МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Київський національний університет будівництва і архітектури

АЛГОРИТМІЗАЦІЯ ТА ПРОГРАМУВАННЯ

Методичні вказівки
до виконання лабораторних робіт 1–15
для здобувачів першого (бакалаврського) рівня вищої освіти
спеціальностей F7 «Комп’ютерна інженерія»
та F5 «Кібербезпека та захист інформації»

Київ 2026

УДК 004.43
А45

Укладачі: О. А. Поплавський, д-р техн. наук, доцент;
І. В. Босенко, д-к. філ., старш. викладач
І.А. Пороховніченко, асистент

Рецензент О.О. Терентьєв, д-р техн. наук, професор

Відповідальна за випуск Т.А. Гончаренко, д-р техн. наук, професор, зав. каф. інформаційних технологій

Затверджено на засіданні кафедри інформаційних технологій, протокол № 6 від 16 лютого 2026 року.

В авторській редакції.

Алгоритмізація та програмування [електронний ресурс]: методичні вказівки до виконання лабораторних робіт 1–15 / уклад. : О. А. Поплавський, І. В. Босенко. – Київ: КНУБА, 2026. – 71 с.

Містять зміст, порядок оформлення і вказівки до виконання лабораторних робіт.

Призначено для здобувачів першого (бакалаврського) рівня вищої освіти спеціальностей F7 «Комп’ютерна інженерія» та F5 «Кібербезпека та захист інформації».

Зміст

Загальні положення

Виконання лабораторних робіт є важливою складовою вивчення освітньої компоненти та спрямоване на закріплення теоретичних знань, отриманих під час лекційних занять і самостійної роботи. У процесі виконання лабораторних робіт здобувачі мають оволодіти базовими підходами до побудови алгоритмів, засобами структурного програмування, принципами модульної організації програм, а також набути практичного досвіду розв'язання прикладних задач засобами мов C та C++ у середовищі розробки, визначеному викладачем.

Метою методичних вказівок є забезпечення послідовної підготовки здобувачів до виконання лабораторних робіт, формування практичних навичок алгоритмізації, розроблення, налагодження, тестування та аналізу програм з урахуванням вимог коректності, надійності, ефективності та базових засад безпечного програмування.

Тематика лабораторних робіт охоплює основні розділи курсу: розроблення та відлагодження програм у середовищі програмування, реалізацію лінійних, розгалужених і циклічних алгоритмів, роботу з одновимірними та багатовимірними масивами, використання функцій, структурованих типів даних, покажчиків, рядків, файлів, динамічних масивів, а також створення багатофайлових проєктів. Окремі завдання можуть передбачати застосування засобів візуалізації, елементів графіки, аналізу ефективності алгоритмів, а також базових підходів до підвищення надійності та безпечності програмної реалізації.

Під час виконання лабораторних робіт здобувач повинен продемонструвати вміння аналізувати умову задачі, будувати алгоритм її розв'язання, обґрунтовувати вибір методів і засобів реалізації, розробляти коректний програмний код, виконувати компіляцію, налагодження та тестування програми, аналізувати отримані результати, а також оцінювати правильність, ефективність і надійність створених програмних рішень. Особлива увага приділяється якості програмної реалізації, дотриманню вимог до оформлення коду, логічності побудови алгоритму, коректності роботи програми на тестових даних та врахуванню базових вимог захисту інформації під час програмної реалізації.

Варіанти індивідуальних завдань до лабораторних робіт визначаються викладачем. Перед початком виконання кожної лабораторної роботи здобувач повинен ознайомитися з її метою, теоретичними відомостями, постановкою завдання, вимогами до результатів виконання та порядком

оформлення звіту. У разі потреби здобувач має виконати попередню підготовку: опрацювати рекомендований теоретичний матеріал, повторити відповідні конструкції мов програмування та підготувати початкові варіанти алгоритмів.

Звіт до лабораторної роботи повинен містити тему, мету роботи, завдання, короткі теоретичні відомості, опис алгоритму розв’язання, текст програми з поясненнями, результати тестування та висновки. За потреби до звіту можуть включатися блок-схеми, таблиці вхідних і вихідних даних, ілюстрації результатів роботи програми та інші матеріали, що підтверджують правильність виконання завдання. Оформлення звіту має відповідати встановленим вимогам.

Методичні вказівки орієнтовані на формування у здобувачів алгоритмічного мислення, навичок самостійної роботи, культури програмування та здатності застосовувати здобуті знання для розв’язання навчальних і прикладних задач у галузях комп’ютерної інженерії, комп’ютерних систем, мереж, кібербезпеки та захисту інформації.

Лабораторна робота №1. Робота з IDE Microsoft Visual Studio

Тема. Програмування алгоритмів лінійної структури, компіляція та відлагодження програм.

Мета роботи: Ознайомлення з основними прийомами роботи в інтегрованому середовищі розробки Microsoft Visual Studio. Формування практичних навичок створення найпростішої програми мовами C та C++, виконання компіляції, налагодження, тестування та первинного аналізу правильності роботи програмного коду.

Короткі теоретичні відомості

Microsoft Visual Studio є сучасним інтегрованим середовищем розробки програмного забезпечення, яке надає засоби створення, редагування, компіляції, налагодження та тестування програм.

Інсталяція IDE Microsoft Visual Studio

Для встановлення Microsoft Visual Studio необхідно мати достатній обсяг вільного місця на диску та доступ до мережі Інтернет. Для завантаження інсталяційного файла потрібно перейти на офіційний сайт Microsoft Visual Studio та обрати версію середовища, рекомендовану викладачем. Доцільно використовувати редакцію Visual Studio Community як безкоштовне інтегроване середовище розробки, придатне для навчальних цілей. Під час інсталяції слід обрати компоненти, необхідні для розроблення програм мовами C та C++ (рис. 1), після чого завершити встановлення програмного забезпечення та, за потреби, перезавантажити комп'ютер.

У середовищі Visual Studio використовується типова структура робочого простору, яка включає головне меню, панелі інструментів, вікна проєкту, вікна властивостей, область редагування програмного коду та засоби налагодження. Набір доступних інструментів залежить від типу створеного проєкту та режиму роботи користувача. Використання цього середовища у межах лабораторної роботи дає змогу здобувачам опанувати базові операції створення проєкту, введення та збереження програмного коду, запуску компіляції, аналізу повідомлень компілятора, виявлення помилок і покрокового налагодження програм.

У сімействі продуктів *Visual Studio* використовується загальне інтегроване середовище розробки, що складається з декількох елементів: панелі інструментів, різних закріплених вікон або вікон що автоматично приховуються в лівій, нижній або правій областях, а також області

редакторів. Набір доступних вікон інструментів, меню і панелей інструментів залежить від типу проєкту або файлу, в якому виконується розробка.

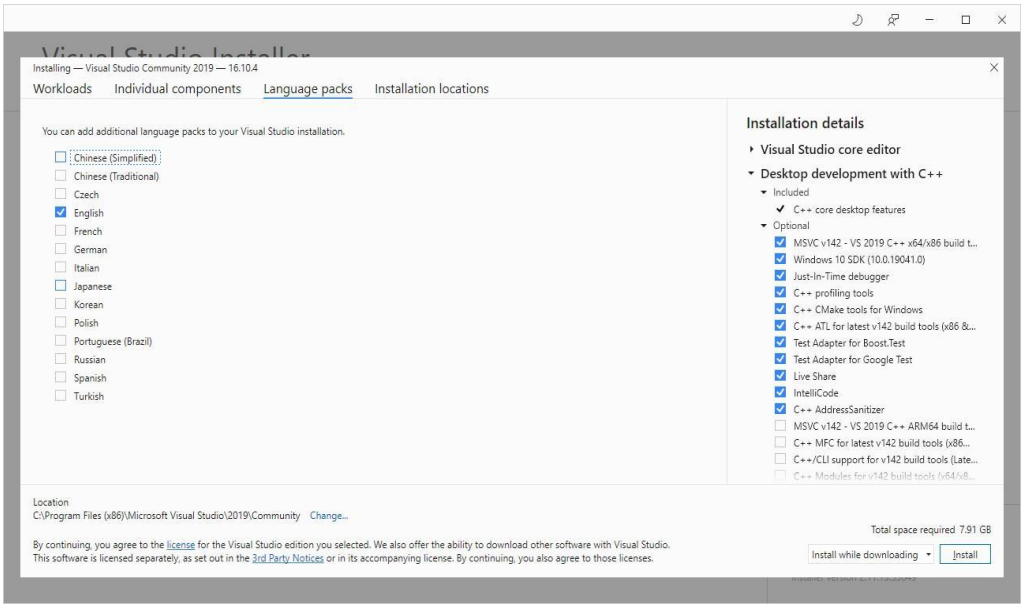

Рис. 1. Налаштування при інсталяції

Рішення та проєкти як контейнери

У *Visual Studio* реалізовані контейнери: рішення та проєкти, які роблять можливим використання в інтегрованому середовищі розробки (*IDE*) всього діапазону засобів, конструкторів, шаблонів і параметрів. Також, *Visual Studio* надає папки рішень для того, щоб структурувати пов'язані проєкти по групах і потім виконувати дії над цими групами проєктів.

Проєкт – це група файлів і налаштувань, з яких збирається остаточна програма або вихідні файли. Проєкт включає набір файлів вихідних текстів та метаданих, наприклад посилання на компоненти та інструкції побудови. Як правило, при побудові проєктів створюється один або кілька файлів із кодом програми. Рішення включає один або декілька проєктів, а також файли і метадані, необхідні для опису рішення в цілому (рис. 2).

Щоб допомогти користувачам організовувати і виконувати стандартні завдання із застосуванням елементів, що розробляються, проєкти *Visual Studio* використовуються як контейнери в межах рішення. Це дозволяє логічно управляти, виконувати побудову і налагоджувати елементи, що утворюють програму. На виході, як правило, ми отримуємо програму що виконується (*EXE*), файл бібліотеки динамічного компонування (*DLL*) або модуль.

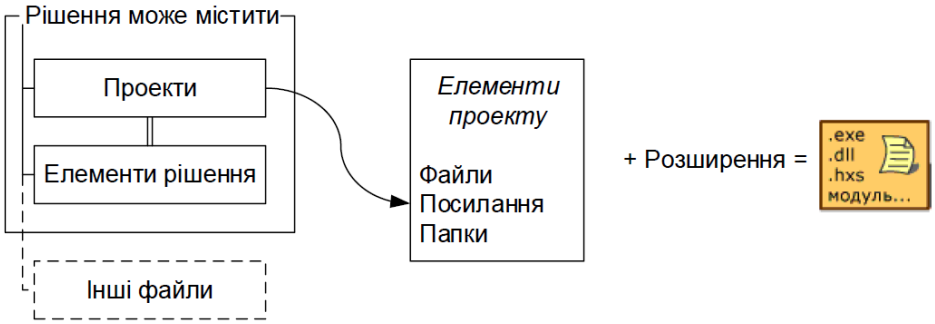

Рис. 2. Схема вмісту рішення

Проєкт може бути простим або складним залежно від конкретних вимог. Простий проєкт може містити файли коду програми та файл проєкту. Більш складні проєкти можуть включати ці ж елементи і, крім того, сценарії баз даних, збережені процедури та посилання на існуючі *XML* (веб-служби).

Шаблони проєктів

Visual Studio надає шаблони для створення проєктів найбільш поширених типів. Використання шаблонів проєктів дає змогу користувачеві зосередитися на розробленні та реалізації окремих функцій програми, тоді як базова структура проєкту обраного типу формується автоматично.

Файли проєктів

Кожен шаблон створює файл проєкту, у якому містяться метадані поточного проєкту. Файл проєкту зберігає його основні параметри, налаштування, а також, за потреби, список файлів проєкту та відомості про їх розташування.

Під час додавання файла до проєкту інформація про його фізичне розташування вноситься до файла проєкту. У разі видалення такого зв’язку ці відомості вилучаються з файла проєкту. Шаблон проєкту визначає, які команди доступні для кожного його елемента.

Майстер створення програм надає інтерфейс для створення проєкту за шаблоном, а також для формування початкових файлів програмного коду. Він налаштовує структуру проєкту, основні меню та панелі інструментів, а також забезпечує автоматичне підключення деяких службових і заголовкових файлів.

Головна сторінка, відкриття та створення проєктів

Після запуску *Visual Studio* відкривається стартова сторінка, яка дозволяє отримати легкий доступ до наявних проєктів або створити новий проєкт.

В області, яка розташована ліворуч, можна обрати та відкрити

проєкти, з якими працювали нещодавно (це область нижче слів *Open recently*). Праворуч, нижче слів *Get started*, розташовані кнопки, які дозволяють відкрити чи створити проєкти та рішення.

Для відкриття вже існуючого проєкту потрібно або натиснути кнопку *Open a project or solution* стартової сторінки, або обрати опції головного меню *File: Open: Project/Solution*…(рис. 3).

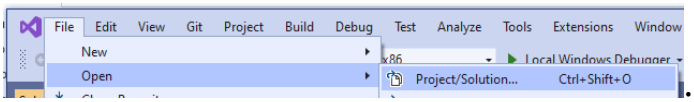

Рис. 3. Вибір існуючого проєкту

Для створення нового проєкту використовується майстер додатків. Для його відкриття потрібно або натиснути кнопку *Create a new project* стартової сторінки, або обрати опції головного меню *File: New: Project*…(рис. 4).

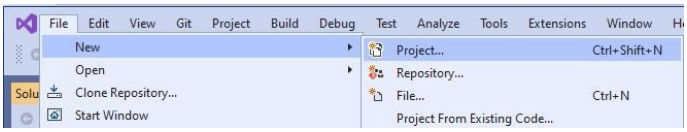

Рис. 4. Створення нового проєкту

В діалоговому вікні можна задати ім'я, місце зберігання і шаблон нового проєкту.

Щоб створити порожній проєкт, використовуйте шаблон *Empty Project.*

У вікні налаштування параметрів проєкту (*Configure your new project*) потрібно вказати ім’я проєкту (поле *Project name*) та місце його зберігання (поле *Location*). Найчастіше ми будемо створювати рішення, які складаються з одного проєкта, тому для спрощення файлової структури можна вибрати опцію “Place solution and project in the same directory” (рис. 5).

Після цього відкриється робоче вікно *Visual Studio.* В області ліворуч розташований Провідник рішень (Solution Explorer), який відображає поточне рішення, яке може містити один чи більше проєктів. На рис. 6 наведено приклад рішення «*Laba5*» (наведено зеленим). Це рішення містить один проєкт із ім’ям «*Laba5*» (наведено синім). В проєкт можуть входити різні типи файлі (наведено жовтим) – файли з кодом програми (*Source Files*) із розширенням **.c*, файли із заголовками (*Header Files*) із розширенням **.h* , та інші. В прикладі на рисунку нижче проєкт ще не містить жодного файлу.

