; i<N; i++)
printf("%d", A[i]); // Виведення результату
}
```

Приклад 3.23 – Програма обробки масиву

На відміну від попередніх, програма, наведена у прикладі 3.24, видає результати не на екран, а у файл output.dat у поточному каталозі.

```
# include <stdio.h>
const int N = 10;
main()
{
int i, A(N);
FILE *fp; // покажчик на файл
fp = fopen("input.dat", "r"); // відкрити файл для читання
if (fp == NULL) {   // обробка помилки
  printf("Немає файлу даних");
  return 1;    // вихід за помилкою, код помилки 1}
for (i=0; i<N; i++)
  if (1!=fscanf(fp,"%d",&A[i])) {// читання та обробка помилки
   printf("Не вистачає даних у файлі");
return 1;
}
fclose (fp); // Закрити файл
for (i=0; i<N; i++)
  A[i]=A[i]*2;
fp = fopen(”output.dat", "w"); // Відкрити файл на запис
for (i=0; i<N; i++) // вивести масив у файл у стовпчик
  fprintf (fp,"%d\n", A[i]);
fclose (fp);
}
```

Приклад 3.24 – Програма обробки масиву у файлі

Масив невідомого розміру

Нехай у файлі input.dat записані у два стовпці пари чисел (x, y). Потрібно записати у файл output.dat у стовпець суми x + y для кожної пари.

Складність цього завдання полягає в тому, що ми не можемо прочитати усі дані відразу, обробити їх і записати у вихідний файл. Не можемо тому, що не знаємо, скільки пар чисел у масиві. Звичайно, якщо відомо, що у файлі, скажімо, не більше 200 чисел, можна виділити масив «із запасом», прочитати стільки даних, скільки потрібно, і працювати тільки з ними. Однак у файлі можуть бути мільярди чисел і такі масиви не помістяться у пам’яті.

Проте, якщо подумати, стає зрозуміло, що для обчислення суми кожної пари потрібні лише дві числа. Коли обчислили їхню суму, її також не треба зберігати у пам’яті, а можна одразу записати у вихідний файл. Тому використовуватимемо такий алгоритм:

1) відкрити два файли: один на читання (з вихідними даними), другий – на запис результату;

2) спробувати прочитати два числа у змінні `x`, `y`; якщо це не вийшло (немає більше даних або дані неправильні), завершити роботу;

3) обчислити суму `x+y` та записати результат у вихідний файл;

4) перейти до кроку 2.

Для того, щоб визначити, чи успішно закінчилося читання, ми будемо використовувати той факт, що функція `fscanf` (як і `scanf`) повертає кількість вдало полічених чисел (приклад 3.25). За один раз читатимемо відразу два числа x, y. Якщо усе завершилося успішно, функція `fscanf` повертає значення 2 (обидві змінні прочитані). Якщо результат цієї функції менший за два, дані закінчились або неправильні.

Зауважимо, що треба працювати одночасно з двома відкритими файлами, тому у пам'яті треба використовувати два покажчики на файли, вони позначені іменами `fin` і `fout`. Для скорочення запису, помилки під час відкриття файлів не обробляються.

```
include <stdio.h>
main()
{
int n, x, y, sum;
FILE *fin, *fout;   // покажчики на файли
fin = fopen("input.dat", "r"); // відкрити файл для читання
fout = fopen("output.dat", "w");// відкрити файл для запису
while (1){
  n = fscanf (fin, "%d%d", &x, &y);
  if (n < 2) break; // дані помилкові чи їх більше немає
  sum = x + y;
  fprintf (fout, "%d\n", sum);}
fclose (fout); // закрити файли
fdose (fin);
}
```

Приклад 3.25 – Програма роботи з масивом невідомого розміру

У програмі використовується нескінченний цикл `while`. Програма виходить із нього тоді, коли дані у файлі закінчилися.

Робота з двійковими файлами

Двійкові файли відрізняються від текстових тим, що в них записано інформацію у внутрішньому машинному поданні. Двійковий файл не можна переглянути на екрані (вірніше, можна переглянути, але дуже складно зрозуміти). Але є й переваги – з двійкових файлів можна читати одразу весь масив у вигляді єдиного блока. Також можна записати весь масив або будь-яку безперервну частину за одну команду [64].

Під час відкриття двійкового файлу замість режимів «r», «w» і «a» використовують відповідно «rb», «wb» і «ab». Додаткова літера «b» вказує на те, що файл двійковий (від англійського слова binary – двійковий) [65].

Нехай нам потрібно ввести масив із 10 цілих чисел із двійкового файлу in.dat, помножити кожен елемент на 2 і вивести у двійковий файл out.dat (приклад 3.26).

```
#include <stdio.h>
const int N = 10;
main()
{
int i, n, A[N];
FILE *fp;                    // покажчик на файл
Fp=fopen("in.dat", "rb"); //відкрити двійковий файл на читання
n = fread (A, sizeof(int), N, fp); // читаємо весь масив
if (n < N) {              // обробка помилки
  printf("Не вистачає даних у файлі");
  return;
  }
fclose (fp);                 // закрити файл
for (i = 0; i < N; i ++)
  A[i] = A[i] * 2;
fp=fopen("out.dat", "wb"); // відкрити двійковий файл на запис
fwrite (A, sizeof(int), N, fp); // записати весь масив
fclose (fp);                // закрити файл
}
```

Приклад 3.26 – Робота з двійковими файлами

Для читання з двійкового файлу використовується функція `fread`, яка приймає 4 параметри:

- адреса області у пам'яті, куди записати прочитані дані (у цьому випадку це адреса першого елемента масиву `A`, який позначається як `&A[0]` або просто `A`);

- розмір одного елемента даних (краще зробити так, щоб машина сама визначила його, наприклад, у нашому випадку – `sizeof(int)` – розмір цілого числа. Хоча в DevC++ ціле число займає 4 байти, в інших системах програмування це може бути не так; наша програма працюватиме і в цьому випадку, тобто стане переносною на іншу платформу;

- кількість елементів даних у масиві (`N`);

- покажчик на відкритий файл, звідки читати дані (`fp`).

Функція `fread` повертає кількість успішно прочитаних елементів масиву – її значення, що повертається, можна використовувати для обробки помилок. Якщо функція `fread` повернула значення, менше, ніж `N`, у файлі не вистачає даних.

Для запису масиву в двійковий файл використовується функція `fwrite` з такими самими параметрами; вона повертає кількість успішно записаних елементів.

Перевага цього способу полягає в тому, що масив читається і записується відразу єдиним блоком. Це дозволяє значно збільшити швидкість запису на диск (порівняно з виведенням у текстовий файл кожного елемента окремо).

Простий пошук у масиві

Часто зустрічаються завдання, де потрібно послідовно перебрати усі елементи масиву і знайти потрібні нам. Далі розглянемо чотири таких завдання:

1. Пошук одного заданого елемента;
2. Виведення всіх елементів, які задовольняють задану умову;
3. Формування нового масиву з усіх відібраних елементів;
4. Пошук мінімального (максимального) елемента.

Усі ці задачі розв'язуються за допомогою циклу, у якому перебираються всі елементи масиву від початкового (нульового) до кінцевого елемента. Такий пошук називається лінійним, оскільки усі елементи проглядаються послідовно один за одним.

Пошук одного елемента

Нехай нам потрібно визначити, чи є у масиві елемент із заданим значенням x, і, якщо він є, знайти його номер.

Якщо немає жодної інформації про розташування елементів масиву, то застосовується лінійний пошук, основна ідея якого – послідовно переглядати масив, допоки не буде виявлено збіг або не буде досягнено кінця масиву. Це реалізує програма, наведена у прикладі 3.27

```
#include <stdio.h>
const int N = 10;
main()
{
int i, X, A[N];
int success = 0; // змінна-прапор, прапорець скинуто
// тут потрібно ввести масив і X
for ( i = 0; i < N; i ++ )
if ( A[i] == X ) { // якщо знайшли, то...
 success = 1; // встановити прапорець
 break; // вийти з циклу
 }
if ( success )
printf ("A[%d] = %d", i, A[i] );
else
 printf ("Елемент %d не знайдено.", X );
}
```

Приклад 3.27 – Програма пошуку одного заданого елемента у масиві

Для того, щоб визначити ситуацію, коли елемент не знайдено, нам треба ввести спеціальну змінну `success`, яка встановлюється в 1, якщо елемент знайдено, і залишається рівною нулю, якщо у масиві немає потрібного елемента. Така змінна називається прапорцем Прапорець може бути встановлений (дорівнює 1) або скинутий (дорівнює 0).

Для лінійного пошуку в найгіршому випадку ми маємо N порівнянь. Зрозуміло, що для прискорення пошуку треба спочатку якось упорядкувати дані, у цьому разі можна зробити пошук ефективним.

Пошук усіх елементів, що відповідають умові

Нехай нам потрібно визначити, скільки у масиві додатних елементів і вивести їх на екран.

Для розв'язання цієї задачі вводимо лічильник – спеціальну змінну, значення якої буде збільшуватися на одиницю, коли ми знайшли черговий позитивний елемент (приклад 3.28).