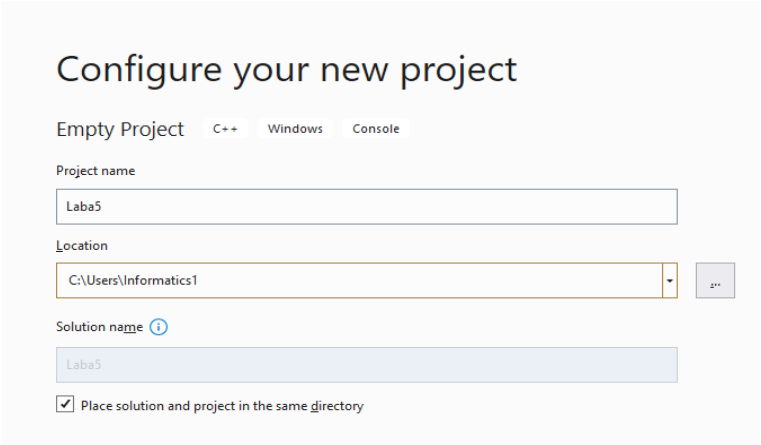

Рис. 5. Налаштування параметрів проєкту

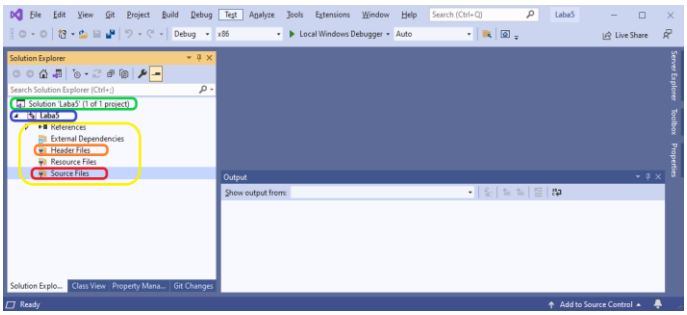

Рис. 6. Приклад рішення «*Laba5*»

Додавання файлів до проєкту

Після створення проєкту до нього можна додавати нові чи вже існуючі файли. Для цього потрібно обрати підкаталог проєкту відповідного типу та, натиснувши правою кнопкою миші, викликати контекстне меню. В цьому контекстному меню потрібно обрати опції *Add: New Item…* для додавання нового файлу чи *Add: Existing Item…* для додавання існуючого файлу. На рис. 7 показано додавання до проєкту «*Laba5*» нового файлу, в якому буде написано код програми:

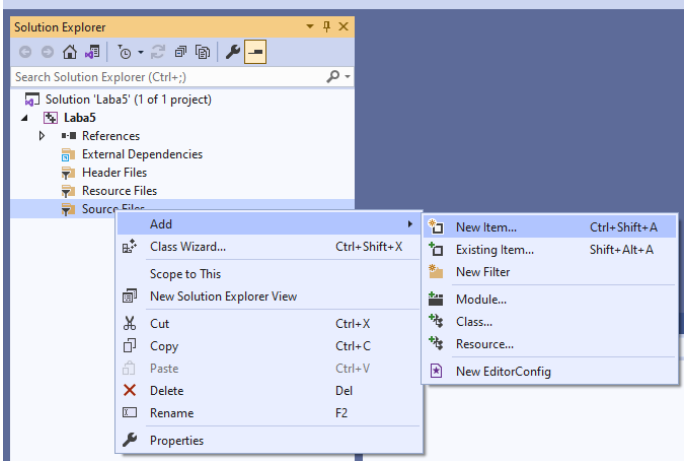

Рис. 7. Додавання нового файлу

Якщо додавати вже існуючий файл, то у провіднику, який відкрився, потрібно просто вказати шлях до потрібного файлу.

Якщо файл потрібно створити (опції *Add: New Item…*), відкривається вікно *Add New Item*, в якому потрібно вказати тип файлу, його ім'я та місце розташування. За замовчуванням файл буде збережений в поточному каталозі проєкту. На рисунку нижче показано додавання файлу, в якому буде написана програма мовою *C++* із ім'ям *Laba5_code.c*. Цей файл буде частиною проєкту *Laba5*.

Після натискання кнопки *Add*, в області ліворуч, над вікном *Output*, у текстовому редакторі відкривається створений файл. Після цього можна писати текст програми (рис. 8).

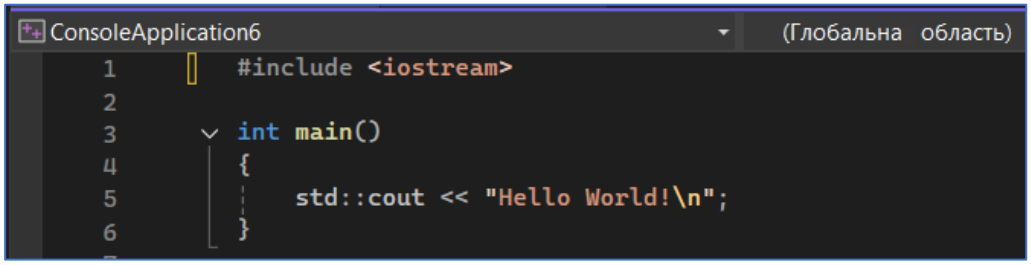

Рис. 8. Вікно для написання коду

Засоби побудови

У середовищі *Visual Studio* передбачений потужний набір засобів побудови та налагодження. За допомогою конфігурацій побудови можна вибирати компоненти для побудови, виключати компоненти, які не потрібно включати в побудову, а також визначати, як будуть побудовані вибрані проєкти і для якої платформи.

Для побудови (збірки) програми потрібно вибрати опції меню *Build*:*Build Solution* (рис. 9).

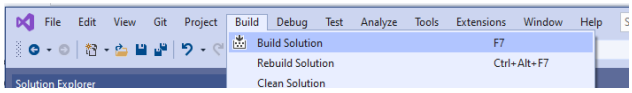

Рис. 9. Збірка програми

У вікні *Output* повинна відображатися інформація про хід збирання, а також про виявлені помилки компіляції. Приклад інформації про результати збирання проєкту можна побачити на рис. 10 (наведено синім).

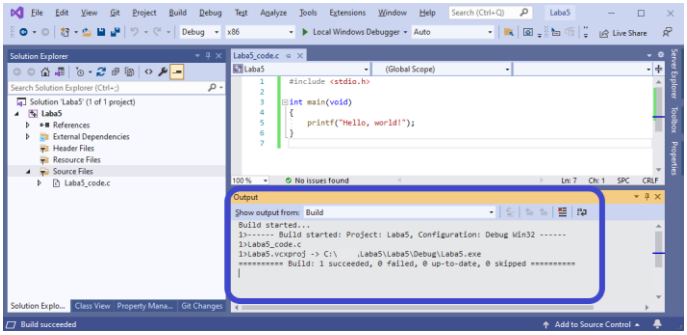

Рис. 10. Вікно *Output*

Якщо при побудові файлу, що виконується були виявлені помилки, то, зазвичай, файл що виконується побудовано не буде і про це буде повідомлено у вікні *Output* (наведено фіолетовим). Список помилок (наведено червоним) та попереджень (наведено жовтим) із детальним описом (наведено помаранчевим) також можна побачити в вікні *Output* (рис. 11).

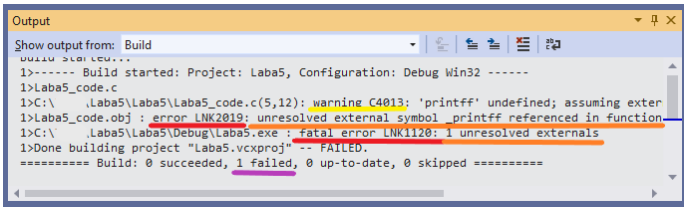

Рис. 11. Помилки при побудові програми

Якщо вікно *Output* було випадково закрито, то знову відкрити його можна обравши опції меню *Debug: Windows: Output* (рис. 12).

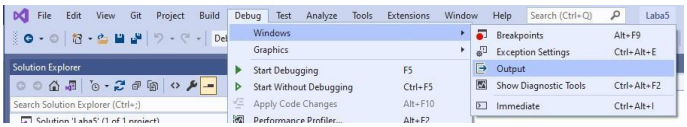

Рис. 12. Відкриття вікна *Output*

Засоби налагодження

Якщо побудова виконуваного файлу пройшла успішно, тобто усі синтаксичні помилки було виправлено, то можна починати процес налагодження.

Процес налагодження, який виконується за допомогою відладчика, дозволяє виявити та усунути таких проблем, як логічні та семантичні помилки, які проявляються вже під час виконання. У режимі зупинки (*Break Point*) можна переглядати локальні змінні та інші дані, використовуючи такі засоби, як Вікна змінних (*Watch*) і Вікно пам'яті (*Memory*).

Для того, щоб зупинити виконання програми до певного рядка, в

цьому рядку потрібно встановити *Break Point.* Його можна встановити клікнувши лівою кнопкою миші на сірій області, яка розташована ліворуч від тексту програми, навпроти обраного рядка (на рис. 13 *Break Point* – це червоний кружечок).

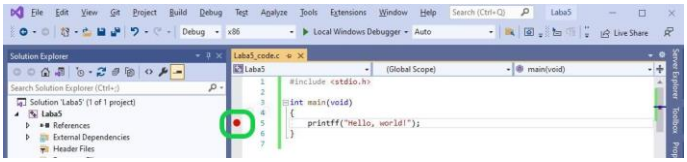

Рис. 13. *Break Point*

Для запуску програми в режимі налагодження потрібно або обрати опції меню

Debug: Start Debugging, або натиснувши гарячу клавішу **F5** (рис. 14).

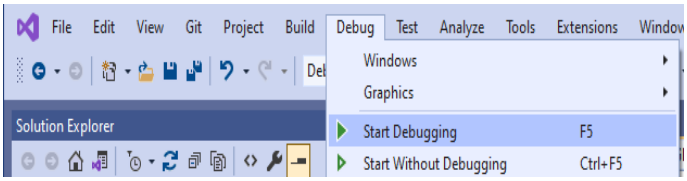

Рис. 14. *Start Debugging*

Якщо заздалегідь були поставлені *Break Point*, то після запуску програми на виконання в режимі налагодження, програма буде виконана тільки до першого *Break Point*, після чого виконання програми буде призупинено. Про місце зупинки програми інформує жовта стрілка на сірому полі ліворуч від тексту програми (на рисунку нижче наведено помаранчевим). Інформацію про значення, які зберігаються у змінних, можна отримати або за допомогою вікна *Autos* (на рис. 15 наведено салатовим), або вікна *Watch* (на рисунку нижче наведено зеленим).

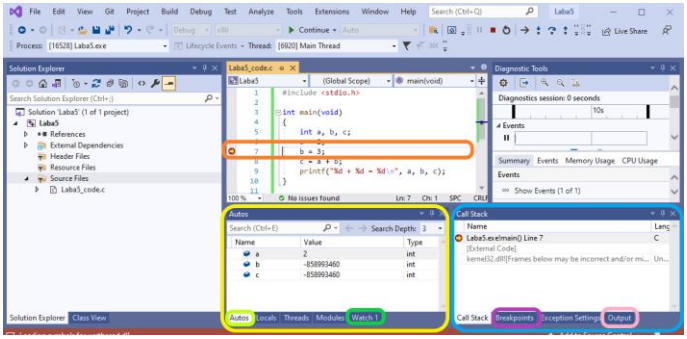

Рис. 15. Вікна *Autos* та *Watch*

До вікна *Autos* автоматично додаються змінні, видимі у межах поточного блоку, а до вікна *Watch* змінні потрібно додавати вручну. Для цього потрібно або безпосередньо набрати назву змінної в полі *Name* вікна

Watch (там де написано підказка *Add item to watch*), або клікнути правою кнопкою миші і в контекстному меню обрати опцію *Add Watch* (рис. 16).

Рис. 16. Функція *Add Watch*

Продовжити виконання програми до наступної *Break Point* або до кінця програми можна або обравши опції меню *Debug: Continue*, або скориставшись гарячою клавішею *F5* (рис. 17).

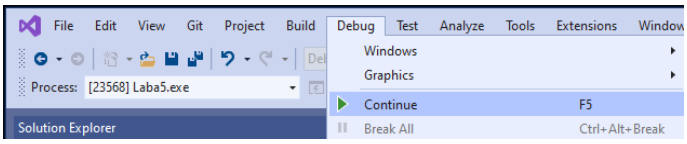

Рис. 17. Функція *Debug: Continue*

Середовище *IDE Visual Studio* також пропонує покрокове виконання програми, коли за один крок буде виконано інструкції певного рядка. Покрокове виконання інструкцій можна виконувати із заходом всередину функцій, що викликаються або без заходу всередину функцій (рис. 18).

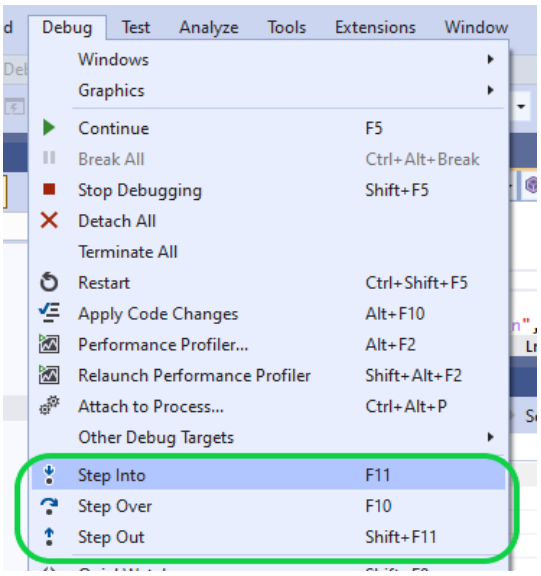

Рис. 18. Покрокове виконання функцій

Якщо обрати опції меню *Debug:Step Over* або натиснути **F10**, то наступний рядок буде виконано без заходу всередину функції.

Якщо обрати опції меню *Debug*:*Step Into* або натисніть **F11,** то наступний рядок буде виконано із заходом всередину функцій яка написана в цьому рядку.

Якщо обрати опцію меню *Debug:Step Out*, то буде виконано вихід із поточної функції в функцію, що її викликала.

Значення змінних у вікні перегляду *Watch* оновлюються по мірі виконання програми.

На рис. 19 наведено приклад вмісту змінної *c* до (рисунок ліворуч) та після (рисунок праворуч) виконання її модифікації (рядок 8).

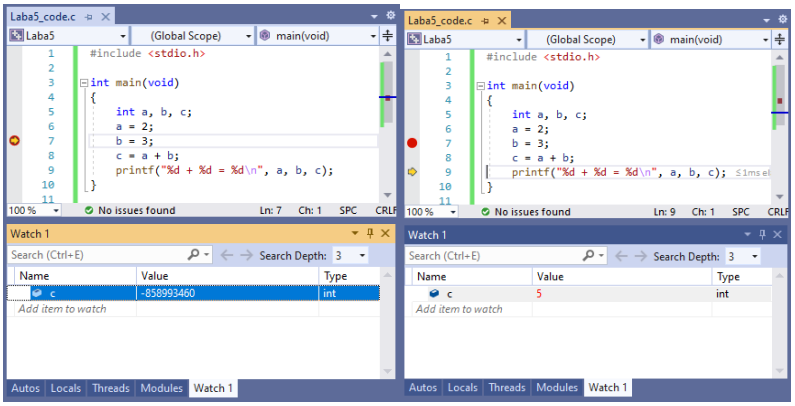

Рис. 19. Вміст змінної *c*

Дострокове завершення роботи програми в налагоджувальному режимі.

Для дострокового завершення роботи в програми, запущеної в налагоджувальному режимі в потрібно обрати опції меню *Debug*:*Stop Debugging* (або натисніть **Shift** + **F5**).

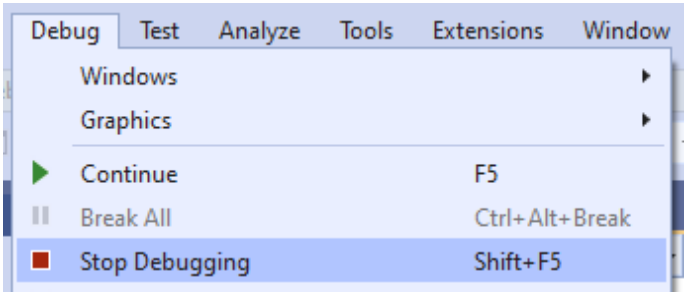

Рис. 20. Функція *Debug*:*Stop Debugging*

Деякі стандартні математичні функції

Стандартні математичні функції оголошені в заголовочному файлі ***math.h***.

abs – абсолютне значення цілого числа - |x|: *int abs(int x);*

fabs – абсолютне значення дробового числа - |x|: *double fabs(double x);*

sqrt – обчислення квадратного кореня: *double sqrt(double x);*

pow – піднесення до степеня: *double pow(double x, double y);*

cos – косинус – *cos(x)* (тут і далі *x* задається в радіанах): *double cos(double x);*
sin – синус – *sin(x)*: *double sin(double x);*
tan – тангенс – *tg(x)*: *double tan(double x);*
acos – арккосинус – *arccos(x)*: *double acos(double x);*
asin – арксинус – *arcsin(x)*: *double asin(double x);*
atan – арктангенс – *arctg(x)*: *double atan(double x);*
atan2 – арктангенс – *arctg(y/x)*: *double atan2(double y, double x);*
exp – експонента: *double exp(double x);*
log – натуральний логарифм – *ln(x)*: *double log(double x);*
log10 – десятковий логарифм - *log10(x)*: *double log10(double x);*

Для роботи з даними типу *float* більшість перерахованих функцій має еквіваленти, такі як *logf*, *powf* і т.д.

Деякі математичні константи

Стандартні математичні функції знаходяться в заголовному файлі ***math.h***. Для використання констант **перед** підключенням файлу заголовків ***math.h*** слід додати наступний рядок

#define _USE_MATH_DEFINES

Перелік констант, які визначені в файлі ***math.h*** , наведено в табл. 1.

Таблиця 1

Перелік констант *math.h*

Символ	Вираз	Значення
M_E	e (експонента)	2.71828182845904523536
M_LOG2E	log2(e)	1.44269504088896340736
M_LOG10E	log10(e)	0.434294481903251827651
M_LN2	ln(2)	0.693147180559945309417
M_LN10	ln(10)	2.30258509299404568402
M_PI	Pi	3.14159265358979323846
M_PI_2	pi/2	1.57079632679489661923
M_PI_4	pi/4	0.785398163397448309616
M_1_PI	1/pi	0.318309886183790671538
M_2_PI	2/pi	0.636619772367581343076
M_2_SQRTPI	2/sqrt(pi)	1.12837916709551257390
M_SQRT2	sqrt(2)	1.41421356237309504880
M_SQRT1_2	1/sqrt(2)	0.707106781186547524401

Робоче завдання

1. Скласти алгоритм розв’язання задачі згідно з варіантом. Побудувати блок-схему алгоритму.
2. Створити новий порожній проєкт.
3. Написати програму, що реалізує складений алгоритм. Вхідні дані, задані у варіанті, подати у вигляді констант, а результати обчислень вивести на екран (використовувати функцію printf()).
4. Скомпілювати проєкт.
5. Провести покрокове налагодження:
   a. Встановити *Break Point* на певному рядку всередині програми.
   b. Додати до віконця *Watch* частину змінних, які використовуються в програмі.
   c. Запустити на виконання програму у режимі налагодження.
   d. Подивитися значення змінних за допомогою віконця *Watch*.
   e. Переконатися за допомогою віконця *Watch* що деякі змінні змінили своє значення.
   f. Виконати програми до кінця та продемонструвати результат обчислення задачі згідно варіанту (табл. 2).

Варіанти завдань

1. Обчислити периметр і площу прямокутника за довжинами його сторін a та b.
2. Обчислити об’єм циліндра і площу його бічної поверхні за радіусом основи r та висотою h.
3. Обчислити гіпотенузу і площу прямокутного трикутника за довжинами його катетів a та b.
4. Обчислити площу ромба за довжинами його діагоналей d_1 та d_2.
5. Обчислити довжину діагоналі прямокутника за довжинами його сторін a та b.
6. Знайти середню швидкість руху, якщо першу ділянку шляху довжиною s_1 пройдено зі швидкістю v_1, а другу ділянку довжиною s_2 – зі швидкістю v_2.
7. Обчислити площу повної поверхні куба та його об’єм за довжиною ребра a.
8. Обчислити периметр і площу рівнобедреного трикутника за основою a та висотою h, проведеною до основи.
9. Обчислити площу круга і довжину кола за заданим діаметром d.

10. Обчислити об’єм прямокутного паралелепіпеда за його вимірами *a, b* і *c*.
11. За координатами двох точок (x_1,y_1) і (x_2,y_2) обчислити відстань між ними.
12. Обчислити площу трапеції за основами *a, b* і висотою *h*.
13. Обчислити радіус кола, якщо відома його площа *S*.
14. Обчислити площу бічної поверхні конуса за радіусом основи *r* та твірною *l*.
15. Обчислити об’єм кулі за заданим діаметром *d*.
16. Обчислити площу правильного шестикутника за довжиною його сторони *a*.
17. Обчислити довжину медіани, проведеної до сторони *c*, у трикутнику зі сторонами *a*, *b* і *c*.
18. Обчислити периметр правильного шестикутника і площу круга, описаного навколо нього, якщо сторона шестикутника дорівнює *a*.
19. Обчислити площу кільця, якщо задано діаметри зовнішнього *D* і внутрішнього *d* кіл, де *D>d*.
20. Обчислити об’єм конуса за радіусом основи *r* та висотою *h*.
21. Обчислити площу трикутника за довжиною сторони *a* і висотою *h*, проведеною до цієї сторони.

Контрольні запитання

1. Які основні компоненти зазвичай включає в себе сучасне середовище розробки?
2. Які засоби відлагодження в середовищі розробки *Visual Studio* вам відомі?
3. Які додаткові можливості дає запуск програми в режимі налагодження?
4. Які стандартні математичні функції мови C++ вам відомі?

Лабораторна робота №2. Програма, що розгалужується

Тема. Програмування алгоритмів, що розгалужуються.

Мета: Набути практичних навичок розроблення програм із розгалуженням з використанням операторів *if* та *switch*.

Теми для попереднього опрацювання:

- локальні та глобальні змінні;
- оператори: арифметичні, порівняння, логічні, побітові;
- порядок виконання операторів;
- складені оператори присвоювання;
- умовний оператор *if*;
- оператор вибору *switch*.

Загальні відомості

Алгоритм називається розгалуженим, якщо він містить кілька гілок, що відрізняються одна від одної виконуваними діями. Перехід обчислювального процесу на ту чи іншу гілку алгоритму визначається результатом обчислення виразу-умови, який може містити арифметичні та логічні операції, а також операції відношення.

Операції відношення:

`>` – більше;

`>=` – більше або рівно;

`<` – менше;

`<=` – менше або рівно;

`==` – рівно;

`!=` – не рівно.

Логічні операції:

`&&` – логічна І;

`||` – логічна АБО.

Оператори керування роботою програми називають керуючими конструкціями програми. До них відносять:

- складені оператори;
- оператори вибору;
- оператори циклів;
- оператори переходу.

Оператор виразу. Будь-який вираз, що закінчується крапкою з комою, розглядається як оператор, виконання якого полягає в обчисленні

цього виразу. Окремий випадок – порожній оператор, який позначається як «;».

Приклади:

```
i++;
a+=2;
x=a+b;
```

Складені оператори. До складених операторів відносять власно складені оператори і блоки. В обох випадках це послідовність операторів, вкладена у фігурні дужки. Блок відрізняється від складеного оператора наявністю визначень у тілі блока. Наприклад:

```
{
    n++;                // це складений оператор
    summa+=n;
}
{
    int n=0;
    n++;                //це блок
    summa+=n;
}
```

Оператори вибору – це умовний оператор і перемикач.

Умовний оператор має повну та скорочену форми.

```
if ( вираз-умова ) оператор;   //скорочена форма
```

Як вираз-умова можуть використовуватися арифметичний вираз, відношення та логічний вираз. Наприклад:

```
if (x<y && x<z) min=x;  //скорочена форма
if ( вираз-умова ) оператор1; //повна форма
else оператор2;
```

Якщо значення виразу-умови відмінні від нуля, то виконується *оператор1*, при нульовому значенні виразу-умови виконується *оператор2*.

Перемикач визначає множинний вибір.

```
switch (вираз)
{
    case константа1 : оператор1 ;
    case константа2 : оператор2 ;
    . . . . . .
    default: оператори;       //може бути відсутнім
}
```

При виконанні оператора switch обчислюється вираз, записаний після *switch*, він має бути цілим числом. Отримане значення послідовно порівнюється з константами, які записані слідом за *case*. При першому же збігу виконуються оператори, позначені даною міткою. Якщо виконані оператори не містять оператора переходу, то далі виконуються оператори всіх наступних варіантів, поки не з'явиться оператор переходу або не закінчиться перемикач. Якщо значення виразу, записаного після *switch*, не збіглося з жодною константою, то виконуються оператори, які ідуть за міткою *default*. Мітка *default* може бути відсутньою.

Оператори переходу виконують безумовну передачу керування.

break – оператор переривання оператора.

Наприклад,

```
switch (вираз)
{
    case константа1 : оператор1 ; break;
    case константа2 : оператор2 ; break;
    . . . . . .
    default: оператори;       //може бути відсутнім
}
```

У разі, коли отримане значення збіглося з однією із констант, які записані слідом за *case*, виконуються оператори, позначені даною міткою, і оператор *break*, який перериває *case*; керування передається наступному за *case* оператору.

Тернарна операція. В C/C++ є умовна операція «? :», яка називається тернарною операцією (тобто тримісна (має три операнди), єдина в C/C++).

Форма запису тернарної операції:

"умова" ? "вираз 1" : "вираз 2";

Якщо умова дійсна, то виконується вираз 1, інакше (умова неправильна) виконується вираз 2.

Наприклад:

a > b ? max= a : max = b; *// якщо a > b, то виконується max= a,*
// інакше виконується max = b

Індивідуальні завдання

1. Знайти найбільше значення серед трьох заданих чисел.

2. Знайти найменше значення серед чотирьох заданих чисел.

3. Для заданих чисел *a, b* і номера дії *k* обчислити результат: при *k=1: a+b*, при *k=2: a-b*, при *k=3: a·b*, при *k=4: a/b*. Передбачити перевірку ділення на нуль.

4. Визначити, чи є задане ціле число парним чи непарним.

5. Визначити, чи належить задане число проміжку *[a;b]*.

6. Для заданого тризначного числа визначити, чи є сума його цифр парною.

7. Для заданого тризначного числа визначити, чи всі його цифри різні.

8. Для заданого тризначного числа визначити, чи утворюють його цифри зростаючу послідовність.

9. За заданими сторонами *a, b, c* визначити, чи може існувати прямокутний трикутник.

10. За заданими сторонами *a, b, c* визначити, чи є трикутник рівнобедреним, рівностороннім або різностороннім.

11. За координатами точки *(x,y)* визначити, у якій координатній чверті вона лежить, або чи належить осі координат.

12. Для заданого року визначити, чи є він високосним.

13. За номером місяця визначити кількість днів у ньому для невисокосного року.

14. За номером місяця визначити назву пори року.

15. За заданою оцінкою у 100-бальній шкалі визначити національну оцінку: відмінно, добре, задовільно або незадовільно.

16. Для заданих дійсних чисел *x* і *y* визначити, яке з них має більший модуль, або чи вони рівні за модулем.

17. Для заданого тризначного числа визначити, яка з його цифр є найбільшою.

18. Для заданого тризначного числа визначити, чи ділиться сума його цифр на 3.

19. За номером дня місяця d і номером місяця m визначити, чи є така дата коректною. Для інших значень вивести повідомлення про помилку.

Додаткові умови виконання завдання:

- текст програми повинен мати коментарі до коду;

Контрольні запитання

1. Як працює умовний оператор *if*?
2. Який вираз називається складеним логічним? Наведіть приклади.
3. Який оператор називають оператором множинного вибору? Наведіть приклад.
4. Як працює оператор *switch*?
5. Як працює тернарний оператор? Наведіть приклад.
6. Коли умовний оператор називається вкладеним?
7. Навіщо в операторі *switch* використовується оператор *break*?
8. Чи можуть бути вкладеними оператори *switch*?
9. Чи можна замість оператора *if* використовувати тернарний оператор і навпаки, – замість тернарного – оператор *if*?

Лабораторна робота №3. Циклічні структури

Тема. Програмування алгоритмів циклічної структури. Документування коду.

Мета: Набути практичних навичок розроблення програм із використанням циклічних структур та операторів *for, while, do-while*.

Теми для попереднього опрацювання:

- оператор *sizeof*;
- оператори циклу *for, while, do-while*;
- оператори переходу *break, continue*.

Загальні відомості

Циклічний алгоритм – це алгоритм, який містить фрагменти, що багаторазово виконуються при різних значеннях проміжних даних. Сукупність дій, що повторюються в циклі, називається тілом циклу. Одноразове виконання тіла циклу називається ітерацією.

Основу таких алгоритмів становлять оператори циклу. Кількість повторень може бути задана явно, тобто цикл виконується наперед визначену кількість разів, або неявно, коли повторення триває доти, доки виконується певна умова.

Оператори циклів. Розрізняють:

1. Цикл з передумовою:

```
while (вираз-умова)
{
    оператор;
}
```

Як <вираз-умова> найчастіше використовується відношення або логічний вираз. Якщо результат виразу відмінний від 0 (*true*), тіло циклу виконується; якщо вираз-умова стане *false* – цикл закінчується.

Наприклад,

```
int a=0, sum = 0;
while (a!= 10)
{
    sum+=a++;
}
```

2. Цикл із післяумовою:

```
do
{
    оператор;
}
while (вираз-умова);
```

Тіло циклу виконується, поки вираз-умова дійсний (*true*).

Наприклад,

```
int a=0, sum = 0;
 do
{
    sum+=a++;
}
while (a <= 10);
```

3. Цикл з параметром:

```
for ( вираз_1; вираз-умова; вираз_3)
{
    оператор;
}
```

Вираз_1 та вираз_3 можуть складатися з декількох виразів, розподілених комами. Вираз_1 задає початкові умови для циклу (ініціалізація). Вираз- умова визначає умову виконання циклу: якщо вона

відмінна від 0, цикл виконується, а потім обчислюється значення виразу_3. Вираз_3 задає зміну параметра циклу або інших змінних (корекція). Цикл триває доти, поки вираз-умова не стане дорівнювати 0.

for (int a=0, sum = 0; a <= 10 ; a++) sum+=a;

Будь-який вираз може бути відсутнім, але поділяючи їх « ; » повинні залишатися завжди.