```
#include <stdio.h>
const int N = 10;
main()
{
int i, A[N], count = 0; // count- лічильник додатних елементів
               // тут потрібно ввести масив
for (i = 0; i < N; i ++) // цикл по всіх елементах масиву
  if (A[i] > 0) {   // якщо знайшли позитивний, ...
    count ++; // збільшуємо лічильник і ...
    printf ("%d ", A[i]); // виводимо елемент (якщо потрібно)
    }
printf ("\n У масиві %d додатних чисел", count);
}
```

Приклад 3.28 – Програма пошуку у масиві додатних елементів

Формування масиву за заданою умовою

Нехай нам потрібно сформувати новий масив `B`, помістивши у нього всі додатні елементи вихідного масиву `A`, і вивести його на екран.

Нехай є масив `A[N]`. Нам потрібно вибрати з нього всі додатні елементи й записати їх у новий масив `B`. Спершу потрібно визначити, скільки місця у пам'яті потрібно виділити для масиву `B`. У «найгіршому» випадку всі елементи масиву `A` будуть додатними і увійдуть до масиву `B`, тому останній має мати такий самий розмір, що й масив `A`.

Можна запропонувати такий спосіб: переглядати весь масив A, і якщо для чергового елемента `A[i] > 0`, його значення копіюється у `B[i]`. Однак у цьому випадку використовувати такий масив `B` дуже складно, тому що потрібні елементи можуть стояти не підряд (коли `A[i] ≤ 0`).

Є більш красивий спосіб. Оголошуємо тимчасову змінну-лічильник `count`, у якій зберігатимемо кількість знайдених додатних елементів. Спочатку вона дорівнює нулю. Якщо знайшли черговий позитивний елемент, то ставимо його в клітинку `B[count]` і збільшуємо лічильник. Таким чином, усі потрібні елементи стоять на початку масиву `B` (приклад 3.29).

```
#include <stdio.h>
const int N = 10;
main()
{
int i, A[N], B[N], count = 0;
for ( i = 0; i < N; i ++ ) // тут слід ввести масив A
  if ( A[i] > 0 ) {
    B[count] = A[i]; count ++; //можна так: B[count++] = A[i];
    }
printf("\n Результат:\n");
for ( i = 0; i < count; i ++ )
  printf("%d ", B[i]);
}
```

Приклад 3.29 – Створення масиву, що містить лише додатні елементи іншого масиву

Зверніть увагу, що змінна в останньому циклі змінюється від 0 до `count-1` (а не до `N-1`) так, що на екран виводяться тільки реально використовувані (а не всі) елементи масиву `B`.

Мінімальний елемент

Нехай нам потрібно знайти і вивести на екран мінімальний елемент масиву `A`.

Для розв'язання такої задачі спершу потрібно виділити у пам'яті комірку (змінну) для зберігання знайденого мінімального значення. Спочатку ми записуємо в цю комірку перший елемент `A[0]` (приклад 3.30). Потім беремо наступний елемент і порівнюємо його з мінімальним. Якщо він менший за мінімальний, записуємо його значення у клітинку мінімального елемента і т. д. Коли ми розглянемо останній елемент масиву, то у додатковій комірці пам'яті буде мінімальне значення елементів масиву.

```
#include <stdio.h>
const int N = 10;
main()
{
int i, A[N],min;
      // тут слід ввести масив A
   min = A[0]; // припускаємо, що A[0] - мінімальний
for (i = 1; i < N; i++) //цикл по від A[1] до A[N-1]
if (A[i] < min) // якщо A[i] менше min, то
min = A[i]; // запам'ятати A[i] як мінімальний
printf("\n Мінімальний елемент %d", min);
}
```

Приклад 3.30 – Програма пошуку мінімального елемента масиву

Зауважимо, що перебір у циклі починається з елемента з номером 1 (а не 0), оскільки початковий елемент ми розглянули окремо.

Тепер трохи ускладнимо задачу і знайдемо ще й номер мінімального елемента. У такому випадку доцільно створити ще одну змінну, де зберіга-

тимемо номер мінімального елемента. Якщо ми знайшли новий мінімальний елемент, то в одну змінну записуємо його значення, а в другу – його номер. Можна також обійтись одною додатковою змінною, оскільки за номером елемента можна легко знайти його значення у масиві (приклад 3.31). Тут у змінній `nMin` зберігатимемо не значення мінімального елемента, а лише його номер.

```
#include <stdio.h>
const int N = 10;
main()
{
int i, A[N], nMin;        // nMin - номер мінімального елемента
          // тут потрібно ввести масив A
nMin = 0;                   // вважаємо, що A[0] - мінімальний
for ( i = 1; i < N; i ++ ) // цикл по всіх інших елементах
if ( A[i] < A[nMin] )       // якщо A[i]<A[nMin], то
nMin = i;                   // запам'ятати i

printf("\n Мінімальний елемент A[%d]=%d", nMin, A[nMin]);
}
```

Приклад 3.31 – Програма пошуку мінімального елементу масиву і номера цього елемента

Сортування масивів

Сортування – це розстановлення елементів деякого списку в заданому порядку [63].

Існують різні види сортування (за алфавітом, за датами тощо), вони відрізняються лише процедурою порівняння елементів. Розглянемо найпростіший варіант сортування – розстановлення елементів масиву в порядку зростання [67].

Натепер існує дуже багато методів сортування. Вони поділяються на дві групи: 1) зрозумілі, але неефективні; 2) ефективні, але незрозумілі (швидке сортування тощо). Розглянемо один з найпростіших методів сортування – бульбашкове.

Бульбашкове сортування

Назва цього методу походить від відомого фізичного явища – бульбашка повітря у воді піднімається вгору. У цьому методі спочатку піднімається «нагору» (до початку масиву) «найлегший» елемент (мінімальний), потім наступний і т. д.

Спочатку порівнюємо останній елемент із передостаннім. Якщо вони розташовані у неправильному порядку (тобто останній елемент «легший» ніж попередній), то міняємо їх місцями. Далі так само розглядаємо наступну пару елементів і т. д. Коли ми обробили пару (A[0], A[1]), мінімальний елемент займе місце A[0]. Це означає, що на наступних етапах його можна не розглядати (рис. 3.2).

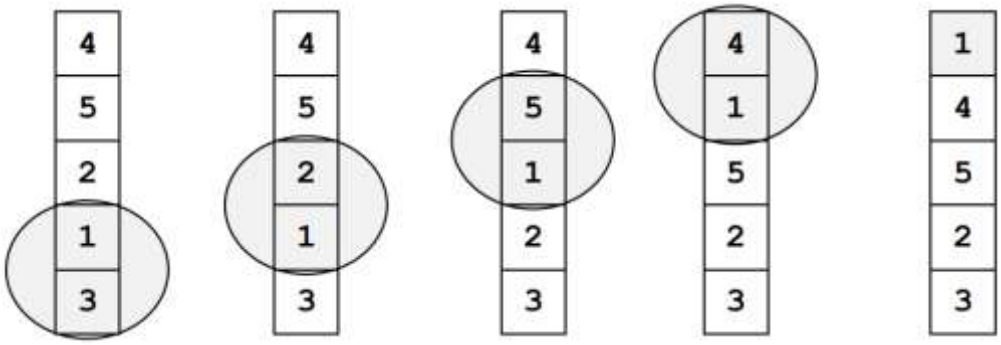


Рисунок 3.2 – Ілюстрація першого кроку роботи бульбашкового сортування елементів масиву

Під час наступного проходу наше завдання – поставити на місце елемент A[1]. Робимо це так само, але вже не розглядаємо A[0], який стоїть на своєму місці. Зробивши N-1 проходів, ми встановимо на місце елементи з A[0] по A[N-2]. Це означає, що останній елемент, A[N-1], уже теж стоїть на своєму місці (приклад 3.32).

```
#include <stdio.h>
const int N = 10;
main()
{
int i, j, A[N], c;
       // тут треба ввести масив A
for (i=0; i<N-1; i++) // достатньо поставити N-1 елементів
 for (j=N-2; j>=i; j--) // йдемо з кінця масиву на початок
  if (A[j] > A[j+1]) // якщо вони розташовані неправильно, то
   {
   c = A[j]; A[j] = A[j+1]; // переставити A[j] і A[j+1]
   A[j+1] = c;
   }
printf("\n Відсортований масив:\n");
for ( i = 0; i < N; i ++ )
  printf("%d ", A[i]);
}
```

Приклад 3.32 – Програма бульбашкового сортування масиву

Бульбашковий метод працює повільно, особливо на великих масивах. Можна показати, що при збільшення розміру масиву у 10 разів час виконання програми збільшується у 100 разів (метод має порядок N^2). На жаль, усі прості алгоритми сортування мають такий (квадратичний) порядок складності.

Метод вибору мінімального елемента

Ще один недолік методу бульбашки полягає в тому, що доводиться занадто часто переставляти місцями сусідні елементи. Цього можна уникну-

ти, якщо використовувати метод вибору мінімального елемента. Його суть така. Шукаємо у масиві мінімальний елемент і ставимо його на перше місце. Потім з решти елементів також шукаємо мінімальний і ставимо на наступне місце тощо (приклад 3.33).

```
#include <stdio.h>
const int N = 10;
main()
{
int i, j, nMin, A[N], c;
      // тут потрібно ввести масив A
for (i=0; i<N-1; i++)
 {
 nMin = i; // шукаємо мінімальний, починаючи з A[i]
 for (j=i+1; j<N; j++)
   if (A[j] < A[nMin])
     nMin = j;
 if (nMin !=i) // якщо мін. ел. не на своєму місці, то
   {
   c = A[i]; A[i] = A[nMin]; // ставимо його на місце
   A[nMin] = c;
   }
 }
printf("\n Відсортований масив:\n");
for (i=0; i<N; i++)
printf("%d ", A[i]);
}
```

Приклад 3.33 – Сортування масиву шляхом вибору

Порівняно з методом бульбашки, цей метод потребує значно менше перестановок елементів (у гіршому випадку N-1). Він дає значний виграш, якщо перестановки складні та займають багато часу.

Двійковий пошук у масиві

Нехай елементи масиву `A` вже розташовані за зростанням і потрібно знайти елемент, що дорівнює `x`, серед елементів з номерами від `L` до `R`. Для цього використаємо таку ідею: обираємо середній елемент між `L` і `R`, він має номер `m=(L+R)/2`, де ділення виконується націло. Порівняємо його з шуканим `x`. Якщо він дорівнює `x`, ми знайшли те, що шукали. Якщо `x` менше `A[m]` – шукаємо між `A[L]` і `A[m]`, а якщо більше `A[m]` – шукаємо між `A[m]` і `A[R]` (приклад 3.34).

Змінна `flag` служить для того, щоб визначити, знайшли ми потрібний елемент чи ні. Якщо знайшли елемент, що дорівнює `x`, треба присвоїти цій змінній значення 1 і вийти з циклу. При цьому у змінній m залишається номер знайденого елемента.

```
#include <stdio.h>
const int N = 10;
main()
{
int L = 0, R = N-1, m, A[N],
   flag = 0; // змінна-прапор, знайшли (1), ні (0)
 // тут треба ввести масив A і відсортувати його за зростанням
printf("Введіть шуканий елемент\n");
scanf("%d", &x);
while (L <= R) {
  m = (L + R) / 2; // середина інтервалу
  if (A[m] == x) { // знайшли потрібний елемент
    flag = 1; // встановити прапор
    break; // вийти з циклу
    }
  if (x < A[m]) R=m-1; // звужуємо межі області пошуку
  else L=m+1;
  }
if (flag)
 printf ("Знайшли: A[%d]=%d", m, A[m] );
else printf ("Такого елемента немає" );
}
```

Приклад 3.34 – Програма двійкового пошуку у масиві

Якщо масив малий, то швидкість двійкового пошуку незначно відрізняється від лінійного. Але чим більше розмір сортованого масиву – тим більше відрізнятиметься швидкість пошуку (табл. 3.9). Уявімо собі, наприклад, що розмір масиву 10^6 елементів і потрібний нам елемент стоїть у кінці масиву (це найгірший варіант для лінійного пошуку).

Таблиця 3.9 – Залежність кількості порівнянь від розміру масиву

Розмір масиву, N	Число порівнянь	
	Лінійний пошук	Двійковий пошук
10	≤ 10	≤ 4
1000	≤ 1000	≤ 10
1000000	≤ 1000000	≤ 20
N	$\leq N$	$\leq \log_2 N$

Таким чином, чим більше елементів у масиві, тим вигідніше використовувати двійковий пошук, оскільки число операцій зростає як логарифм N, тобто повільніше, ніж збільшується розмір масиву. Його недоліком є те, що елементи мають бути заздалегідь відсортовані. Двійковий пошук використовується під час пошуку інформації у великих базах даних.

3.7 Застосування процедур

Процедура – це допоміжна програма (підпрограма), призначена для виконання певних дій, що зустрічаються у кількох місцях програми [64].

Процедура оформляється майже так само, як і основна програма, тільки її ім'я – не `main`, а інше. Вона складається з заголовка, після якого усередині фігурних дужок записують тіло процедури, тобто команди, які виконуються під час її виклику. Ці команди мають уже бути відомі транслятору [65].

У програмі, що наведена у прикладі 3.35, процедура `add()` отримує два цілочислові аргументи x та y, обчислює їх суму і виводить результат на консоль. Функція `main()` пропонує користувачеві ввести два числа, зчитує їх з консолі за допомогою `scanf()`, а потім викликає процедуру `add()`, передаючи їй ці два числа як аргументи. Процедура `add()` не повертає ніякого значення, а просто виводить результат у консоль.

Зрозуміло, що викликати процедуру можна стільки разів, скільки потрібно.

```
#include <stdio.h>
void add(int x, int y) {// Процедура обчислення суми двох чисел
    int sum = x + y;
    printf("Сума %d та %d = %d\n", x, y, sum);
}
int main() {
    int num1, num2;
    printf("Введіть перше число: ");
    scanf("%d", &num1);
    printf("Введіть перше число: ");
    scanf("%d", &num2);
    add(num1, num2); // Виклик процедури add()
    return 0;
}
```

Приклад 3.35 – Програма, що використовує процедуру пошуку суми двох чисел

У програмі, що наведена в прикладі 3.36, ми визначаємо трохи складнішу процедуру `printMessage`, яка отримує цілочисловий аргумент `n` і виводить на консоль повідомлення "`Hello, world!`" `n` разів за допомогою циклу `for`.

У головній функції ми використовуємо `scanf` для зчитування цілочислового `n` від користувача, який потім передається у процедуру `printMessage`. Процедура викликається один раз, у результаті чого повідомлення "`Hello, world!`" буде надруковано на консолі вказану користувачем кількість разів.

```
#include <stdio.h>
void printMessage(int n) {
  int i;
  for (i = 0; i < n; i++) {
    printf("Hello, world!\n");
  }
}

int main() {
  int numRepeats;
  printf("Введіть скільки разів потрібно надрукувати рядок: ");
  scanf("%d", &numRepeats);
  printMessage(numRepeats);
  return 0;
}
```

Приклад 3.36 – Програма, що використовує процедуру друкування рядка слів, задану користувачем кількість разів

Програма демонструє, як процедури можна використовувати для багаторазового виконання певної підзадачі, причому вхідні дані користувача можуть впливати на поведінку процедури.

Взагалі, можна констатувати нижчевказане.

- Процедура оформляється так само, як основна програма: заголовок і тіло процедури має бути у фігурних дужках.

- Перед іменем процедури ставиться слово void. Це означає, що вона призначена не для обчислень, а для виконання деяких дій.

- Після імені у дужках через кому (якщо вони є) перераховуються параметри процедури (ті величини, від яких залежить її робота). Параметри іноді називають аргументами процедури.

- Для кожного параметра окремо вказується його тип (int, float, char).

- Імена параметрів можна вибирати будь-які, допустимі в мові Сі.

- Параметри, перераховані у заголовку процедури, називаються формальними. Це означає, що вони доступні лише всередині процедури під час її виклику.

- Потрібно вибирати осмислені імена параметрів процедури, оскільки це дає змогу легше розібратися у програмі потім, коли вже все забуто.

- Під час виклику процедури треба вказати її ім’я і в дужках фактичні параметри, які підставляються у процедурі замість формальних параметрів (якщо такі існують).

- Фактичні параметри – це числа або будь-які арифметичні вирази (у цьому разі спочатку розраховується їхнє значення).

- Перший фактичний параметр підставляється у процедурі замість першого формального параметра, і т. д.

- Процедура має бути оголошена до основної програми для того, щоб до моменту виклику процедури транслятор знав, що є така процедура, а також скільки вона має параметрів і яких. Це дає змогу знаходити помилки на етапі трансляції, наприклад, занадто мало параметрів, занадто багато тощо.

- Часто процедури викликаються лише один раз. У цьому випадку їхнє завдання – розбити велику основну програму (або процедуру) на кілька самостійних частин, оскільки рекомендується, щоб кожна процедура була завдовжки не більше 50 рядків (2 екрани по 25 рядків), інакше в ній досить складно розібратися.

- Для дострокового виходу з процедури використовується оператор `return`, при його виконанні робота процедури завершується.

- У процедурі можна використовувати кілька операторів `return`: під час виконання будь-якого з них робота процедури завершується.

3.8 Застосування функцій

Розглянемо чим функції відрізняються від процедур. Функції, як і процедури, призначені для виконання однакових операцій у різних частинах програми. Проте вони мають одну суттєву відмінність: завдання функції – обчислити і повернути у програму, що викликає, значення-результат (у найпростішому випадку це ціле, дійсне або символьне значення) [69].

Функція – це допоміжна програма (підпрограма), призначена для отримання деякого об’єкта-результату (наприклад, числа). Вона також може виконувати якісь корисні дії [64].

Покажемо використання функції на прикладі. Нехай нам потрібно написати програму, яка вводить ціле число і визначає суму його цифр із використанням функції обчислення такої суми.

Для знаходження останньої цифри числа потрібно взяти залишок від його ділення на 10. Потім потрібно поділити число на 10, відкидаючи його останню цифру, тощо. Склавши усі ці залишки-цифри, ми і отримаємо шукану суму цифр числа (приклад 3.37).