Наприклад:

```
int sum = 0; int a = 0;
for ( ; a <= 10 ; a++) s+=a;
```

Оператори переходу. Виконують безумовну передачу керування:

break – оператор, що перериває виконання операторів *while, do-while, for* та *switch*. Управління передається наступному оператору за перерваним. Оператор break перериває виконання найближчого циклу або оператора switch, після чого керування передається оператору, що йде безпосередньо після нього. Використовується, якщо умову продовження ітерацій треба перевіряти в середині циклу;

continue – оператор переходу до наступної ітерації циклу *while, do-while* або *for*. У циклах *while, do-while* наступна ітерація починається з обчислення умовного виразу, у циклі *for* – з обчислення виразу для зміни параметра циклу, а потім - умовного виразу.

Текст програми необхідно супроводжувати *doxygen* коментарями.

Основне завдання

Реалізувати програму відповідно до індивідуального завдання за допомогою трьох типів циклів: *for*, *while*, *do-while* (отримати три однакових результати).

Індивідуальні завдання

1. Обчислити суму всіх цілих чисел від 1 до n.
2. Обчислити добуток усіх парних чисел від 2 до n.
3. Для заданого цілого числа визначити суму його цифр.
4. Для заданого цілого числа визначити добуток його цифр.
5. Визначити, чи є задане ціле число паліндромом.
6. Визначити кількість непарних чисел у заданому діапазоні.
7. Обчислити суму квадратів усіх цілих чисел від 1 до n.
8. Знайти найменший дільник заданого числа, більший за 1.
9. Визначити кількість чисел у заданому діапазоні, що діляться на 5 без залишку.
10. Обчислити суму всіх чисел заданого діапазону, що є парними.

11. Для заданого цілого числа визначити кількість нулів у його записі.
12. Знайти перше число у заданому діапазоні, що ділиться на 9 без залишку.
13. Обчислити добуток усіх чисел заданого діапазону, що діляться на 4 без залишку.
14. Визначити кількість чисел у заданому діапазоні, квадрат яких не перевищує заданого числа n.
15. Для заданого натурального числа обчислити суму всіх його дільників.
16. Визначити, чи є задане натуральне число степенем двійки.
17. Знайти найбільшу цифру у записі заданого цілого числа.
18. Знайти найменшу цифру у записі заданого цілого числа.
19. Обчислити суму всіх двоцифрових чисел, що діляться на 7 без залишку.
20. Визначити кількість трицифрових чисел, у яких сума цифр дорівнює заданому числу.
21. Для заданого натурального числа n обчислити значення суми 1 + 1/2 + 1/3 + ... + 1/n.

Додаткові умови виконання завдання:

- звіт має бути виконаний згідно з вимогами до оформлення лабораторних робіт.

Контрольні запитання

1. Як записується і як працює оператор *for*?
2. У чому відмінність оператора *while* від оператора *do while*?
3. Як програмуються циклічні алгоритми з явно заданою кількістю повторень циклу?
4. Як програмуються циклічні алгоритми із заздалегідь невідомим числом повторень циклу?
5. Напишіть оператор циклу, який не виконується жодного разу.
6. Напишіть оператор циклу, який виконується необмежену кількість раз.
7. Замініть фрагмент програми з оператором *for* рівнозначним фрагментом програми з оператором *while*.
8. Замініть фрагмент програми з оператором *for* рівнозначним фрагментом програми з оператором *do while*.
9. Як можна перервати виконання оператора цикла?
10. Яке призначення операторів *break* і *continue*?

Лабораторна робота №4. Одновимірні масиви

Тема. Одновимірні масиви.

Мета: Набути практичних навичок розроблення програм з використанням статичних масивів.

Теми для попереднього опрацювання:

- одновимірні масиви;
- ініціалізація масиву;
- рядки.

Загальні відомості

Масиви є важливою структурою даних у програмуванні. Масив – це впорядкована сукупність елементів одного типу, об’єднаних спільним іменем. Масиви використовуються для зберігання та оброблення великої кількості однотипних даних. Окремий елемент масиву ідентифікується індексом.

Масиви можуть бути одновимірними, двовимірними, тривимірними та багатовимірними. Кількість вимірів визначає спосіб організації даних і доступу до них.

У мові C/C++ індексація елементів масиву починається з нуля. Для доступу до елемента масиву використовують ім’я масиву та індекс відповідного елемента.

Масиви належать до структурованих типів даних. Існують два основні способи роботи з масивами:

1. виділення пам’яті під заздалегідь відому кількість елементів;
2. виділення пам’яті під час виконання програми для кількості елементів, що визначається в процесі роботи.

У багатьох задачах кількість елементів масиву відома заздалегідь і не змінюється під час виконання програми. У такому разі розмір масиву задається безпосередньо у програмному коді. Змінити кількість елементів такого масиву під час виконання програми неможливо. Такі масиви називаються статичними.

Одновимірний масив – масив з одним параметром, що характеризує кількість елементів у ньому. Фактично одновимірний масив – це масив, у якому може бути тільки один рядок, і n кількість стовпців. Стовпці в одновимірному масиві – це елементи масиву. Приклад оголошення масиву:

int a[16];

Номери елементів будуть мати значення в інтервалі від 0 до 15 включно. Завжди відразу після імені масиву йдуть квадратні дужки, у яких задається розмір одновимірного масиву, цим масив і відрізняється від усіх інших змінних.

Масиви при оголошенні можуть бути проініціалізовані, наприклад,

int a[16] = { 5,-12,-12,9,10,0,-9,-12,-1,23,65,64,11,43,39,-15};

Ініціалізація одновимірного масиву виконується у фігурних дужках після знака присвоювання; кожний елемент масиву відділяється від попереднього комою.

int a[]={5,-12,-12,9,10,0,-9,-12,-1,23,65,64,11,43,39,-15};

// ініціалізація масиву без визначення його розміру.

У цьому випадку компілятор сам визначить розмір одновимірного масиву. Розмір масиву можна не вказувати тільки при його ініціалізації, при звичайному оголошенні масиву обов'язково потрібно вказувати розмір масиву.

Масиви використовують також для роботи з текстовими даними.

У мові програмування С++ для роботи з текстом можуть використовуватися С-рядки (масиви символів char[]) та об'єкти типу string.

Масив символів закінчується нульовим символом ('\0'). У символьний масив можна ввести відразу весь рядок, використовуючи оператор вводу.

Символьна константа – це один символ, укладений в апострофи.

Рядкова константа – це послідовність символів, укладена у подвійні лапки.

Індивідуальні завдання

1. У масиві цілих чисел, що задає завантаження процесора за 24 години, визначити максимальне значення.
2. У масиві цілих чисел, що задає обсяг переданих пакетів за 16 інтервалів часу, знайти сумарний обсяг переданих даних.
3. У масиві цілих чисел, що задає час відгуку сервера, визначити середнє значення.
4. У масиві цілих чисел, що задає кількість помилок у роботі мережі за певні проміжки часу, знайти кількість нульових значень.
5. У масиві цілих чисел, що задає рівень сигналу, знайти кількість від'ємних значень.
6. У масиві цілих чисел, що задає температури процесора, визначити,

скільки значень перевищують заданий поріг.

7. Для заданого масиву цілих чисел визначити номер першого елемента, що дорівнює заданому значенню.

8. У масиві цілих чисел, що задає розміри файлів журналу, знайти мінімальний і максимальний елементи.

9. У масиві цілих чисел, що задає швидкість передавання даних, замінити всі від'ємні значення на нуль.

10. У масиві цілих чисел, що задає номери портів, визначити кількість парних елементів.

11. У масиві цілих чисел, що задає значення байтів, підрахувати кількість елементів у діапазоні від 0 до 127.

12. У масиві цілих чисел, що задає коди помилок, відсортувати елементи за зростанням.

13. У масиві цілих чисел, що задає коди помилок, відсортувати елементи за спаданням.

14. У заданому символьному рядку підрахувати кількість цифр.

15. У заданому символьному рядку підрахувати кількість великих латинських літер.

16. У заданому символьному рядку підрахувати кількість малих латинських літер.

17. У заданому символьному рядку визначити кількість символів «.», «:» і «-».

18. Для заданого рядка, що містить IPv4-адресу, визначити кількість символів «.».

19. У заданому рядку замінити всі малі латинські літери на великі.

20. У заданому рядку визначити, який символ трапляється найчастіше.

21. У масиві цілих чисел, що задає послідовність контрольних значень, знайти суму всіх елементів з непарними індексами.

Додаткові умови виконання завдання:

- Обов'язково використовувати анотації *@author* і *@date*;
- звіт має бути виконаний згідно з вимогами до оформлення лабораторних робіт.

Контрольні запитання

1. Як рядки представляються в мові програмування C++?
2. Яке призначення індексів при оголошенні масиву символів, що

завершуються нульовим байтом (С–рядок)?

3. Що таке «мірність масиву»?
4. Від чого залежить об’єм пам’яті, що необхідний для зберігання елементів масиву?
5. Як можна звернутися до елементу масиву?
6. Як оголошуються масиви?
7. Які операції можуть бути застосовані до рядків типу char?

Лабораторна робота №5. Функції

Тема. Функції. Варіативні функції.

Мета: Набути практичні навички щодо розроблення програм з використанням функцій.

Загальні відомості

Функція – це синтаксично виділений іменований програмний модуль, що виконує певну дію або групу дій. Кожна функція має свій інтерфейс і реалізацію (визначення).

Інтерфейс функції – заголовок функції, у якому вказується назва функції, список її параметрів і тип значення, що повертається.

Прототип функції – це оголошення функції, але не її визначення.

Визначення (опис, реалізація) функції – це програмний код, який реалізує розроблений для даної функції алгоритм.

Не можна визначати будь-яку функцію в тілі іншої функції. У мові С/С++є два типи функцій:

- які не повертають значень;
- що повертають значення.

Функції, що не повертають значення по завершенню роботи, мають таке визначення:

void ім’я функції **(параметри функції)** // заголовок функції

```
{
    // тіло функції
}
```

Наприклад:

```
void printArr(int*, int);
```

Якщо потрібно функції передавати якісь дані, то усередині круглих

дужок оголошуються параметри функції, які відділяються один від одного комами.

Функції, що повертають значення по завершенню своєї роботи, визначають у такий спосіб:

тип даних, що повертаються ім’я функції **(параметри функції)**
{
// тіло функції
return *значення, що повертається*;
}

Наприклад:

int max (int, int);
float inputMassa();

return – оператор, який закінчує виконання функції. Він повертає виконання у ту функцію, з якої був виклик; управління передається наступному оператору за оператором виклику. Формат оператора такий:

return [<вираз>];

При виконанні *return* обчислюється значення виразу, якщо він є, приводиться до типу, який оголошений у функції, і повертається у ту функцію, з якої був виклик. У разі, коли вираз відсутній, значення, що повертається функцією, не визначено.

У C/C++ існує можливість помістити оголошення функцій в окремий файл; тоді такий файл із функціями необхідно буде підключати, як у випадку з підключенням стандартних заголовних файлів. Є два способи розміщення функцій:

- створення файлу типу *.c (*.cpp), у якому оголошуються та визначаються функції;
- створення файлів типу *.c (для визначення) і *.h (*.hpp) (для оголошення функцій).

Переважним стилем програмування вважається другий спосіб.

Функції дозволяють зробити програму модульною, тобто розділити її на кілька функцій, які в сукупності виконують поставлене завдання. Ще один величезний плюс функцій у тому, що їх можна багаторазово використовувати.

Функція виконується в момент її виклику. Після оголошення до функції можна звертатися у програмі по імені. Якщо функція не повертає жодного результату, тобто оголошена як *void*, її виклик не може бути використаний як операнд більш складного виразу (наприклад, значення

такої функції не можна привласнити будь-якій змінній). Виклик функції можна використовувати як складову частину більш складного виразу, якщо вона повертає результат.

Прототип вказують у тих випадках, коли функція описується пізніше свого використання. Наприклад, можна оголосити функцію до *main*, викликати її з *main*, але описати тільки після *main*.

Формальні й фактичні параметри. Формальні параметри існують у прототипі і тілі визначення функції. Вони задаються деякими унікальними іменами і усередині функції доступні як локальні змінні.

Фактичні параметри існують в основній програмі. Вони вказуються при виклику функції на місці формальних. У момент виклику функції значення фактичних параметрів привласнюються формальним.

Черговість виклику та рекурсія. Одна функція викликається всередині іншої. Зокрема, усередині свого тіла функція може викликати саму себе. Таке явище називається рекурсією, а така функція – рекурсивною.

Способи передачі параметрів у функцію.

1) Передача параметрів за значенням. При виклику функції значення фактичного параметра копіюється в локальну змінну, доступну як формальний параметр усередині функції.

Передача параметрів за значенням має такі обмеження:

- з тіла функції не можна звернутися до будь-якого об’єкта, якщо він не є глобальним або якщо його ім’я перекрите однойменною локальною змінною;
- при передачі об’єктів виконується їх копіювання, як наслідок, у випадку великих об’єктів витрачається багато пам’яті.

2) Передача параметрів по посиланню. У цих випадках у функцію передається адреса об’єкта і, відповідно, робота усередині функції відбувається не з копією, а з оригіналом об’єкта.

Щоб параметр передавався по посиланню, досить у прототипі функції поставити знак & після типу параметра.

Наприклад, є функція:

```
void func1(int val, int& ref)
{
        val++;
        ref++;
}
```

В іншій функції є таке:

```
int a = 10, b = 10;
```

func1(a,b);

// **a = 10**, значення буде збільшено, але усередині функції, як локальне

// **b = 11**, буде збільшене значення зовнішньої змінної **b**

При цьому, навіть якщо імена формального і фактичного параметрів будуть однакові, жодної проблеми не виникне.

Варіативні функції – це функції зі змінною кількістю аргументів. У оголошенні та визначенні такої функції змінне число аргументів задається трьома крапками, обов'язково наприкінці списку формальних параметрів.

При цьому, для коректної роботи функції рекомендується, щоб перший параметр задавав кількість фактично переданих аргументів та/або їх типи (наприклад, як у функції *printf()*).

Для реалізації функцій зі змінною кількістю аргументів у мові програмування С потрібно підключити заголовний файл *stdarg.h*, для С++ – *cstdarg*.

Основне завдання

Створити функцію яка виконує обчислення у відповідності до індивідуального завдання. Продемонструвати роботу функції на декількох наборах вхідних даних (2..5).

Індивідуальні завдання

1. У масиві цілих чисел, що задає завантаження процесора за 24 години, визначити максимальне значення.
2. У масиві цілих чисел, що задає обсяг переданих пакетів за 16 інтервалів часу, знайти сумарний обсяг переданих даних.
3. У масиві цілих чисел, що задає час відгуку сервера, визначити середнє значення.
4. У масиві цілих чисел, що задає кількість помилок у роботі мережі за певні проміжки часу, знайти кількість нульових значень.
5. У масиві цілих чисел, що задає рівень сигналу, знайти кількість від'ємних значень.
6. У масиві цілих чисел, що задає температури процесора, визначити, скільки значень перевищують заданий поріг.
7. Для заданого масиву цілих чисел визначити номер першого елемента, що дорівнює заданому значенню.
8. У масиві цілих чисел, що задає розміри файлів журналу, знайти мінімальний і максимальний елементи.
9. У масиві цілих чисел, що задає швидкість передавання даних,

замінити всі від’ємні значення на нуль.

10. У масиві цілих чисел, що задає номери портів, визначити кількість парних елементів.