```
#include <stdio.h>
int SumDigits ( int N ) // заголовок функції
{ // початок функції
int d, sum = 0;
while ( N != 0 )
 {
 d = N % 10; // тіло функції
 sum = sum + d;
 N = N / 10;
 }
return sum; // функція повертає значення sum
} // кінець функції
main()
{
int N, s;
printf ("\nВведіть ціле число ");
scanf ("%d", &N );
s = SumDigits (N); // виклик функції
printf ("Сума цифр числа %d дорівнює %d\n", N, s );
getchar();
}
```

Приклад 3.37 – Програма обчислення суми цифр натурального числа з використанням функції

- Функція оформляється так само, як процедура: заголовок і тіло функції мають бути у фігурних дужках.

- Перед іменем функції ставиться тип результату (`int`, `float`, `char` тощо). Це означає, що вона повертає значення зазначеного типу.

- Після імені в дужках через кому перераховуються параметри функції – тобто, ті величини, від яких залежить її робота.

- Для кожного параметра окремо вказується його тип (`int`, `float`, `char`).

- Імена параметрів можна вибирати будь-які, допустимі у мові Сі.

- Параметри, перелічені у заголовку функції, називаються формальними – це означає, що вони доступні лише всередині функції під час її виклику.

- Варто вибирати осмислені імена параметрів, що дає змогу легше розібратися у програмі потім.

- Під час виклику функції треба вказати її ім'я і в дужках фактичні параметри, які використовуються всередині функції замість формальних параметрів.

- Фактичні параметри – це числа або будь-які арифметичні вирази (у цьому разі спочатку розраховується їхнє значення).

- Перший фактичний параметр використовується всередині функції замість першого формального параметра і т. д.

- Для того, щоб визначити значення функції, використовується оператор `return`, після якого через проміжок записується значення, що повертається, тобто, число або арифметичний вираз (нариклад, `return 87`; `return s`; `return a + 6*b - 2`).

Піл час виконання оператора `return` робота функції завершується.

- У функції можна використовувати кілька операторів `return`.

- Якщо функції розташовані нижче основної програми, їх необхідно оголосити до основної програми. Для оголошення функції треба написати її заголовок із крапкою з комою у кінці.

- Під час оголошення функції після заголовка ставиться крапка з комою, а у тому місці, де записано тіло функції – не ставиться.

Логічні функції

Досить часто треба скласти функцію, яка просто вирішує якесь питання і відповідає на запитання «Так» або «Ні». Такі функції називаються логічними. Згадаймо, що у мові С нуль означає хибну умову, а одиниця – істинну [62].

Логічна функція – це функція, що повертає 1 (якщо відповідь «Так») або 0 (якщо – «Ні»).

Логічні функції використовуються, головним чином, у двох випадках:

- якщо потрібно проаналізувати ситуацію і відповісти на запитання, від якого залежать подальші дії програми;

- якщо потрібно виконати деякі складні операції та визначити, чи була при цьому якась помилка.

Нехай перед нами поставлене таке завдання: ввести число N і визначити, просте воно чи ні. При цьому потрібно використати функцію, яка відповідає на це запитання.

Тепер розташуємо тіло функції нижче основної програми. Для того, щоб транслятор знав про цю функцію під час обробки основної програми, треба оголосити її заздалегідь (приклад 3.38).

```
#include <stdio.h>
int Prime (int N); // оголошення функції
main()
{
int N;
printf ("\nВведіть ціле число ");
scanf ("%d", &N);
if ( Prime(N) ) // виклик функції
 printf ("Число %d - просте\n", N );
else printf ("Число %d - складене\n", N );
getchar();
}
int Prime (int N) // опис функції
{
for (int i=2; i*i<=N; i++)
 if (N % i == 0 ) return 0; // знайшли дільник - складене!
return 1; // не знайшли жодного дільника - просте!
}
```

Приклад 3.38 – Програма, що використовує логічну функцію

Функції, що повертають два значення

За означенням функція може повернути лише одне значення (результат). Якщо треба повернути два і більше результатів, доводиться використовувати спеціальний прийом – передачу параметрів за посиланням [66].

Нехай, наприклад, потрібно написати функцію, яка визначає максимальне та мінімальне з двох цілих чисел.

У програмі, наведеній у прикладі 3.39, використовується досить цікавий прийом: ми зробимо так, щоб функція змінювала значення змінної, яка належить основній програмі. Один результат (мінімальне з двох чисел) функція поверне як зазвичай, а другий – за рахунок зміни змінної, яка передана з основної програми.

```
#include <stdio.h>
int MinMax ( int a, int b, int &Max )
{
if ( a > b ) { Max = a; return b; }
else { Max = b; return a; }
}
main()
{
int N, M, min, max;
printf ("\nВведіть 2 цілих числа");
scanf ("%d%d", &N, &M );
min = MinMax ( N, M, max ); // виклик функції
printf ("Найменше з них %d, найбільше - %d\n", min, max );
getchar();
}
```

Приклад 3.39 – Програма, яка містить функцію, що повертає два значення

Зазвичай під час передачі параметра у процедуру або функцію у пам’яті створюється копія змінної, і функція працює з такою копією. Це означає, що всі зміни змінної-параметра, зроблені у функції, не позначаються на значенні цієї змінної у програмі, що викликає.

Якщо перед іменем параметра у заголовку функції поставити знак `&` (він також використовується для визначення адреси змінної), то функція працює прямо зі змінною з програми, що викликає, а не з її копією. Тому в нашому прикладі функція змінить значення змінної `max` з основної програми і запише туди максимальне з двох чисел.

Цей прийом можна використовувати і для процедур: хоча формально вони не повертають жодного значення-результату, можна все-таки передавати дані у програму, що викликає, через змінювані параметри.

Отже, резюмуємо.

- Якщо потрібно, щоб функція повернула два і більше результатів, роблять так:

  - один результат передається як зазвичай, за допомогою оператора `return`.

  - інші значення, що повертаються, передаються через змінювані параметри.

- Звичайні параметри не можуть змінюватися підпрограмою, тому що вона працює з копіями параметрів (наприклад, якщо змінювати значення `a` і `b` у функції `MinMax`, відповідні їм змінні `N` і `M` в основній програмі не зміняться).

- Будь-яка процедура та функція може повертати значення через змінювані параметри.

- Змінні параметри (або параметри, що передаються за посиланням) оголошуються у заголовку підпрограми спеціальним чином: перед їх ім’ям ставиться знак `&` – у цьому випадку він означає посилання, тобто підпрограма може змінювати значення параметра (у цьому випадку функція змінює значення змінної `max` в основній програмі).

- Під час виклику таких функцій і процедур замість кожного фактичного змінюваного параметра треба підставляти тільки ім’я змінної (не число і не арифметичний вираз – у таких випадках транслятор видає попередження і формує у пам’яті тимчасову змінну).

3.9 Рекурсія

Рекурсія – це визначення об’єкта через самого себе [64].

Гарним візуальним прикладом рекурсії є картинка, що містить своє зображення (рис. 3.3).

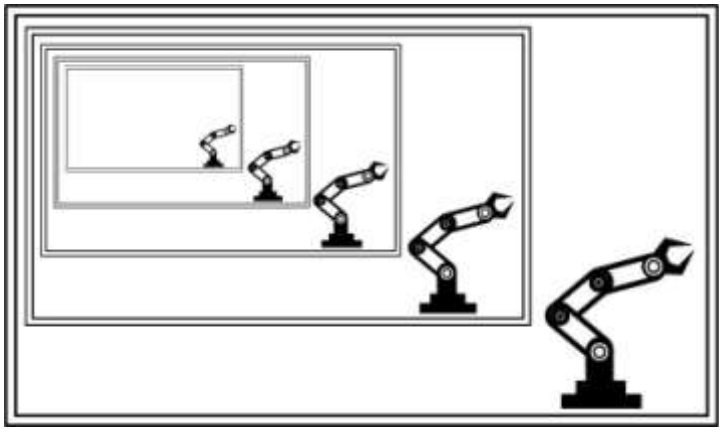


Рисунок 3.3 – Картинка, що містить своє зображення

Цікаво також згадати оповідання про машину, яка була досить розумною та лінивою, щоб для розв’язання поставленої перед неї задачі побудувати собі подібну та доручити їй розв’язування цієї задачі. У результаті маємо нескінченну рекурсію, де кожна нова машина будувала собі подібну і передавала їй розв’язування задачі.

Досить широке застосування рекурсія набула у математиці. Так, наприклад, за її допомогою визначають багато нескінченних множин. Ось приклад задання множини натуральних чисел:

Натуральне число.

1) 1 – натуральне число.

2) Число, наступне за натуральним – натуральне.

Також рекурсивно можна задати і факторіал n! = 1·2·3· ... ·(n-1)·n

Факторіал.

1) 0! = 1 (так умовно прийнято).

2) n! = n·(n-1)!

Рекурсивні процедури та функції

Рекурсивними називаються процедури та функції, які викликають самі себе. Наприклад, функцію обчислення факторіала можна записати, як наведено у прикладі 3.40.

```
int Factorial (int n)
  {
  if (n <= 0) return 1; // повернути 1
  else return n*Factorial(n-1); // рекурсивний виклик
  }
```

Приклад 3.40 – Функція обчислення факторіала

Тут функція `Factorial` викликає сама себе, якщо `n > 0`.

Для розв’язання цієї задачі можна використовувати і рекурсивну процедуру, а не функцію. Згадаймо, як рекурсивна процедура може повернути значення-результат? Через параметр, переданий за посиланням (в оголо-

шенні процедури біля його імені стоїть знак посилання `&`). При рекурсивних викликах процедура змінює це значення (приклад 3.41).

```
void Factorial ( int n, int &fact )
 {
 if ( n == 0 ) fact = 1; // рекурсія закінчилася
 else {
  Factorial(n-1, fact); // рекурсивний виклик, рахуємо (n-1)!
  fact *= n; // n! = n*(n-1)!
  }
 }
```

Приклад 3.41 – Процедура обчислення факторіала

На відміну від функції процедура може (якщо потрібно) повертати кілька значень результатів за допомогою параметрів, що передаються за посиланням.

Непряма рекурсія

Рідше використовується складніша конструкція, коли процедура викликає сама себе не безпосередньо, а через іншу процедуру (або функцію). Це описується схемою, що наведена на рис. 3.4

Такий прийом називається непрямою рекурсією.

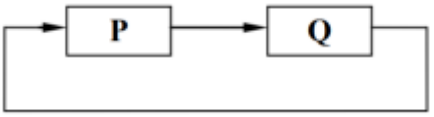


Рисунок 3.4

Під час використання рекурсивних процедур і функцій є велика небезпека, що вкладені виклики триватимуть нескінченно (це схоже на зациклення циклу `while`). Тому в таких функціях необхідно передбачити умову, яка перевіряється на кожному кроці і закінчує вкладені виклики, коли перестає виконуватися.

Для функції обчислення факторіала такою умовою є `n <= 0`. Доведемо, що рекурсія у прикладі, розглянутому вище, закінчиться.

1. Рекурсивні виклики закінчуються, коли n стає рівним нулю.

2. При кожному новому рекурсивному виклику значення n зменшується на 1 (це випливає з того, що викликається `Factorial(n-1,...)`).

3. Тому, якщо на початку `n > 0`, то, поступово зменшуючись, `n` досягає значення `0`, і рекурсія завершується.

Рекурсивна процедура або функція має містити умову, за якої рекурсія завершується (не відбувається нового вкладеного виклику).

Коли рекурсія не потрібна

При новому рекурсивному виклику комп'ютер: по-перше – запам'ятовує стан обчислень на даному етапі; по-друге – у стеку (особливій ділянці пам'яті) створює новий набір локальних змінних (для того, щоб не зіпсувати змінні поточного виклику). Оскільки під час кожного виклику витрачається нова пам'ять і витрачається час на виклик процедури та по-

вернення з неї, під час використання рекурсії потрібно пам'ятати, що глибина рекурсії (кількість вкладених викликів) має була досить мала.

Програма, що використовує рекурсію з великою глибиною, буде виконуватися довго і може викликати переповнення стека (брак стекової пам'яті). Тому якщо завдання може бути легко вирішене без використання рекурсії, рекурсію використовувати небажано.

Наприклад, задача обчислення факторіала дуже просто розв'язується за допомогою звичайного циклу `for` (приклад 3.42). Таке розв'язання за допомогою циклів називають ітеративним, циклічним.

```
int Factorial ( int n )
 {
 int i, fact = 1;
 for ( i = 2; i <= n; i ++ )
   fact *= i;
 return fact;
 }
```

Приклад 3.42 – Функція обчислення факторіала, що працює набагато швидше за рекурсивну

Доведено, що будь-яка рекурсивна програма може бути написана без використання рекурсії, хоча така реалізація може виявитися дуже складною.

Наприклад, якщо нам потрібно скласти функцію для обчислення чисел Фібоначчі f_i, які задаються так:

1. $f_0 = 0$, $f_1 = 1$.
2. $f_i = f_{i-1} + f_{i-2}$ для $i > 1$.

Використання рекурсії «у лоб» дає функцію, що наведена у прикладі 3.43.

```
int Fib ( int n )
  {
  if ( n == 0 ) return 0
  if ( n == 1 ) return 1;
  return Fib(n-1) + Fib(n-2);
  }
```

Приклад 3.43 – Рекурсивне обчислення чисел Фібоначчі

Зауважимо, що кожен рекурсивний виклик при `n > 1` породжує ще 2 виклики функції, багато виразів (числа Фібоначчі для малих `n`) обчислюються багато разів. Тому практичного значення цей алгоритм не має, особливо для великих `n`. Схему обчислення `Fib(5)` наведено у вигляді дерева на рис. 3.5.

Зауважимо, що чергове число Фібоначчі залежить тільки від двох попередніх, які будемо зберігати у змінних `f1` і `f2`. Спочатку приймемо `f1=1` і `f2=0`, потім обчислюємо наступне число Фібоначчі і записуємо його у

змінну x (приклад 3.44). Тепер значення `f2` вже не потрібне і ми скопіюємо `f1` у `f2` і `x` у `f1`.

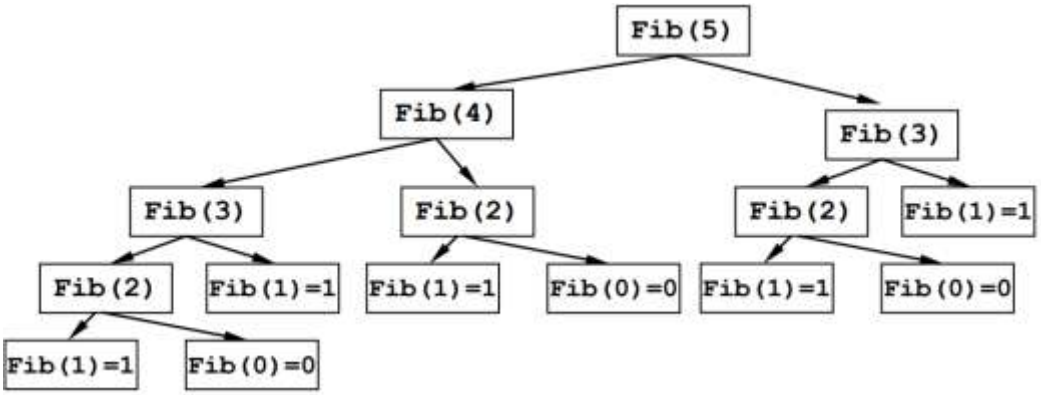


Рисунок 3.5 – Схема обчислення `Fib(5)`

```
int Fib2(int n)
{
  int i, f1 = 1, f2 = 0, x;
  for (i = 2; i <= n; i ++) {
    x = f1 + f2; // наступне число
    f2 = f1; f1 = x; // зсув значень
    }
  return x;
}
```

Приклад 3.44 – Ефективна процедура обчислення чисел Фібоначчі

Така процедура для великих n (>20) працює в сотні тисяч разів швидше, ніж рекурсивна!

Отже, можемо зробити висновок: там, де можна легко обійтися без рекурсії, треба без неї обходитися.

Розглянемо ще один приклад реалізації рекурсії на прикладі пошуку.

Нехай нам потрібно визначити, скільки разів зустрічається задане слово у реченні. Рекурсивна процедура буде така:

1) шукаємо перше задане слово за допомогою функції strstr; якщо не знайшли, то зупинка;

2) кількість слів = 1 + кількість слів у частині рядка, що залишилася.