11. У масиві цілих чисел, що задає значення байтів, підрахувати кількість елементів у діапазоні від 0 до 127.

12. У масиві цілих чисел, що задає коди помилок, відсортувати елементи за зростанням.

13. У масиві цілих чисел, що задає коди помилок, відсортувати елементи за спаданням.

14. У заданому символьному рядку підрахувати кількість цифр.

15. У заданому символьному рядку підрахувати кількість великих латинських літер.

16. У заданому символьному рядку підрахувати кількість малих латинських літер.

17. У заданому символьному рядку визначити кількість символів «.», «:» і «-».

18. Для заданого рядка, що містить IPv4-адресу, визначити кількість символів «.».

19. У заданому рядку замінити всі малі латинські літери на великі.

20. У заданому рядку визначити, який символ трапляється найчастіше.

21. У масиві цілих чисел, що задає послідовність контрольних значень, знайти суму всіх елементів з непарними індексами.

22.

Контрольні запитання

1. Чому при передачі параметра за значенням усі зміни параметра у функції не відбиваються на значенні аргументу?
2. Скільки операторів *return* може бути в тілі функції?
3. Що таке «прототип функції», яке його призначення?
4. Як визначити список параметрів функції змінної довжини? Наведіть приклад.
5. Як описати функцію, яка не має значень, що повертаються?
6. Як описати, що функція не має аргументів?
7. Наведіть приклад функції, яка не має аргументів та нічого не повертає.
8. Які функції називаються варіативними?
9. Як описати функцію, яка має параметри за замовчуванням?
10. Скільки параметрів за замовчуванням може мати функція?

Лабораторна робота №6. Багатовимірні масиви

Тема. Функції. Робота з багатовимірними масивами. Використання функцій стандартної бібліотеки. Функція *rand()*.

Мета: Придбати практичні навички щодо розроблення програм з використанням функцій користувача, функції генерації псевдовипадкових чисел, а також передачі масиву як аргументу функції.

Теми для попереднього опрацювання:

- функції, інтерфейс функції;
- передача параметрів;
- одно- та двовимірні масиви.

Загальні відомості

Мова програмування C/C++ дозволяє створювати багатовимірні масиви. Найпростішим видом багатовимірного масиву є двовимірний масив.

Двовимірний масив – це масив одновимірних масивів, що декларується так:

тип ім'я_масиву [кількість_рядків][кількість_стовпців];

В перших квадратних дужках вказується кількість рядків двовимірного масиву, у других – кількість стовпців.

Наприклад, для оголошення двовимірного масиву цілих чисел з 10 рядків по 20 елементів у кожному рядку треба записати таке:

int d[10][20];

Для доступу до елементу масиву **d** з індексами **3**, **5** необхідно використовувати такий запис: **d[3][5]**

При оголошенні двовимірного масиву можна виконувати його ініціалізацію. Наприклад,

```
int a[5][3] = { {4, 7, 8}, {9, 66, -1}, {5, -5, 0},
                {3, -3, 30}, {1, 1, 1} };
```

Після знака присвоєння ставляться загальні фігурні дужки, всередині яких ставиться стільки пар фігурних дужок, скільки має бути рядків у двовимірному масиві, причому ці дужки розділяються комами. У кожній парі фігурних дужок записуються через кому елементи двовимірного масиву. У всіх фігурних дужках кількість елементів має збігатися.

Перетворення одновимірного масиву у двовимірний передбачає наступне. Наприклад, задано одновимірний масив:

[1 2 3 5 6 7 8 9 0]

Необхідно перетворити його у такий масив:

$$\begin{vmatrix} 1 & 2 & 3 \\ 5 & 6 & 7 \\ 8 & 9 & 0 \end{vmatrix}$$

Часто на етапі налагодження програм у масиви записують випадкові числа. Випадкові числа в C/C++ можуть бути генеровані функцією *rand()* із стандартної бібліотеки <*cstdlib*>. Функція генерує числа в діапазоні від 0 до *RAND_MAX* (константа, яка визначена в бібліотеці <*cstdlib*>).

Для отримання різної послідовності випадкових чисел при кожному наступному запуску програми треба використовувати функцію *srand( )*.

srand(time(0)); // автоматична рандомізація

Основне завдання:

- розробити функцію, яка на вході приймає одновимірний масив. Одновимірний масив розміром *M*N* елементів перетворити у двовимірний масив, що має *M* рядків і *N* стовпців. Отриманий двовимірний масив обробити відповідно до індивідуального завдання і вивести його на екран;
- одновимірний масив заповнити псевдовипадковими числами (функція *rand ()*) у діапазоні від *5 * K* до *10 * K*, де *K* – номер варіанта.

Індивідуальні завдання

1. Дано двовимірний масив з N*N цілих чисел, що задає матрицю затримок між вузлами мережі. Транспонувати його.
2. Дано двовимірний масив з N*N цілих чисел, що задає матрицю обміну даними між вузлами. Помножити кожен елемент рядка на значення елемента головної діагоналі відповідного рядка.
3. Дано двовимірний масив з N*N цілих чисел, що задає матрицю службових параметрів мережі. Помножити кожен елемент рядка на значення елемента побічної діагоналі відповідного рядка.
4. Дано двовимірний масив з N*N цілих чисел, що задає таблицю завантаження каналів. Помножити кожен елемент рядка на середнє значення елементів відповідного рядка. Повернути суму середніх значень рядків матриці.
5. Дано двовимірний масив з N*N цілих чисел, що задає таблицю звернень до ресурсів. Помножити кожен елемент стовпця на середнє

значення елементів відповідного стовпця. Повернути добуток середніх значень стовпців матриці.

6. Дано двовимірний масив з N*N цілих чисел, що задає кількість пакетів у чергах маршрутизаторів. Упорядкувати за зростанням елементи кожного рядка окремо.

7. Дано двовимірний масив з N*M цілих чисел, що задає карту навантаження на мережеві інтерфейси. Виконати дзеркальне відображення матриці по горизонталі.

8. Дано двовимірний масив з N*N цілих чисел, що задає матрицю доступу. Поміняти елементи головної і побічної діагоналей місцями.

9. Дано двовимірний масив з N*N цілих чисел, що задає таблицю запитів до серверів. Додати до кожного елемента рядка елемент цього рядка, що має максимальне значення.

10. Дано двовимірний масив з N*N цілих чисел, що задає таблицю подій журналу. Додати до кожного елемента стовпця елемент цього стовпця, що має максимальне значення.

11. Дано двовимірний масив з N*N цілих чисел, що задає матрицю зв’язності вузлів. Помножити матрицю саму на себе відповідно до правил множення матриць.

12. Дано двовимірний масив з N*M цілих чисел, що задає таблицю параметрів пристроїв. Кожний елемент рядка помножити на випадкове число, причому для різних рядків числа-множники мають бути різними.

13. Дано двовимірний масив з N*M цілих чисел, що задає таблицю кодів подій. Повернути з функції середнє значення мінімальних елементів кожного рядка матриці.

14. Дано двовимірний масив з N*M цілих чисел, що задає таблицю рівнів сигналу. Повернути з функції середнє значення максимальних елементів кожного рядка матриці.

15. Дано двовимірний масив з N*N цілих чисел, що задає матрицю контролю доступу. Визначити, на скільки сума елементів, розташованих нижче головної діагоналі, більша або менша за суму елементів головної діагоналі.

16. Дано двовимірний масив з N*N цілих чисел, що задає таблицю обміну між вузлами. Попарно поміняти місцями елементи головної та побічної діагоналей.

17. Дано двовимірний масив з N*N цілих чисел, що задає таблицю параметрів мережевих сеансів. Виконати циклічне зрушення

елементів кожного рядка зліва направо.

18. Дано двовимірний масив з N*N цілих чисел, що задає таблицю значень контрольних перевірок. Поміняти місцями максимальний і мінімальний елементи масиву.

19. Дано двовимірний масив з N*N цілих чисел, що задає таблицю характеристик вузлів. Елементи головної діагоналі записати в одновимірний масив і впорядкувати його за зростанням.

20. Дано двовимірний масив з N*N цілих чисел, що задає матрицю маршрутів. Визначити, на скільки сума елементів, розташованих вище головної діагоналі, більша або менша за суму елементів, розташованих нижче головної діагоналі.

21. Дано двовимірний масив з N*N цілих чисел, що задає таблицю параметрів передавання даних. Виконати циклічне зрушення елементів кожного рядка справа наліво.

Додаткові умови виконання завдання:

- звіт має бути виконаний згідно з вимогами до оформлення лабораторних робіт.

Контрольні запитання

1. Що таке «вимірність масиву», якою вона може бути?
2. При доступі до елементів двовимірного масиву укажіть, яке призначення першого та другого індексів?
3. Від чого залежить обсяг пам’яті, що необхідний для збереження масива?
4. Чому при роботі з масивами використовують функцію rand()?
5. Як забезпечити генерацію іншої послідовності чисел при кожному наступному виконанні програми?
6. Дайте оцінку наступному оголошенню масиву:

   int mas[][2] = {{1,2}, {3,4}, {5,6}, {7,8}};
7. Як звернутися до елементу масиву? Наведіть приклад.
8. В якому порядку зберігаються у пам’яті комп’ютера елементи двовимірного масиву?
9. Скільки вимірів може мати масив?

Лабораторна робота №7. Структуровані типи даних

Тема. Робота із структурованими типами даних.

Мета: придбати практичні навички щодо розроблення програм із застосуванням структур.

Теми для попереднього опрацювання:

- структури, доступ до членів структури;
- покажчики на структури;
- об’єднання (union) та перерахування.

Загальні відомості

Структура — це складений тип даних, який створюється програмістом і являє собою сукупність змінних різних типів, об’єднаних спільною назвою. Оголошення структури задає шаблон, на основі якого створюються змінні структурного типу.

Змінні, що входять до складу структури, називаються членами структури. Їх також називають полями або елементами структури. Зазвичай члени структури логічно пов’язані між собою.

Ключове слово *struct* повідомляє компілятору про оголошення структури.

```
struct addr
{
  char name[30];
  char street[40];
};
```

Оголошення обов’язково завершується крапкою з комою, оскільки оголошення структури – це оператор. Ім’я структури (у прикладі – *addr*) ідентифікує структуру даних і є специфікатором типу.

Для оголошення змінної, відповідної до даної структури, слід написати:

```
struct addr addr_info;
```

Структура не може містити екземпляри самої себе, але можна визначати покажчики на структуру.

Для доступу до елементу структури необхідно вказати її ім’я, поставити крапку і вказати ім’я потрібного елементу. Для доступу до елементу структури через покажчик на об’єкт замість крапки використовується оператор «стрілка».

Об’єднання (union) – це складений тип даних; являє собою набір змінних різних типів, але які перебувають в одній і тій же області пам’яті. Пам’ять виділяється такого об’єму, щоб її було достатньо для зберігання найбільшого члена.

Для оголошення об’єднань використовується ключове слово *union*.

Приклад шаблону об’єднання з іменем типу *Telements*:

```
union Telements
{
  int number;
  char symbol;
  char *pointer;
};
```

Приклад оголошення даних типу суміші:

```
union Telements dat, mas[5], *pu;
```

Для доступу до членів об’єднання використовуються ті ж синтаксичні конструкції, що і для структури (крапка і стрілка).

Приклад доступу до елементів об’єднання:

```
dat.number=57;
mas[2].symbol='A';
pu=&dat;
x=pu->number;
```

Основне завдання

Розробити структуру з трьох полів відповідно до індивідуального завдання. Створити чотири змінні структурного типу та заповнити їх значеннями. При цьому:

- числові поля згенерувати за допомогою функції rand();
- рядкові поля ввести з клавіатури;
- числові значення генерувати у прийнятному діапазоні;
- після заповнення всі структурні змінні вивести на екран.

Індивідуальні завдання

1. Створити структуру Device, яка містить поля для назви пристрою, інвентарного номера та року введення в експлуатацію.
2. Створити структуру UserAccount, яка містить поля для імені користувача, ідентифікатора користувача та рівня доступу.
3. Створити структуру NetworkNode, яка містить поля для назви

вузла, номера вузла та IP-адреси.

4. Створити структуру Server, яка містить поля для назви сервера, обсягу оперативної пам’яті та типу операційної системи.

5. Створити структуру Workstation, яка містить поля для назви робочої станції, номера аудиторії та кількості встановлених програм.

6. Створити структуру Router, яка містить поля для моделі маршрутизатора, кількості портів та адреси встановлення.

7. Створити структуру Switch, яка містить поля для моделі комутатора, кількості активних портів та швидкості передавання даних.

8. Створити структуру AccessPoint, яка містить поля для назви точки доступу, номера каналу та зони покриття.

9. Створити структуру LogEntry, яка містить поля для номера запису, типу події та короткого опису.

10. Створити структуру SecurityEvent, яка містить поля для коду події, рівня небезпеки та джерела події.

11. Створити структуру FirewallRule, яка містить поля для номера правила, назви служби та статусу правила.

12. Створити структуру Packet, яка містить поля для номера пакета, протоколу та розміру пакета.

13. Створити структуру Session, яка містить поля для ідентифікатора сеансу, імені користувача та тривалості сеансу.

14. Створити структуру ProcessInfo, яка містить поля для назви процесу, ідентифікатора процесу та використання пам’яті.

15. Створити структуру SensorData, яка містить поля для назви датчика, поточного значення та одиниці вимірювання.

16. Створити структуру BackupFile, яка містить поля для назви архіву, розміру файла та дати створення.

17. Створити структуру DatabaseRecord, яка містить поля для номера запису, назви таблиці та статусу запису.

18. Створити структуру Connection, яка містить поля для номера з’єднання, типу протоколу та стану з’єднання.

19. Створити структуру AntivirusReport, яка містить поля для назви файла, результату перевірки та кількості виявлених загроз.

20. Створити структуру SystemModule, яка містить поля для назви модуля, номера версії та стану модуля.

Додаткові вимоги виконання завдання:

- звіт має бути виконаний згідно з вимогами до оформлення лабораторних робіт.

Контрольні запитання

1. Як виконати доступ до окремих елементів структури?
2. Чи можна вміст однієї структури присвоїти іншій того ж типу, використовуючи звичайний оператор присвоювання?
3. Коли необхідно використовувати покажчики на структури?
4. Чи може бути членом структури інша структура?
5. Чим відрізняється об’єднання від структури?
6. Що дозволяє визначити оператор *sizeof*? Чим він корисний?
7. Як працює функція *rand*()?

Лабораторна робота №8. Малювання в Visual Studio

Тема. Малювання графічних примітивів.

Мета: набути практичних навичок створення найпростішого графічного Windows-застосунку в середовищі Microsoft Visual Studio та використання засобів Win32 API і GDI для побудови графічних примітивів і виведення тексту у вікні програми.

Загальні відомості

У середовищі Microsoft Visual Studio графічні завдання виконуються у клієнтській області вікна за допомогою функцій Windows API та бібліотеки GDI.

Графічне виведення у Windows-застосунках здійснюється за допомогою контексту пристрою, який описується типом HDC. Контекст пристрою містить параметри малювання: поточне перо, пензель, шрифт, кольори ліній і тексту та інші атрибути, необхідні для побудови графічних об’єктів.

Для коректного відображення графіки малювання слід виконувати в обробнику повідомлення WM_PAINT. Контекст пристрою в такому разі отримують функцією BeginPaint(), а після завершення малювання звільняють функцією EndPaint().

Для побудови найпростіших графічних об’єктів можуть використовуватися такі функції:

- MoveToEx() і LineTo() – для побудови відрізків;
- Rectangle() – для побудови прямокутників;
- Ellipse() – для побудови кіл та еліпсів;
- Polygon() – для побудови багатокутників;
- TextOut() або DrawText() – для виведення тексту;
- SetTextColor() – для задання кольору тексту.

Для зміни параметрів малювання використовують графічні об’єкти HPEN і HBRUSH. Перо визначає колір, стиль і товщину контуру, а пензель – колір заповнення фігури. Перед використанням створені графічні об’єкти вибирають у контекст пристрою, а після завершення роботи їх потрібно коректно знищити.

Створення проєкту.

Для виконання лабораторної роботи необхідно створити проєкт типу **Windows Desktop Application** або **Windows Desktop Wizard** у середовищі Microsoft Visual Studio. Під час створення проекту потрібно обрати мову програмування **C++**, платформу **Windows** і тип проєкту **Desktop**.

Після створення проєкту програма повинна створювати головне вікно застосунку, обробляти повідомлення Windows і виконувати малювання в обробнику WM_PAINT.

Приклад найпростішої графічної програми

Нижче наведено приклад найпростішого Windows-застосунку, створеного в середовищі Microsoft Visual Studio. Програма відкриває вікно, у якому в обробнику WM_PAINT малюється рамка, коло, лінії та текстовий напис червоного кольору. Цей приклад демонструє базовий принцип роботи з функціями Win32 API і GDI та може бути використаний як основа для виконання індивідуального завдання.

Після створення проєкту в середовищі Visual Studio основний код малювання слід розміщувати в обробнику повідомлення WM_PAINT. У шаблоні проєкту потрібно знайти фрагмент:

```
case WM_PAINT:
{
PAINTSTRUCT ps;
HDC hdc = BeginPaint(hWnd, &ps);
// код малювання
```

```
EndPaint(hWnd, &ps);
}
break;
```

Саме **всередину цього блоку**, між викликами BeginPaint() і EndPaint(), потрібно додавати команди побудови графічних примітивів.

Нижче наведено приклад найпростішого фрагмента програми для побудови рамки, кола, лінії та текстового напису:

```
case WM_PAINT:
{
    PAINTSTRUCT ps;
    HDC hdc = BeginPaint(hWnd, &ps);

    // Малювання зовнішньої рамки
    Rectangle(hdc, 30, 30, 550, 350);

    // Малювання кола
    Ellipse(hdc, 100, 100, 200, 200);

    // Малювання лінії
    MoveToEx(hdc, 250, 120, NULL);
    LineTo(hdc, 400, 220);

    // Встановлення червоного кольору тексту
    SetTextColor(hdc, RGB(255, 0, 0));

    // Виведення підпису у вікні
    TextOut(hdc, 360, 320, L"Прізвище І.П.", 13);

    EndPaint(hWnd, &ps);
}
break;
```

Основне завдання

Створити в середовищі Microsoft Visual Studio графічний Windows-застосунок. У клієнтській області вікна побудувати рамку у вигляді прямокутника. Усередині рамки за допомогою не менше ніж трьох графічних елементів намалювати кольорове зображення згідно з

індивідуальним завданням. Для побудови рисунка дозволяється використовувати лінії, прямокутники, кола, еліпси, багатокутники та текстові написи.

У правому нижньому куті вікна необхідно вивести прізвище та ініціали студента червоним кольором.

Індивідуальні завдання

1. Мережевий вузол. Побудувати умовне позначення мережевого вузла у вигляді кола або прямокутника з підписом.
2. Сервер. Побудувати зображення серверного блока у вигляді прямокутника з індикаторами.
3. Комутатор. Побудувати зображення комутатора у вигляді прямокутника з рядом портів.
4. Маршрутизатор. Побудувати зображення маршрутизатора у вигляді прямокутника або еліпса з антенами чи стрілками передавання.
5. Точка доступу Wi-Fi. Побудувати зображення точки доступу з дугами радіосигналу.
6. Моніторинг мережі. Побудувати екран із графічним індикатором активності мережі.
7. Робоча станція. Побудувати зображення комп’ютера з монітором і системним блоком.
8. Ноутбук. Побудувати зображення ноутбука у відкритому вигляді.
9. Сховище даних. Побудувати умовне позначення накопичувача або бази даних.
10. Пакет даних. Побудувати умовне зображення пакета даних у вигляді прямокутного блока зі стрілкою руху.
11. Передавання даних. Побудувати схему з двох вузлів, з’єднаних лінією зв’язку.
12. Топологія мережі. Побудувати спрощену мережеву схему з кількох з’єднаних вузлів.
13. Екран авторизації. Побудувати вікно входу з полями імені користувача та пароля.
14. Замок безпеки. Побудувати зображення замка як символу захисту інформації.
15. Щит кіберзахисту. Побудувати зображення щита з умовним знаком безпеки.

16. Попередження про загрозу. Побудувати трикутний знак попередження з символом оклику.
17. Журнал подій. Побудувати умовне вікно журналу з кількома рядками записів.
18. Антивірусна перевірка. Побудувати екран перевірки з індикатором стану та галочкою.
19. Камера спостереження. Побудувати зображення камери як елемента системи безпеки.
20. Центр обробки даних. Побудувати спрощене зображення серверної стійки або групи серверів.

Додаткові вимоги виконання завдання:

– звіт має бути виконаний згідно з вимогами до оформлення лабораторних робіт.

Контрольні запитання

1. Що таке контекст пристрою HDC і для чого він використовується?
2. У якому обробнику повідомлень слід виконувати малювання у Windows-застосунку?
3. Для чого призначені функції BeginPaint() і EndPaint()?
4. Які функції використовують для побудови лінії, прямокутника та еліпса?
5. Яке призначення мають об’єкти HPEN і HBRUSH?
6. Як виконати виведення тексту у вікні програми?
7. Як задати колір тексту?
8. Чому віконний застосунок є кращим для графічних завдань, ніж консольне вікно?
9. Які дії потрібно виконати для коректного звільнення графічних ресурсів?
10. Що відбувається з графічним вмістом вікна після його перекриття або зміни розміру?

Лабораторна робота №9. Покажчики

Тема. Основи роботи з покажчиками. Робота з динамічними масивами.

Мета: набути практичних навичок щодо розроблення програм з динамічно створеними даними.

Теми для попереднього опрацювання:

- адресна арифметика;
- покажчики;
- масиви;
- виділення та звільнення пам’яті.

Загальні відомості

Кожна змінна, оголошена у програмі, має адресу, тобто номер комірки пам’яті, у якій вона розміщена. Адреса є однією з основних характеристик змінної. Можна оголосити іншу змінну, яка зберігатиме цю адресу. Така змінна називається покажчиком.

Покажчики використовуються під час передавання у функції параметрів, значення яких потрібно змінювати, під час роботи з масивами, динамічною пам’яттю, структурами даних та в інших випадках. Використання покажчиків дає змогу ефективно працювати з пам’яттю, організовувати динамічне створення даних і забезпечувати гнучкіші способи доступу до елементів програмних структур.

У мові C/C++ покажчик зберігає адресу об’єкта певного типу. Для роботи з покажчиками використовують операції отримання адреси та розіменування. Правильне застосування покажчиків є важливим для створення програм, що працюють із динамічними масивами, списками, рядками та іншими структурами даних.

Оголошення покажчика має такий синтаксис:

*<тип> *<ідентифікатор> [= <ініціалізатор>];*

Покажчик може вказувати на значення базового типу, структури, об’єднання, функції, покажчика. Наприклад,

```
int *pi; // Покажчик на int
```

Для роботи з покажчиками використовують такі основні операції:

- операція отримання адреси (адресація) &;

- операція отримання значення за адресою (непряма адресація або розіменування) *.

*int a, *p;*

p = &a; // змінній p присвоюється адреса змінної a

**p = 0; // Значення за адресою, що знаходиться у змінній p (тобто значення змінної a), буде дорівнювати 0*

Функція *malloc()* визначена у бібліотеці *stdlib.h*. Функція malloc() використовується для динамічного виділення блоку пам'яті потрібного розміру. Пам'ять виділяється з сектора оперативної пам'яті, що доступний для будь-яких програм, які виконуються на даній машині.

Оскільки різні типи даних мають різні вимоги до пам'яті, треба навчитися визначати розмір пам'яті в байтах для різного типу даних. Наприклад, пам'ять для масиву елементів типу *int* – це один обсяг пам'яті, а якщо необхідно виділити пам'ять для масиву такого ж розміру, але вже типу char – це буде інший обсяг. Для визначення розміру в байтах використовується оператор sizeof. Наприклад, *sizeof(int)* поверне кількість байтів, необхідних для зберігання даних типу int.

Звільнення пам'яті виконується за допомогою функції *free()*.

Основне завдання

1. Всі масиви в індивідуальному завданні – динамічні.
2. Розмірність масивів ввести клавіатури
3. Масиви заповнити випадковими числами.
4. В кінці кожної програми вивести в консоль розмір пам'яті яку займає оброблений масив та ім'я автора розробника коду

Індивідуальні завдання

1. Дано динамічний масив із N цілих чисел, що задає час відгуку мережевого вузла. Визначити мінімальне і максимальне значення та створити другий масив з елементів, розташованих між ними.
2. Дано динамічний масив із N цілих чисел, що задає розміри пакетів даних. Визначити, чи є в масиві однакові значення. Якщо такі є, створити масив повторюваних елементів.
3. Дано динамічний масив із N цілих чисел, що задає коди подій журналу. Підрахувати кількість додатних, від'ємних і нульових елементів та записати результати в окремий масив.

4. Дано динамічний масив із N цілих чисел, що задає рівні сигналу. Створити другий масив, який міститиме всі елементи, більші за середнє арифметичне початкового масиву.
5. Дано динамічний масив із N цілих чисел, що задає кількість звернень до сервера за послідовні проміжки часу. Знайти найдовшу неперервну неспадну підпослідовність і записати її в окремий масив.
6. Дано динамічний масив із N цілих чисел, що задає значення контрольних перевірок. Створити другий масив, який міститиме лише парні елементи початкового масиву.
7. Дано динамічний масив із N цілих чисел, що задає номери портів. Вилучити з масиву всі елементи, що повторюються, і сформувати новий масив з унікальних значень.
8. Дано динамічний двовимірний масив розміру N×N, що задає матрицю затримок між вузлами мережі. Мінімальні елементи кожного рядка записати в одновимірний масив.
9. Дано динамічний двовимірний масив розміру N×N, що задає матрицю обміну даними. Максимальні елементи кожного стовпця записати в одновимірний масив.
10. Дано два динамічні масиви: mas1[N] і mas2[M], що задають послідовності мережевих подій. Створити третій масив, до якого спочатку записати елементи mas1, а потім елементи mas2. Отриманий масив упорядкувати за зростанням.
11. Дано два динамічні масиви: mas1[N] і mas2[M], що задають параметри двох каналів зв’язку. Створити третій масив, у який почергово записувати по одному елементу з кожного вхідного масиву.
12. Дано динамічний масив із N цілих чисел, що задає обсяг використаної пам’яті процесами. Визначити кількість пар сусідніх елементів з однаковими значеннями та записати ці пари в окремий масив.
13. Дано динамічний масив із N цілих чисел, що задає значення байтів. Упорядкувати елементи так, щоб усі додатні розташовувалися на початку масиву, усі від’ємні — у кінці, а нульові — між ними.
14. Дано динамічний двовимірний масив розміру N×N, що задає матрицю параметрів пристроїв. Елементи головної діагоналі записати в одновимірний масив і впорядкувати його за зростанням.

15. Дано динамічний двовимірний масив розміру N×N, що задає матрицю параметрів пристроїв. Елементи побічної діагоналі записати в одновимірний масив і впорядкувати його за спаданням.

16. Дано динамічний масив із N цілих чисел, що задає коди доступу. Підрахувати кількість неперервних ділянок, елементи яких утворюють неспадну послідовність. Найдовшу таку ділянку записати в окремий масив.

17. Дано динамічний масив із N цілих чисел, що задає результати перевірки файлів. Знайти всі елементи, що трапляються більше одного разу, і записати їх у другий масив без повторень.

18. Дано динамічний масив із N цілих чисел, що задає навантаження на мережеві інтерфейси. Знайти неперервну підпослідовність елементів з максимальною сумою та записати її в окремий масив.

19. Дано динамічний двовимірний масив розміру N×N, що задає таблицю мережевих показників. Для кожного рядка визначити кількість додатних елементів і записати результати в одновимірний масив.

20. Дано динамічний масив із N цілих чисел, що задає ідентифікатори подій безпеки. Створити другий масив, який міститиме всі елементи, розташовані між першим і другим від'ємними елементами початкового масиву.

Додаткові умови виконання завдання:

- звіт має бути виконаний згідно з вимогами до оформлення лабораторних робіт;

Контрольні запитання

1. Як створити покажчик на масив?
2. Які операції можуть бути застосовані до покажчиків?
3. Як здійснюється звільнення пам'яті?
4. Як здійснюється виділення пам'яті?
5. Як створюється контроль за витоком пам'яті?
6. Чим відрізняється статичний масив від динамічного?
7. Як визначити поточний обсяг пам'яті для динамічного масиву?
8. Чому треба звільняти пам'ять, яка динамічно виділялась?

Лабораторна робота №10. Рядки типу char*

Тема. Робота з рядками типу char*.

Мета: Набути практичних навичок щодо розроблення програм з використанням рядків.

Теми для попереднього опрацювання:

- масиви;
- функції;
- рядки;
- покажчики.

Загальні відомості

Рядок – це одновимірний масив символів, останнім елементом якого є символ кінця рядка – нуль (рядок, що завершується нулем, тобто NULL *terminated string).*

Оголосити змінну типу рядок можна трьома способами:

1. Оголосити масив символів фіксованої довжини (із місцем під завершальний нульовий символ ‘\0’). Наприклад, для рядка довжиною до 40 символів:

   char s[40 + 1];

2. Оголосити масив і ініціалізувати його рядковим літералом (довжину масиву обчислює компілятор):

   char s [] = "Приклад ініціалізації рядка";

Під час ініціалізації в кінець рядка автоматично додається нульовий символ ‘\0’. У цьому варіанті створюється масив, що містить копію рядкового літерала, тому елементи такого рядка можна змінювати.

3. Оголосити вказівник на символ і ініціалізувати його адресою рядкового літерала:

   *char *s = "Другий варіант ініціалізації";*

У цьому випадку змінна s є вказівником, який зберігає адресу першого символу рядкового літерала. Рядковий літерал має статичну тривалість зберігання та не призначений для зміни, тому використовується тип const char. Поширена помилка полягає в ототожненні “рядка” з “вказівником”: вказівник лише вказує на послідовність символів і не виділяє пам’ять для змінюваного рядка.

string.h – заголовок стандартної бібліотеки мови С, що містить

функції для роботи з нуль-термінованими рядками.

Для доступу до окремого символу у рядку треба вказати назву рядка та у квадратних дужках – номер символу. Нумерація символів у рядках починається з 0.

Загальне завдання

1. В кінці кожної програми вивести в консоль ім’я автора розробника коду

2. Текст задавати рядковою константою, якщо інше не передбачено умовою завдання.