```
int HowMany(char *s, char *word)
{
 char *p = strstr(s, word); // шукаємо перше слово
 if (!p) return 0; // не знайшли (тобто 0 слів)
 return 1 + // одне вже знайшли, ...
 HowMany(p+strlen(word),word); // шукаємо далі
}
```

Приклад 3.45 – Функція визначення кількості заданого слова у реченні

Функція вийшла досить коротка та зрозуміла, але за швидкістю роботи не найефективніша [69].

3.10 Контрольні питання

1. Поясніть синтаксис оголошення змінних у мові С та наведіть основні підтримувані типи даних.
2. Поясніть різницю між типами даних int, float, double та char у С.
3. Поясніть, як оголошуються та використовуються масиви у мові С, і які типові операції виконуються над масивами.
4. Наведіть призначення вказівників у мові С, та поясніть як вони оголошуються та перенаправляються.
5. Опишіть концепцію динамічного виділення пам’яті у мові С за допомогою функцій malloc, calloc та realloc.
6. Поясніть, чим відрізняються оператори == та = у мові С та як вони використовуються.
7. Поясніть використання операторів if, else if та else для прийняття рішень у мові С.
8. Поясніть, як використовуються цикли for, while та do-while для ітерації у мові С, і в чому полягають їх відмінності.
9. Обговоріть роль функцій у програмуванні мовою С та поясніть процес визначення та виклику функцій.
10. Поясніть, що таке прототипи функцій і навіщо вони потрібні в програмуванні мовою С.
11. Поясніть поняття рекурсії у мові С та наведіть приклади рекурсивних функцій.
12. Поясніть, як визначаються та використовуються структури у мові С, і які переваги дає використання цих структур.
13. Обговоріть концепцію роботи з файлами у мові С, охоплюючи відкриття, читання, запис та закриття файлів.
14. Поясніть, яке призначення ключового слова typedef у мові С і як воно використовується для визначення користувацьких типів даних.
15. Опишіть різницю між передачею за значенням та передачею за посиланням в аргументах функцій мови С.
16. Поясніть, що таке заголовні файли у мові С і як вони використовуються для організації коду та полегшення модульного програмування.
17. Поясніть роль директив препроцесора у мові С, зокрема #include, #define та #ifdef.
18. Поясніть, як використовуються побітові оператори (&, |, ^, <<, >>, ~) у мові С та яке їхнє застосування.
19. Обговоріть поняття функціональних вказівників у мові С та наведіть приклади їх використання.
20. Опишіть процес компіляції та виконання програми на С, у тому числі використання компіляторів та компонувальника (лінкера).

ГЛОСАРІЙ

Актуатор – пристрій, який перетворює електричну або пневматичну енергію в механічний рух для маніпулювання фізичним середовищем робота. Приклади містять двигуни, сервоприводи, пневматичні циліндри та гідравлічні приводи.

Алгоритми розгалуження – конструкції, що керують ходом виконання програми, зокрема оператори if, else, switch, for, while та do-while.

Ардуіно – апаратна обчислювальна платформа для аматорського конструювання, основними компонентами якої є плата мікроконтролера з елементами введення / виведення та середовище розробки Processing/Wiring мовою програмування, що є спрощеною підмножиною C/C++.

Атрибут – параметр файлу, що визначає певні властивості файлів і встановлюється як властивість конкретного файла.

Бібліотека – об'єктів чи підпрограм для вирішення близьких за тематикою задач.

Біт – мінімальна одиниця кількості інформації, яка дорівнює одному двійковому розряду, який може бути рівним одному з двох значень/станів, застосовуваних для подання даних у двійковій системі числення.

Бітові оператори – оператори, які виконують побітові операції над операндами на двійковому рівні, зокрема & (AND), | (OR), ^ (XOR), << (зсув вліво), >> (зсув вправо) та ~ (побітове NOT).

Взаємодія людина-робот (HRI) – вивчення того, як люди і роботи спілкуються, співпрацюють і взаємодіють один з одним, охоплюючи фізичну і когнітивну взаємодію.

Виконуваний файл – файл, що містить інструкції машинного коду, згенеровані компілятором і компонувальником, які можуть виконуватися безпосередньо процесором комп'ютера.

Вікно – візуально відокремлена область екрана з певним інтерфейсом користувача. Зазвичай вікно має прямокутну форму. В ньому відображаються елементи введення і, можливо, виведення для одного або декількох процесів. Вікнами можна керувати за допомогою курсора миші та клавіатури.

Вказівник – змінна, яка зберігає адресу пам'яті іншої змінної. Вказівники використовуються для динамічного виділення пам'яті, передачі за посиланням, роботи з масивами і структурами.

Вказівник функції – вказівник, який вказує на адресу функції. Вказівники на функції використовуються для реалізації зворотних викликів, динамічного виклику функцій та поліморфізму.

Динамічне виділення пам'яті – процес виділення пам'яті під час виконання програми за допомогою таких функцій, як malloc, calloc і realloc для динамічного керування пам'яттю.

Директива препроцесора – Інструкції препроцесору C, які змінюють вихідний код перед компіляцією, такі як #define, #include та #ifdef.

Драйвер – комп'ютерна програма, за допомогою якої операційна система отримує доступ до певного приладу чи частини апаратного забезпечення.

Дрон – робот, здатний до польоту, зокрема дрони та безпілотні літальні апарати (БПЛА), що використовуються для повітряного спостереження, картографування та моніторингу.

Енергоефективність – оптимізація споживання енергії роботом з метою максимізації часу його роботи та зменшення впливу на навколишнє середовище, що досягається за рахунок ефективного дизайну, алгоритмів керування та стратегій управління живленням.

Заголовний файл – файл, що містить прототипи функцій C, макроозначення та декларації, зазвичай розміщують на початку вихідних файлів за допомогою директиви #include.

Змінна – іменована комірка в пам'яті, яка використовується для зберігання значень даних у програмі C. Змінні мають бути оголошені з певним типом даних.

Інтегроване середовище розробки (IDE) – комплексне програмне рішення для розробки програмного забезпечення. Зазвичай, складається з редактора початкового коду, інструментів для автоматизації складання та налаштування програм.

Інтерфейс апаратний – сукупність алгоритмів обміну і технічних засобів, що забезпечують обмін інформацією між пристроями. Апаратний інтерфейс є системою шин, роз'ємів, узгоджувальних пристроїв, алгоритмів і протоколів, що забезпечують зв'язок всіх частин мікропроцесорної системи між собою. Від характеристик інтерфейсу залежить швидкодія і надійність системи. У розгорнутих мікропроцесорних системах для розвантаження процесора апаратний інтерфейс забезпечується контролерами.

Інтерфейс користувача – різновид інтерфейсу, в якому одна сторона – це людина (користувач), інша – машина/пристрій. Це сукупність засобів і методів, за допомогою яких користувач взаємодіє з різними, найчастіше складними, машинами, пристроями, апаратурою і програмним забезпеченням. Часто термін застосовується щодо комп'ютерних програм, однак під ним може матися на увазі набір засобів, методів і правил взаємодії будь-якої системи, керованої людиною. Інтерфейс вважається двонаправленим (інтерактивним), якщо пристрій, отримавши команди від користувача і виконавши їх, видає інформацію користувачу наявними у нього засобами – візуальними, звуковими, тактильними тощо. Інтерфейс – це сукупність елементів, які також можуть складатися з елементів.

Інтерфейс програмний – набір готових класів, процедур, функцій, структур і констант, що надаються інформаційною системою (бібліотекою, сервісом) для використання у зовнішніх програмних продуктах.

Використовується програмістами для написання власних програмних засобів.

Кінематика роботів – вивчення руху роботизованих систем без урахування сил, що спричиняють рух, зосереджуючись на положенні, швидкості та прискоренні.

Колісний робот – робот, який використовує колеса для пересування, підходить для таких рівних і гладких поверхонь, як внутрішні приміщення і дороги.

Компілятор – програма, яка перекладає вихідний код, написаний мовою С, у машинний код або проміжний код, який може бути виконаний комп'ютером.

Лінкер (компонувальник) – програма, яка об'єднує об'єктні файли, згенеровані компілятором, для створення виконуваної програми шляхом розв'язання зовнішніх посилань і бібліотечних залежностей.

Локалізація – процес визначення положення та орієнтації робота відносно навколишнього середовища, часто з використанням таких датчиків, як GPS, лідар та візуальна одометрія.

М'яка робототехніка – галузь робототехніки, яка фокусується на проєктуванні та створенні роботів з м'якими тілами, що деформуються, за зразком таких біологічних організмів, як восьминоги та черв'яки.

Масив – набір елементів одного типу даних, що зберігаються в суміжних комірках пам'яті, доступ до яких здійснюється за допомогою індексу.

Машинне навчання – підрозділ штучного інтелекту, який дозволяє комп'ютерам навчатися на основі даних і підвищувати ефективність виконання конкретних завдань без явного програмування.

Мейкблок (mBlock) – платформа коду Arduino на основі Scratch, яка дозволяє створювати проєкти за допомогою блоків Scratch.

Мікроконтролер (або однокристальний мікрокомп'ютер) – виконаний у вигляді мікросхеми спеціалізований комп'ютер, що містить мікропроцесор, оперативну та постійну пам'ять для збереження виконуваного коду програм і даних, порти введення-виведення та блоки зі спеціальними функціями.

Мова програмування С – процедурна мова програмування загального призначення, розроблена Деннісом Рітчі на початку 1970-х років у Bell Labs. С широко використовується для розробки системного та прикладного програмного забезпечення.

Оператори – символи, якими подаються операції, що виконуються над операндами. Прикладами у С є арифметичні оператори (+, -, *, /), реляційні оператори (==, !=, <, >, <=, >=) та логічні оператори (&&, ||, !).