Індивідуальне завдання

1. Визначити, скільки разів у тексті зустрічається слово «сервер».
2. Визначити, скільки у тексті IPv4-адрес.
3. Визначити, скільки у тексті MAC-адрес.
4. У кожному рядку тексту замінити всі символи табуляції на один пробіл.
5. У кожному рядку тексту видалити зайві пробіли між словами, залишивши між словами лише один пробіл.
6. Визначити кількість рядків, у яких міститься слово «error».
7. Визначити кількість рядків, у яких міститься слово «warning».
8. Знайти в тексті всі номери портів і вивести їх окремо.
9. Визначити, скільки разів у тексті зустрічаються доменні імена із зоною .com.
10. Для кожного рядка тексту визначити, чи є він коректним записом електронної адреси.
11. У тексті замінити всі входження слова «admin» на слово «user».
12. Для кожного рядка тексту визначити, чи починається він з цифри.
13. Для кожного рядка тексту визначити, чи закінчується він символом «;».
14. Визначити, скільки у тексті рядків, що містять одночасно цифри і літери.
15. У тексті знайти всі слова, довжина яких більша за 8 символів.
16. Для кожного рядка тексту підрахувати кількість символів «/», «\» і «:».
17. У тексті замінити всі малі латинські літери на великі.
18. Визначити, чи є в кожному рядку тексту правильно розставлені дужки (), {}, [].

19. Для кожного рядка тексту виконати маскування: усі цифри, крім останніх двох, замінити символом «*».
20. Для кожного рядка тексту виконати циклічне зрушення символів на n позицій праворуч. Значення n ввести з клавіатури.

Додаткові вимоги виконання завдання

- звіт має бути виконаний згідно з вимогами до оформлення робіт;

Контрольні запитання

1. Як «склеїти» два рядки?
2. Як визначити, чи є в заданому рядку заданий підрядок?
3. Чому рядки в мові С закінчуються «\0» ?
4. Які функції використовують для введення С-рядків з клавіатури?
5. Як порівняти два рядки?
6. Як ввести рядок (символ) з клавіатури?
7. Як видати рядок (символ) на екран?
8. Як у заданому рядку видалити заданий підрядок?

Лабораторна робота №11. Робота з файлами

Тема. Основи роботи з файлами.

Мета: Набути практичних навичок розроблення програм із використанням файлів для зчитування та збереження даних.

Теми для попереднього опрацювання:

- файли;
- рядки;
- покажчики.

Загальні відомості

Файл – це іменована галузь даних на будь-якому носії інформації. Для програміста відкритий файл представляється як послідовність зчитуваних або записуваних даних. При відкритті файлу з ним зв’язується потік введення- виведення. Інформація, що виводиться, записується в потік, а інформація, що вводиться, зчитується з потоку.

Коли потік відкривається для введення-виведення, він зв’язується зі стандартною структурою типу *FILE*, яка визначена у файлі *stdio.h*.

Структура *FILE* містить необхідну інформацію про файл.

Робота з файлами розподіляється на три етапи:

Відкриття файлу. Функція fopen() повертає покажчик на потік типу FILE*. У разі помилки повертається NULL.

FILE *fopen(name, type);

де *name* – ім’я файлу, що відкривається (включаючи шлях),

type – покажчик на рядок символів, що визначають спосіб доступу до файлу:

"r" – відкрити файл для читання (файл повинен існувати);

"w" – відкрити порожній файл для запису; якщо файл існує, то його вміст губиться;

"a" – відкрити файл для запису в кінець (для додавання); файл створюється, якщо він не існує;

"r+" – відкрити файл для читання і запису (файл повинен існувати);

"w+" – відкрити порожній файл для читання і запису; якщо файл існує, то його вміст губиться;

"a+" – відкрити файл для читання і доповнення, якщо файл не існує, то він створюється.

Значення, що вертається, – покажчик на відкритий потік. Якщо виявлена помилка, то вертається значення NULL.

*Читання і запис даних.*Для запису даних у файл використовують функції fprintf(), fputs(), fputc().

Для читання даних із файлу використовують функції fscanf(), fgets(), fgetc().

Закриття файлу. Після завершення роботи з файлом його потрібно закрити за допомогою функції fclose().

Стандартні функції роботи з файлами містяться в заголовному файлі stdio.h.

Основне завдання

У програму, що розроблена в попередній лабораторній роботі (робота з рядками), виконати наступні зміни:

- читання даних виконувати не з клавіатури, а з файлу.
- видача даних проводиться і у консоль і у окремий файл.

Додаткові вимоги виконання завдання:

- звіт має бути виконаний згідно з вимогами до оформлення робіт;

Контрольні запитання

1. Що таке файл?
2. Які існують функції неформатованого введення даних?
3. Які існують функції неформатованого виведення даних?
4. Як визначити розмір файлу?
5. Як виконувати читання даних з файлу, коли кількість їх невідома?
6. Чи можна форматувати дані при їх записі у файл?
7. Як ввести з файлу та записати у файл рядки?
8. Як ввести з файлу та записати у файл символ?
9. Порівняйте текстові та двійкові файли.

Лабораторна робота №12. Рекурсивні функції в мові програмування C/C++

Тема. Основи роботи з рекурсивними функціями в мові програмування C/C++.

Мета: Набути практичних навичок щодо розроблення програм із використанням рекурсивних функцій та їх використанням на мові програмування C/C++. Розвиток навичок аналізу рекурсивних алгоритмів та їх реалізація в програмному коді.

Теми для попереднього опрацювання:

- функції;
- масиви;
- рядки;
- умовні оператори;
- цикли.

Загальні відомості

Рекурсія в програмуванні — це спосіб організації обчислень, за якого функція викликає саму себе прямо або опосередковано. Рекурсивні функції використовують для розв’язання задач, які можна поділити на подібні підзадачі меншого розміру. Важливим поняттям рекурсії є базовий випадок – умова, за якої рекурсивні виклики припиняються, що дає змогу уникнути нескінченного виконання функції.

Принцип роботи. Коли рекурсивна функція викликає сама себе, вона робить це з новим набором параметрів, приближаючись до базового випадку. Кожен виклик функції зберігається в стеку викликів до тих пір,

поки не буде досягнуто базового випадку. Після цього стек починає розгортатись, повертаючи контроль до попередніх рівнів виклику, аж до першого виклику функції.

Переваги рекурсії:

- спрощення програмного коду для окремих класів задач;
- природність розв'язання задач, які мають рекурсивну структуру;
- зручність реалізації деяких алгоритмів обробки дерев, пошуку, перебору та математичних обчислень.

Недоліки рекурсії:

- додаткові витрати пам'яті на зберігання стеку викликів;
- можливість переповнення стеку при великій глибині рекурсії;
- у деяких випадках менша ефективність порівняно з ітеративними алгоритмами.

Основні терміни:

Базовий випадок **–** умова , за якої рекурсія припиняється.

Рекурсивний випадок **–** частина функції, у якій виконується рекурсивний виклик із новими параметрами.

Стек викликів **–** структура даних, у якій зберігається інформація про активні виклики функцій і їх параметри.

Основне завдання

Написати програму на мові C/C++, яка використовує рекурсивну функцію для обчислення індивідуального завдання. Всі дані вводяться користувачем через консольний ввід, окрім елементів масиву які заповняються за допомогою генератора випадкових чисел (якщо це необхідно до індивідуального завдання).

Індивідуальне завдання

1. Рекурсивно обчислити суму елементів масиву, що задає розміри мережевих пакетів.
2. Рекурсивно знайти максимальний елемент масиву, що задає час відгуку мережевих вузлів.
3. Рекурсивно знайти мінімальний елемент масиву, що задає рівні сигналу.
4. Рекурсивно визначити кількість активних портів у масиві, де значення 1 означає активний стан, а 0 — неактивний.
5. Рекурсивно визначити кількість нульових елементів у масиві, що

задає коди станів системи.

6. Рекурсивно обчислити добуток усіх додатних елементів масиву, що задає контрольні значення.

7. Рекурсивно перевірити, чи впорядкований за зростанням масив, що задає затримки передавання даних.

8. Рекурсивно вивести елементи масиву, що задає коди подій журналу, у зворотному порядку.

9. Рекурсивно обчислити суму цифр числового ідентифікатора користувача або пристрою.

10. Рекурсивно визначити кількість цифр у числовому ідентифікаторі події безпеки.

11. Рекурсивно обчислити степінь числа, що задає коефіцієнт зростання навантаження на систему.

12. Рекурсивно визначити, скільки разів заданий символ зустрічається в рядку, що містить запис журналу подій.

13. Рекурсивно обчислити довжину рядка, що містить ім’я хоста або доменне ім’я.

14. Рекурсивно виконати обернення рядка, що містить назву мережевого пристрою.

15. Рекурсивно перевірити, чи є символьний рядок, що задає ідентифікатор, паліндромом.

16. Рекурсивно підрахувати кількість великих латинських літер у рядку, що містить назву системного модуля.

17. Рекурсивно підрахувати кількість цифр у рядку, що містить серійний номер обладнання.

18. Рекурсивно знайти перший від’ємний елемент у масиві, що задає результати перевірки з’єднань, і визначити його позицію.

19. Рекурсивно обчислити суму елементів головної діагоналі квадратної матриці, що задає матрицю затримок між вузлами.

20. Рекурсивно визначити кількість елементів масиву, що перевищують заданий поріг і задають навантаження на мережеві інтерфейси.

Додаткові вимоги виконання завдання:

- звіт має бути виконаний згідно з вимогами до оформлення робіт;

Контрольні запитання

1. Що таке рекурсія в програмуванні?
2. Як працює рекурсивна функція на прикладі обчислення факторіалу числа?
3. Які основні компоненти має рекурсивна функція?
4. Що таке базовий випадок і чому він є обов’язковим?
5. Як рекурсія впливає на стек викликів програми?
6. Що може статися при надмірній глибині рекурсії?
7. У чому полягають переваги і недоліки рекурсії порівняно з ітерацією?
8. Які підходи можна використати для перетворення рекурсивного алгоритму на ітеративний?

Лабораторна робота №13. Використання динамічних числових масивів

Тема. Робота з динамічними числовими масивами в мові програмування C/C++.

Мета: Набути практичних навичок роботи з динамічними одновимірними та двовимірними масивами, використання функцій динамічного виділення пам’яті, звертання до елементів масиву за допомогою індексів і покажчиків, а також звільнення виділеної пам’яті після завершення роботи програми.

Теми для попереднього опрацювання:

- масиви;
- покажчики;
- функції;
- динамічне виділення пам’яті;
- двовимірні масиви.

Загальні відомості

Динамічний масив відрізняється від звичайного тим, що пам’ять під нього виділяється під час виконання програми, а кількість елементів може задаватися змінною. Це дає змогу створювати масиви, розмір яких визначається лише після запуску програми.

Для створення динамічних масивів у мові C/C++ можуть

використовуватися функції malloc() і calloc(). Функція malloc() виділяє потрібний обсяг пам’яті без її ініціалізації, а функція calloc() додатково заповнює виділену пам’ять нульовими значеннями.

Синтаксис оголошення динамічного одновимірного масиву за допомогою malloc() має вигляд:

тип *ім’я = (тип*)malloc(sizeof(тип) * кількість_елементів);

Наприклад:

int N = 10;

float *a = (float*)malloc(sizeof(float) * N);

Оголошення динамічного одновимірного масиву за допомогою calloc():

int N = 10;

float *a = (float*)calloc(N, sizeof(float));

Після завершення роботи з динамічним масивом виділену пам’ять потрібно звільнити за допомогою функції free():

free(a);

До елементів динамічного одновимірного масиву можна звертатися як за допомогою індексів, так і за допомогою покажчиків. Наприклад, запис a[i] еквівалентний запису *(a + i).

Двовимірний динамічний масив можна організувати двома способами. Перший спосіб полягає у виділенні пам’яті під один суцільний блок із m*n елементів. У такому разі масив фактично зберігається в пам’яті як одновимірний, а доступ до елемента a[i][j] виконується через адресний вираз.

Наприклад, оголошення такого масиву:

float *a = (float*)malloc(m * n * sizeof(float));

або

float *a = (float*)calloc(m * n, sizeof(float));

Другий спосіб полягає у створенні масиву покажчиків на рядки. У цьому випадку спочатку виділяється пам’ять під масив покажчиків, а потім окремо під кожний рядок матриці:

float **a = (float**)malloc(m * sizeof(float*));

for (int i = 0; i < m; i++)

a[i] = (float*)malloc(n * sizeof(float));

Перевагою такого підходу є можливість звертатися до елементів у звичному вигляді – a[i][j].

Отже, використання динамічних масивів дає змогу гнучко працювати

з даними змінного розміру, однак вимагає обов’язкового контролю за коректним виділенням і звільненням пам’яті.

Основне завдання

Розробити програму мовою C/C++, у якій використовується динамічний числовий масив для розв’язання індивідуального завдання. Під час виконання роботи необхідно:

- організувати динамічне виділення пам’яті під масив або матрицю;
- реалізувати введення, оброблення та виведення даних;
- використати звертання до елементів масиву за індексами або за допомогою покажчиків;
- після завершення роботи програми звільнити всю динамічно виділену пам’ять.

Індивідуальне завдання

1. Заповнити матрицю випадковими числами, що імітують затримки між мережевими вузлами. Знайти суму всіх елементів матриці.
2. Заповнити матрицю випадковими числами, що імітують завантаження каналів зв’язку. Знайти середнє арифметичне всіх елементів матриці.
3. Заповнити матрицю випадковими числами, що імітують рівні сигналу. Визначити кількість додатних, від’ємних і нульових елементів.
4. Заповнити матрицю випадковими числами, що імітують час відгуку серверів. Знайти найбільший елемент матриці та його індекси.
5. Заповнити матрицю випадковими числами, що імітують значення контрольних перевірок. Знайти найменший елемент матриці та його індекси.
6. Заповнити матрицю випадковими числами, що імітують кількість пакетів у чергах маршрутизаторів. Обчислити суму елементів кожного рядка.
7. Заповнити матрицю випадковими числами, що імітують інтенсивність обміну даними. Обчислити суму елементів кожного стовпця.
8. Заповнити матрицю випадковими числами, що імітують параметри з’єднань. Знайти добуток додатних елементів кожного рядка.
9. Заповнити матрицю випадковими числами, що імітують коди станів пристроїв. Знайти кількість парних елементів у кожному рядку.
10. Заповнити матрицю випадковими числами, що імітують мережеві

показники. Знайти кількість непарних елементів у кожному стовпці.

11. Заповнити квадратну матрицю випадковими числами, що імітують затримки в мережі. Обчислити суму елементів головної діагоналі.

12. Заповнити квадратну матрицю випадковими числами, що імітують службові параметри системи. Обчислити суму елементів побічної діагоналі.

13. Заповнити матрицю випадковими числами, що імітують навантаження на мережеві інтерфейси. Знайти найбільший елемент у кожному рядку.

14. Заповнити матрицю випадковими числами, що імітують результати перевірки вузлів. Знайти найменший елемент у кожному стовпці.

15. Заповнити матрицю випадковими числами, що імітують таблицю маршрутів. Поміняти місцями перший і останній рядки матриці.

16. Заповнити матрицю випадковими числами, що імітують таблицю доступу. Поміняти місцями перший і останній стовпці матриці.

17. Заповнити матрицю випадковими числами, що імітують журнал подій. Вивести елементи матриці у зворотному порядку за рядками.