Пневмопривод – виконавчий механізм, який перетворює енергію стиснутого повітря в механічну у вигляді лінійного або обертального руху.

Порт мікроконтролера – пристрої введення / виведення, що дозволяють мікроконтролеру передавати або приймати дані. Стандартний порт

мікроконтролера AVR має вісім розрядів даних, які можуть передаватися або прийматися паралельно. Кожному розряду (або біту) відповідає вивід (ніжка) мікроконтролера.

Послідовний порт (COM порт) – двонаправлений послідовний інтерфейс, призначений для обміну байтовою інформацією.

Привод – пристрій, що надає рух будь-якій машині, механізму.

Пристрій виведення інформації – периферійний пристрій для виведення інформації для людей і в зрозумілій для людей формі.

Програма – послідовність інструкцій, призначених для виконання пристроєм управління обчислювальної машини. Програма – один з компонентів програмного забезпечення. Залежно від контексту цей термін може також стосуватися і вихідних текстів програми.

Програматор – апаратно-програмний пристрій, призначений для запису / зчитування інформації на постійному запам'ятовувальному пристрої.

Програмне забезпечення – сукупність програм, процедур і правил, а також документації, що стосуються функціонування системи оброблення даних з використанням електронних обчислювальних машин.

Промисловий робот – робот, який використовується у таких виробничих процесах, як зварювання, фарбування, складання та пакування, для автоматизації повторюваних завдань на виробничій лінії.

Редуктор (механічний) – механізм на основі однієї або більше передач зачепленням, що входить у приводи машин і слугує для збільшення крутного моменту разом із зниженням кутової швидкості веденого вала.

Рекурсія – техніка програмування, коли функція прямо чи опосередковано викликає сама себе для вирішення проблеми.

Робот – програмована машина, призначена для виконання завдань автономно або під дистанційним керуванням, зазвичай для виконання нудних, брудних або небезпечних для людини завдань.

Робот на ногах – робот з ногами для пересування, створений за зразком таких біологічних організмів, як тварини, здатний пересуватися по складній місцевості і нерівних поверхнях.

Робота з файлами – процес читання і запису до файлів у C за допомогою функцій fopen, fclose, fread, fwrite, fprintf і fscanf.

Робототехніка – міждисциплінарна галузь, яка охоплює проєктування, конструювання, експлуатацію та використання роботів для автономного або напівавтономного виконання завдань.

Ройова робототехніка – галузь робототехніки, яка вивчає координацію та співпрацю декількох роботів для досягнення спільної мети, створена за зразком колективної поведінки соціальних комах.

Сенсор – пристрій, який виявляє і реагує на фізичні подразники або зміни в навколишньому середовищі, забезпечуючи зворотний зв'язок з системою управління роботом. Приклади містять камери, лідар, ультразвукові датчики та інерційні вимірювальні пристрої (IMU).

Сервісний робот – робот, призначений для допомоги людині в непромисловому середовищі, охоплюючи домашніх роботів, роботів у сфері охорони здоров'я та персональних асистентів.

Сервопривод – пристрій в системах автоматичного регулювання або дистанційного керування, що за рахунок енергії допоміжного джерела здійснює механічне переміщення регулювального органу відповідно до отримуваних від системи керування сигналів.

Синтаксис мови С – набір правил, що визначає комбінації символів, які вважаються правильно структурованими програмами або виразами у С.

Скетч – програма, написана для платформи Arduino, має певну структуру. Скетч обов'язково містить 2 функції: функцію Setup і функцію Loop.

Структура – визначений користувачем тип даних у мові С, який дозволяє групувати змінні під одним іменем. Структури можуть містити змінні різних типів даних.

Ступені свободи (DOF) – кількість незалежних параметрів, які визначають конфігурацію робота, також поступальні та обертальні рухи вздовж різних осей.

Телеуправління – керування роботом на відстані людиною-оператором, що часто використовується в небезпечних умовах або для виконання завдань, які потребують людського досвіду.

Типи даних – категорії значень, які визначають можливі операції над цими значеннями та спосіб їх зберігання в пам'яті. Прикладами у мові С є int, float, double та char.

Функція – самодостатній блок коду, який виконує певну задачу. Функції забезпечують модульність та багаторазове використання у програмах мовою С.

Цикл – різновид керівної конструкції у високорівневих мовах програмування, призначений для організації багаторазового виконання набору інструкцій. Також циклом може називатися будь-яка багатократно виконувана послідовність команд, організована будь-яким чином.

ШІМ (PWM) – широтно-імпульсна модуляція. Сигнал змінної шпаруватості, але постійної частоти.

Штучний інтелект (ШІ) – імітація процесів людського інтелекту за допомогою машин, охоплюючи навчання, міркування, вирішення проблем, сприйняття і розуміння природної мови.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Herath D., St-Onge D. Foundations of robotics. A multidisciplinary approach with python and ROS. Quebec : Springer–Kinova Inc., 2022. 543 p.
2. Морзе Н. В., Варченко-Троценко Л. О., Гладун М. А. Основи робототехніки : навч. посіб. Кам'янець-Подільський : ПП Буйницький О. А., 2016. 184 с.
3. Мокренко П. В., Ядловська В. В. Огляд розвитку робототехніки. Частина 1. (Робототехніка до XX століття) [Електронний ресурс]. *Automation, measuring and management*. 2020. С. 67–77. Режим доступу : https://doi.org/10.23939/amm2020.01.067 (дата звернення: 22.02.2024). – Назва з екрана.
4. Робототехніка : підручник / В. І. Костюк та ін. Київ : Вища шк., 1994. 447 с.
5. Цвіркун Л. І., Глулер Г. Робототехніка та мехатроніка : навч. посіб. / за заг. ред. Л. І. Цвіркуна; М-во освіти і науки України, Держ. заклад вищої освіти. «Нац. гірн. ун-т№. Вид. 3-тє, перероб. і доп. Дніпро : НГУ, 2017. 224 с.
6. Белзецький Р. С., Полторак О. М. Робототехніка як інструмент сучасної технічної освіти [Електронний ресурс] : матеріали XLVI Науково-технічної конференції підрозділів ВНТУ, Вінниця, 22 – 24 березня 2017 р. Електрон. текст. дані. 2017. Режим доступу : https://conferences.vntu.edu.ua/index.php/all-ininv/all-ininv-2017/paper/view/2375.
7. Барабан С. В., Арсенюк І. Р., Гринюк В. В. Керування рухом робота на базі нечіткої логіки. *«Інтернет-освіта-наука-2020»*, XII Міжнародна науково-практична конференція ІОН-2020, 26 – 29 травня, 2020 : збірник праць. Вінниця, 2020. С. 46 – 47.
8. Застосування чисельних методів для реалізації системи позиціонування мобільного робота / А. О. Семенов та ін. *Вісник ВПІ.* 2020. № 1. С. 77 – 83.
9. Robotics [Electronic resource] / Matjaž Mihelj [et al.]. Cham : Springer International Publishing, 2019. – Mode of access: https://doi.org/10.1007/978-3-319-72911-4 (date of access: 22.02.2024). – Title from screen.
10. Дмитрів В. Т., Ланець О. С. Динаміка і точність роботів : навч. посіб. Львів : Львівська політехніка, 2021. 200 с.
11. Погрібний В. О. Інформаційні та процесорні пристрої роботів і систем управління : навч. посіб. Київ : УМК ВО, 1990. 136 с.
12. Кирилович В. А., Савчук І В. Аналіз методів опису агрегатно-модульних промислових роботів для їх автоматизованого вибору. *Вісник ЖІТІ: Технічні науки*. 2000. № 13. С. 3–9.
13. Кириченко М. Ф., Крак Ю. В., Сорока Р. О. Оптимізація маніпуляційних роботів Київ : Либідь, 1990. 145 с.

14. Корендій В. М., Качур О. Ю., Гурський В. М. Дослідження кінематичних і динамічних характеристик колісних роботів з віброприводом : збірник тез доповідей III Міжнар. наук.-техн. конф. *«Перспективи розвитку машинобудування та транспорту – 2023»*, 1–3 черв. 2023 р. Вінниця : ВНТУ, 2023. С. 309–310.

15. Rojas A. Redesigning travel and tourism: Contactless experiences [Electronic resource]. Mode of access: https://medium.com/predict/redesigning-travel-and-tourism-contactless-experiences-926f6e24a204 (date of access: 02.03.2024). Title from screen.

16. The Use of Artificial Intelligence (AI) in Times of Pandemic [Electronic resource]. Mode of access: https://medium.com/predict/ai-covid19-ca5e4e23da02 (date of access: 02.03.2024). Title from screen.

17. COVID-19 cases | WHO COVID-19 dashboard [Electronic resource] // datadot. – Mode of access: https://covid19.who.int/ (date of access: 02.03.2024). – Title from screen.

18. COVID-19 [Electronic resource] // European Centre for Disease Prevention and Control. – Mode of access: https://www.ecdc.europa.eu/en/covid-19 (date of access: 02.03.2024). – Title from screen.

19. Morris S., Langari. R. Measurement and instrumentation : theory and application.*Third edition. Academic Press*. 2020. 736 p.

20. Osadchuk A. V. and ets. Noncontact infrared thermometer based on a self-oscillating lambda type system for measuring human body's temperature *CriMiCo – 2013* 23rd International Crimean Conference Microwave and Telecommunication Technology, Conference Proceedings, 2013. P. 1069–1070.

21. Rosete A., Soares B., Salvadorinho J., Reis J., Amorim M. Service Robots in the Hospitality Industry: An Exploratory Literature Review. *In Exploring Service Science*. *Springer International Publishing*. P. 174-186). 2020. https://doi.org/10.1007/978-3-030-38724-2_13

22. Javelosa J. Meet Connie–The Hilton’s Newest Hotel Concierge [Electronic resource] *Futurism*. Mode of access: https://futurism.com/meet-connie-hiltons-newest-hotel-concierge (date of access: 02.03.2024). – Title from screen.

23. How Hotels are Using Robots to Reduce Costs, Improve Operations and Enhance the Guest Experience | [Electronic resource] // Hotel Technology News. – Mode of access: https://hoteltechnologynews.com/2022/09/how-hotels-are-using-robots-to-reduce-costs-improve-operations-and-enhance-the-guest-experience/#:~:text=Aloft%20launched%20its%20robot%20butler,The%20results%20have%20been%20impressive (date of access: 02.03.2024). Title from screen.

24. At YOTEL Boston, This is the Droid You're Looking For [Electronic resource]. Mode of access: https://www.travelpulse.com/news/hotels-and-resorts/at-yotel-boston-this-is-the-droid-you-re-looking-for (date of access: 02.03.2024). – Title from screen.

25. Robot Mario, the new employee and mascot of the Marriott Hotel Ghent horecatrends.com [Electronic resource]. horecatrends.com *Van Spronsen*

& Partners horeca-advies. Mode of acess: https://www.horecatrends.com/en/robot-mario-the-new-employee-and-mascot-of-the-marriott-hotel-ghent/ (date of access: 02.03.2024). – Title from screen.

26. Japan's Henn na Hotel fires half its robot workforce [Electronic resource]. *Hotel Management*. Mode of access: https://www.hotelmanagement.net/tech/japan-s-henn-na-hotel-fires-half-its-robot-workforce (date of access: 02.03.2024). – Title from screen.

27. Bertolini A. Aiello G. Robot companions: A legal and ethical analysis [Electronic resource]. *The Information Society*. 2018. Vol. 34, no. 3. P. 130–140. Mode of access: https://doi.org/10.1080/01972243.2018.1444249 (date of access: 20.03.2024). Title from screen.

28. Farrant J., Ford C. Autonomous weapons and weapon reviews. *The UK Second International Weapon Review Forum International Law Studies*. 2017. Vol 93. P. 390–422.

29. Scharre P. Centaur warfighting: The false choice of humans vs. automation autonomous legal reasoning: Legal and ethical issues in the technologies in conflict. *Temple International & Comparative Law Journal*. 2016. № 30(1). P. 151–166.

30. Spiegeleire S., Matthijs M., Sweijs T. Artificial intelligence and the future of defense: Strategic implications for small- and medium-sized force providers [Electronic resource]. *The Hague Centre for Strategic Studies (HCSS*). 2017. Mode of access: https://www.researchgate.net/ publication/316983844_Artificial_Intelligence_and_the_Future_of_Defense (date of access: 20.03.2024). – Title from screen.

31. Prospects for the global governance of autonomous weapons: comparing Chinese, Russian, and US practices [Electronic resource] / Ingvild Bode [et al.]. *Ethics and Information Technology*. 2023. Vol. 25, no. 1. Mode of access: https://doi.org/10.1007/s10676-023-09678-x (date of access: 21.03.2024). – Title from screen.

32. Fox A. The deadly, incredible and absurd robots of the US Military. 2017. [Electronic resource]. Mode of access: https://www.cnet.com/ pictures/deadly-incredible-absurd-robots-the-us-military/

33. Gunkel D. The other question: can and should robots have rights? [Electronic resource]. *Ethics and Information Technology*. 2017. Vol. 20, № 2. P. 87–99. Mode of access: https://doi.org/10.1007/s10676-017-9442-4 (date of access: 21.03.2024). – Title from screen.

34. Winfield A. Robotics: A very short introduction. 2012. 143 p.

35. Sparrow R. How Robots Have Politics. [Electronic resource] *Oxford Handbook of Digital Ethics*, *Oxford Academic*. 2021. Mode of access: https://doi.org/10.1093/oxfordhb/9780198857815.013.16 (date of access: 21.03.2024). – Title from screen.

36. How safe are service robots in urban environments? Bullying a robot [Electronic resource] / P. Salvini [et al.]. *2010 RO-MAN: The 19th IEEE International Symposium on Robot and Human Interactive Communication,* Viareg-

gio, 13–15 September 2010. Mode of access: https://doi.org/10.1109/ roman.2010.5654677 (date of access: 21.03.2024). – Title from screen.

37. Malinowska J. K. Can I Feel Your Pain? The Biological and Socio-Cognitive Factors Shaping People's Empathy with Social Robots [Electronic resource]. *International Journal of Social Robotics*. 2021. Mode of access: https://doi.org/10.1007/s12369-021-00787-5 (date of access: 21.03.2024). – Title from screen.

38. Pardun J. Good Samaritan Laws [Electronic resource]. A global perspective comment. *Loyola of Los Angeles International and Comparative Law Journal*/ 1998. Vol. 20. № 3. P. 591–614. Mode of access: https://digitalcommons.lmu.edu/ilr/vol20/iss3/8 (date of access: 21.03.2024). – Title from screen.

39. Mamak K. Whether to Save a Robot or a Human: On the Ethical and Legal Limits of Protections for Robots [Electronic resource]. *Frontiers in Robotics and AI*. 2021. Vol. 8. Mode of access: https://doi.org/10.3389/frobt.2021.712427 (date of access: 21.03.2024). – Title from screen.

40. Darling K. New Breed: What Our History with Animals Reveals about Our Future with Robots. Darling : Holt & Company, Henry, 2021. 352 p.

41. Download mBlock [Electronic resource]. Mode of access: https://www.mblock.cc/en/download/. – Title from screen.

42. Makeblock_Driver_Installer [Electronic resource]. Mode of access: https://raw.githubusercontent.com/Makeblock-official/Makeblock-USB-Driver/master/Makeblock_Driver_Installer.zip. – Title from screen.

43. Arduino IDE 2.3.1 [Electronic resource]. Mode of access: https://www.arduino.cc/en/software – Title from screen.

44. Example Codes mBot's main board is mCore [Electronic resource]. Mode of access: https://docs.google.com/document/d/16uXDUmgN_9jM2sp_KGJtZZfQTpQ2PzLDtjUFla_FcA/edit

45. Бібліотеки Arduino IDE. Режим доступу: http://tinker.uamper.com/docs/biblioteki-arduino-ide.html. – Назва з екрана.

46. Схема електрична-принципова плати mCore. Режим доступу: https://content.instructables.com/FQT/NQAB/J9OWNEFV/FQTNQABJ9OWNEFV.pdf

47. Таблиця кольорових кодів RGB. Режим доступу: https://www.rapidtables.org/uk/web/color/RGB_Color.html

48. Підбір кольорів. Режим доступу: https://paletter.beloweb.name/color-picker/?color=%23fbff00

49. Двигун постійного струму [Електронний ресурс] *Вікіпедія – вільна енциклопедія.* Режим доступу: https://uk.wikipedia.org/wiki/Двигун_постійного_струму. Назва з екрана.

50. Toshiba Bi-CD Integrated Circuit Silicon Monolithic. TB6612FNG. – Mode of access: https://www.sparkfun.com/datasheets/Robotics/TB6612FNG.pdf – Title from screen.

51. Що таке ШІМ (широтно-імпульсна модуляція), простою і зрозумілою мовою для початківця. Режим доступу: https://bitkit.com.ua/chto-takoe-shim-shirotno-impulsnaya-modulyaciya

52. Sawicz D. Hobby servo fundamentals [Electronic resource]. Mode of access: https://www.princeton.edu/~mae412/TEXT/ NTRAK2002/292-302.pdf. Title from screen.

53. Керування серводвигуном за допомогою Arduino UNO [Електронний ресурс] // Fmuser International Group INC. Режим доступу: https://uk.fmuser.net/content/?18260.html. Назва з екрана.

54. Григор'єв А. С. Стенд для дослідження електричних двигунів [Електронний ресурс] : магістерська дисертація. Київ, 2020. 86 с. Режим доступу: https://ela.kpi.ua/server/api/core/bitstreams/ 30ccdc4b-4baf-465a-ac6c-468ed6f38315/content. Назва з екрана.

55. MakeBlock. Construkt your dreams. Mode of access: https://download.makeblock.com/mbot/mBot_V1.1_Blue_STD_Instruction_EN_D1.2.1_7.40.4100_Print.pdf. – Title from screen.

56. Photo modules for PCM remote control system. TSOP48. Mode of access: https://html.alldatasheet.com/html-pdf/26656/VISHAY/TSOP4838/182/1/TSOP4838.html. – Title from screen.

57. Photo modules for PCM remote control system. TSOP17. Mode of access: https://docs.rs-online.com/9680/0900766b8001d427.pdf. – Title from screen.

58. IR transmitter and receiver circuits. Mode of access: https://www.electronicshub.org/ir-transmitter-receiver-circuits/ – Title from screen.

59. Makeblock Me LED Matrix 8x16 manual. Mode of access: https://www.manua.ls/makeblock/me-led-matrix-8x16/manual?p=6 – Title from screen.

60. Me Ultrasonic Sensor. Mode of access: https://education.makeblock.com/help/me-ultrasonic-sensor/ – Title from screen.

61. Using the HC-SR04 Ultrasonic Distance Sensor with Arduino – Mode of access: https://dronebotworkshop.com/hc-sr04-ultrasonic-distance-sensor-arduino/ – Title from screen.

62. Бевз О. М., Довгалець С. М., Маслій Р. В. Програмування : навч. посіб. Ч. 2. Програмування мовою С. Вінниця : ВНТУ, 2017. 149 с.

63. Ришковець Ю. В., Висоцька В. А. Алгоритмізація та програмування : навч. посіб. Ч. 1. Львів : Новий Світ-2000, НУ «Львівська політехніка». 2018. 337 с.

64. Gookin D. C Programming for Dummies, Edition 2nd. Gookin. : Wiley & Sons, Incorporated, John, 2020. 464 p.

65. Long S. Introduction to C & GUI Programming. Edition Long. : Raspberry Pi Press, 2019. 156 p.

66. Веклич Р. А., Карнаух Т. О., Ставровський А. Б. Вступ до програмування мовою С++. Структури даних : навч. посіб. Київ : ВПЦ «Київський університет», 2018. 99 с.

67. Козак Л. І., Костюк І. В., Стасевич С. Л. Основи програмування : навч. пос. Львів : ЛДІНТУ ім. В. Чорновола. Новий Світ-2000, 2012. 328 с.

68. Ward H. H. C Programming for the PIC Microcontroller [Electronic resource]. Berkeley, CA : Apress, 2020. Mode of access: https://doi.org/10.1007/978-1-4842-5525-4 (date of access: 23.02.2024). Title from screen.

69. Malhotra N., Malhotra D. C++ Programming Fundamentals. : Mercury Learning & Information, 2022. 270 p.

Навчальне електронне видання
комбінованого використання.
Можна використовувати в локальному та мережному режимах

Сергій Володимироаич Барабан
Руслан Станіславович Белзецький
Ігор Ростиславович Арсенюк

Комп'ютерна інженерія та основи робототехніки

Навчальний посібник

Рукопис оформлено *С. Барабаном*

Редактор *В. Дружиніна*

Оригінал-макет виготовлено *Т. Старічек*

Підписано до видання 12.06.2024 р.
Гарнітура Times New Roman.
Зам. № Р2024-117.

Видавець та виготовлювач
Вінницький національний технічний університет,
Редакційно-видавничий відділ.
ВНТУ, ГНК, к. 114.
Хмельницьке шосе, 95, м. Вінниця, 21021.
Тел. (0432) 65-18-06.
press.vntu.edu.ua;
E-mail: irvc.ed.vntu@gmail.com.
Свідоцтво суб'єкта видавничої справи
серія ДК № 3516 від 01.07.2009.