18. Заповнити матрицю випадковими числами, що імітують результати діагностики мережі. Замінити всі від'ємні елементи на нулі.

19. Заповнити матрицю випадковими числами, що імітують числові ідентифікатори подій. Замінити всі парні елементи на їх квадрати.

20. Заповнити матрицю випадковими числами, що імітують параметри передавання даних. Для кожного рядка знайти середнє арифметичне його елементів.

Додаткові вимоги виконання завдання:

- звіт має бути виконаний згідно з вимогами до оформлення робіт;

Контрольні запитання

1. Для чого використовуються динамічні масиви?
2. За допомогою яких функцій можна створити динамічний масив?
3. За допомогою якої функції звільняється пам'ять, виділена під динамічний масив?
4. Яким чином можна звертатися до елементів динамічного масиву?
5. Що таке адресний вираз під час звертання до елементів масиву?

6. У чому полягає різниця між одновимірним і двовимірним динамічним масивом?
7. Які способи організації двовимірного динамічного масиву існують?
8. У якому порядку потрібно звільняти пам’ять, виділену під двовимірний динамічний масив?

Лабораторна робота №14. Розробка багатофайлових проєктів

Тема. Розробка багатофайлових проєктів у мові програмування C/C++.

Мета: Набути практичних навичок розроблення багатофайлових програм, розподілу коду між кількома файлами, використання заголовних файлів, оголошення та визначення функцій у різних модулях, а також організації структури проєкту в середовищі розробки.

Теми для попереднього опрацювання:

- функції;
- масиви;
- рядки;
- покажчики;
- динамічні масиви;
- робота з файлами.

Загальні відомості

У процесі розроблення програм невеликі навчальні приклади часто розміщують в одному файлі. Проте зі збільшенням обсягу програмного коду та кількості функцій такий підхід стає незручним. Великі програми, як правило, поділяють на окремі модулі, кожен з яких відповідає за виконання певної частини задачі. Такий підхід називають **багатофайловою організацією проєкту**.

Під час розроблення багатофайлових проєктів програмний код поділяють на кілька типів файлів:

- **файли з вихідним кодом** (.c, .cpp) — містять визначення функцій і реалізацію алгоритмів;
- **заголовні файли** (.h) — містять оголошення функцій, констант, структур, макросів і типів даних, які використовуються в кількох файлах проєкту.

Поділ програми на кілька файлів дає змогу:

- зробити структуру програми зрозумілішою;
- спростити відлагодження та супровід коду;
- повторно використовувати окремі модулі;
- організувати командну розробку, коли різні частини проєкту створюються окремо.

Заголовний файл підключається до інших файлів за допомогою директиви препроцесора #include. Наприклад:

#include "functions.h"

У заголовному файлі зазвичай записують лише оголошення функцій, а їх реалізацію розміщують в окремому файлі вихідного коду. Наприклад:

Файл functions.h

```
int sum(int a, int b);
int max(int a, int b);
```

Файл functions.cpp

```
int sum(int a, int b)
{
   return a + b;
}

int max(int a, int b)
{
   return (a > b) ? a : b;
}
```

Файл main.cpp

```
#include <iostream>
#include "functions.h"

int main()
{
   int x = 5, y = 8;
   std::cout << "Сума = " << sum(x, y) << std::endl;
   std::cout << "Максимум = " << max(x, y) << std::endl;
   return 0;
}
```

Під час розроблення багатофайлового проєкту важливо дотримуватися таких правил:

- у заголовних файлах розміщувати лише оголошення;
- у файлах реалізації розміщувати визначення функцій;
- імена файлів і функцій мають бути змістовними;
- структура проєкту має бути логічною та зрозумілою.

Основне завдання

Розробити багатофайловий проєкт мовою C/C++, який складається щонайменше з трьох файлів:

- головного файла програми;
- заголовного файла;
- файла реалізації функцій.

Індивідуальне завдання

1. Розробити програму для аналізу параметрів мережевого з'єднання: визначення мінімальної, максимальної та середньої затримки. Кожну дію оформити окремою функцією.
2. Розробити програму для роботи з масивом, що задає розміри мережевих пакетів: обчислення суми елементів, пошук найбільшого елемента та кількості пакетів, розмір яких перевищує задане значення.
3. Розробити програму для аналізу журналу подій: підрахунок кількості символів, цифр і пробілів у рядку. Кожну дію оформити окремою функцією.
4. Розробити програму для роботи з рядком, що містить мережеве ім'я вузла: визначення довжини рядка, кількості цифр і кількості символів-розділювачів.
5. Розробити програму для перевірки параметрів доступу: визначення, чи є число додатним, парним і таким, що належить заданому діапазону. Кожну перевірку оформити окремою функцією.
6. Розробити програму для роботи з масивом кодів подій: знаходження найменшого елемента, найбільшого елемента та кількості нульових значень.
7. Розробити програму для роботи з матрицею, що задає затримки між вузлами: обчислення суми елементів головної діагоналі, пошук максимального елемента та кількості від'ємних елементів.
8. Розробити програму для роботи з матрицею, що задає навантаження на канали зв'язку: знаходження суми елементів кожного рядка та кожного стовпця.

9. Розробити програму для аналізу символьного рядка, що містить запис події: визначення кількості слів, кількості великих латинських літер і кількості цифр.
10. Розробити програму для роботи з одновимірним масивом, що задає рівні сигналу: обчислення середнього арифметичного, кількості додатних елементів та кількості елементів, менших за середнє значення.
11. Розробити програму для аналізу параметрів пристрою: обчислення периметра і площі прямокутної плати за заданими сторонами, а також довжини її діагоналі. Кожну дію оформити окремою функцією.
12. Розробити програму для роботи з файлами: запис масиву значень датчиків у файл, зчитування масиву з файла та виведення його на екран. Кожну дію оформити окремою функцією.
13. Розробити програму для сортування масиву числових ідентифікаторів за зростанням і за спаданням. Кожен варіант сортування оформити окремою функцією.
14. Розробити програму для перевірки рядка, що містить умовний пароль: визначення його довжини, кількості цифр і кількості великих латинських літер.
15. Розробити програму для аналізу таблиці параметрів мережі: знаходження мінімального елемента кожного рядка та максимального елемента кожного стовпця.
16. Розробити програму для роботи з масивом значень завантаження процесора: визначення мінімального, максимального та середнього значень. Кожну дію оформити окремою функцією.
17. Розробити програму для аналізу списку номерів портів: визначення кількості парних, непарних і нульових значень. Кожну дію оформити окремою функцією.
18. Розробити програму для роботи з рядком, що містить IPv4-адресу: визначення кількості цифр, кількості символів «.» та загальної довжини рядка.
19. Розробити програму для аналізу матриці, що задає коди станів пристроїв: визначення кількості додатних, від’ємних і нульових елементів. Кожну дію оформити окремою функцією.
20. Розробити програму для роботи з масивом розмірів файлів журналу: знаходження суми елементів, середнього значення та кількості елементів, що перевищують заданий поріг. Кожну дію оформити окремою функцією.

Додаткові вимоги виконання завдання:

- звіт має бути виконаний згідно з вимогами до оформлення робіт;

Контрольні запитання

1. Що таке багатофайловий проєкт?
2. Для чого програму поділяють на кілька файлів?
3. Яке призначення мають заголовні файли?
4. Яка різниця між оголошенням і визначенням функції?
5. Для чого використовується директива #include?
6. Що таке захист від повторного включення заголовного файла?
7. Які файли зазвичай входять до складу багатофайлового проєкту?
8. Які переваги має багатофайлова організація програми?

Лабораторна робота №15. Робота з формами і графічним інтерфейсом

Тема. Розроблення програм із використанням форм і елементів графічного інтерфейсу користувача.

Мета: Набути практичних навичок створення віконних програм, роботи з формами, розміщення елементів керування на формі, оброблення подій користувача та організації простого графічного інтерфейсу.

Теми для попереднього опрацювання:

- змінні та типи даних;
- умовні оператори;
- цикли;
- функції;
- основи роботи в середовищі розробки Microsoft Visual Studio.

Загальні відомості

Графічний інтерфейс користувача – це сукупність візуальних елементів, за допомогою яких користувач взаємодіє з програмою. До таких елементів належать вікна, кнопки, текстові поля, написи, списки, прапорці, перемикачі та інші компоненти.

Основним елементом віконної програми є **форма**. Форма являє собою вікно, на якому розміщують елементи керування. Саме через форму

користувач вводить дані, запускає обчислення, отримує результати та керує роботою програми.

До основних елементів графічного інтерфейсу належать:

- **Label** — текстовий напис;
- **TextBox** — поле для введення тексту;
- **Button** — кнопка для виконання дії;
- **CheckBox** — прапорець для вибору параметра;
- **RadioButton** — перемикач для вибору одного з кількох варіантів;
- **ListBox** або **ComboBox** — список для вибору значень;
- **PictureBox** — область для відображення зображення;
- **Form** — головне вікно програми.

Кожний елемент графічного інтерфейсу має властивості, методи та події.

Властивості визначають зовнішній вигляд і стан елемента. Наприклад:

- Text — текст, що відображається;
- Name — ім’я елемента;
- Size — розмір;
- Location — розташування;
- Visible — видимість елемента;
- Enabled — доступність елемента для взаємодії.

Методи задають дії, які виконує елемент.

Події виникають у відповідь на дії користувача або зміну стану програми. Наприклад:

- натискання кнопки;
- зміна тексту в полі введення;
- вибір елемента зі списку;
- завантаження форми.

Однією з найпоширеніших є подія натискання кнопки. У програмі для неї створюють спеціальний обробник події, у якому записують код, що має виконуватися після натискання кнопки.

Використання графічного інтерфейсу дає змогу:

- зробити програму зручнішою для користувача;
- спростити введення та виведення даних;
- зробити результати роботи програми більш наочними;
- підвищити зручність керування програмою.

Основне завдання

Розробити віконну програму з графічним інтерфейсом користувача. Програма повинна містити форму та елементи керування, необхідні для введення даних, виконання обчислень або оброблення інформації, а також для виведення результату.

Під час виконання роботи необхідно:

- створити форму;
- розмістити на формі необхідні елементи керування;
- налаштувати їх основні властивості;
- реалізувати оброблення подій користувача;
- забезпечити коректне виведення результатів у графічному інтерфейсі.

Індивідуальне завдання

1. Розробити форму для перевірки коректності введення IPv4-адреси.
2. Розробити форму для визначення класу IPv4-адреси за першим октетом.
3. Розробити форму для підрахунку кількості символів, цифр і спеціальних знаків у введеному паролі.
4. Розробити форму для перевірки складності пароля за найпростішими ознаками.
5. Розробити форму для переведення обсягу даних з байтів у кілобайти, мегабайти і гігабайти.
6. Розробити форму для обчислення середнього часу відгуку за кількома введеними значеннями.
7. Розробити форму для визначення мінімального і максимального значень серед введених мережевих затримок.
8. Розробити форму для підрахунку кількості активних портів у списку значень 0 і 1, введених через пробіл.
9. Розробити форму для перевірки, чи належить введений номер порту до діапазону зарезервованих, зареєстрованих або динамічних портів.
10. Розробити форму для визначення кількості входжень символу «.» у введеному рядку з IP-адресою або доменним іменем.
11. Розробити форму для зміни регістру введеного імені хоста або тексту повідомлення.
12. Розробити форму для підрахунку кількості слів у введеному рядку, що містить опис мережевої події.

13. Розробити форму для пошуку найдовшого слова у введеному тексті журналу подій.
14. Розробити форму для перевірки, чи є введений рядок паліндромом.
15. Розробити форму для визначення кількості цифр у введеному ідентифікаторі пристрою або користувача.
16. Розробити форму для обчислення контрольної суми як суми цифр введеного числового коду.
17. Розробити форму для сортування введених числових значень за зростанням.
18. Розробити форму для обчислення суми, середнього арифметичного і добутку елементів масиву чисел, введених через пробіл.
19. Розробити форму для перевірки правильності введення електронної адреси за найпростішими ознаками.
20. Розробити форму для аналізу рядка, що містить запис події безпеки: визначити кількість літер, цифр, пробілів і спеціальних символів.

Додаткові вимоги виконання завдання:

- звіт має бути виконаний згідно з вимогами до оформлення робіт;

Контрольні запитання

1. Що таке графічний інтерфейс користувача?
2. Що таке форма у віконній програмі?
3. Які основні елементи керування використовують у графічному інтерфейсі?
4. Що таке властивості елемента керування?
5. Що таке подія в графічному інтерфейсі?
6. Для чого використовується обробник події?
7. Яке призначення мають елементи Label, TextBox і Button?
8. Як відрізняються прапорець і перемикач?
9. Як у програмі можна перевірити правильність введених даних?
10. Які переваги має графічний інтерфейс порівняно з консольним?

Список літератури

1. Stroustrup, B. *Programming: Principles and Practice Using C++*. 3rd ed. Addison-Wesley Professional, 2024.
2. Stroustrup, B. *A Tour of C++*. 3rd ed. Addison-Wesley Professional, 2022.
3. Davidson, J. Guy, Gregory, K. *Beautiful C++: 30 Core Guidelines for Writing Clean, Safe, and Fast Code*. Addison-Wesley Professional, 2021.
4. Roy, P. *C++ Memory Management: Write Leaner and Safer C++ Code Using Proven Memory-Management Techniques*. Packt, 2025. – 442 p.
5. Bolboaca, A., Deák, F.-L. *Debunking C++ Myths: Embark on an Insightful Journey to Uncover the Truths Behind Popular C++ Myths and Misconceptions*. Packt, 2024. – 226 p.
6. Іванов, Є.О., Ліндер, Я.М., Жереб, К.А. *Основи мови програмування С++: навчальний посібник*. Київ: Логос, 2020. 90 с.
7. Зеленський, О.С., Лисенко, В.С. *Основи програмування на С++: навчальний посібник*. – Кривий Ріг: ДУЕТ, 2023. – 269 с.
8. Grigoryan, V., Wu, S. *Expert C++: Become a Proficient Programmer by Learning Coding Best Practices with C++17 and C++20's Latest Features*. 2nd ed. Packt Publishing, 2020. – 606 p.
9. Бібліотека КНУБА. URL: http://library.knuba.edu.ua/.
10. Освітній сайт КНУБА. URL: http://org.knuba.edu.ua/.

Навчально-методичне видання

АЛГОРИТМІЗАЦІЯ ТА ПРОГРАМУВАННЯ

Методичні вказівки
до виконання лабораторних робіт 1–15
для здобувачів першого (бакалаврського) рівня вищої освіти
спеціальностей F7 «Комп'ютерна інженерія»
та F5 «Кібербезпека та захист інформації»

Укладачі: ПОПЛАВСЬКИЙ Олександр Анатолійович,
БОСЕНКО Ігор Валерійович
ПОРОХОВНІЧЕНКО Ірина Анатоліївна

Комп'ютерне верстання *А. П. Селівестрової*

Ум. друк. арк. 4,18. Обл.-вид. арк. 4,5
Електронний документ. Вид № 38/V-26.

Виконавець і виготовлювач

Київський національний університет будівництва і архітектури
Проспект Повітряних Сил, 31, Київ, Україна, 03037

Свідоцтво про внесення до Державного реєстру суб'єктів
видавничої справи ДК № 808 від 13.02.2002