**Лекція № 11. Програми лінійної, розгалуженої та циклічної структури.**

**На період** роботи в дистанційній формі навчання на надані в кінці запитання потрібно надати письмові відповіді в текстовому файлі (бажано WORD). Кожна відповідь оцінюється в 0,5 балів. Відповіді повинні бути не довгими і змістовними. Не копіюйте фрагментів з різних інформаційних джерел, подумайте і викладіть свою точку зору. При наявності відповідей-"близнюків" відповідь буде зараховуватися першому за часом надсилання.

Результати надсилати на електронну адресу викладача

**t.i.lumpova@gmail.com** у вигляді файлу з іменем у форматі

**<Номер групи> <Номер Лекції ><Прізвище >**

Наприклад, IPZ31-Лекція01Ivanov.docx.

Тему в заголовку листа записати

**ОП+АМ<Номер групи>- <Номер лекція>-<Прізвище>**

**Строк відсилки відповідей**

Всі запитання, що виникнуть, надсилайте на електронну адресу викладача, тему в заголовку листа записати

**Запитання-ОП+АМ <Номер групи>- <Прізвище >**.

На попередніх заняттях ми розглядали програми лінійної структури, які мали алгоритмічну структуру слідування. В цій лекції ми розглянемо реалізацію розгалуження та циклу.

**Інструкції розгалуження**.

Інструкції розгалуження реалізують оператори вибору, до яких відносять оператор умовного переходу **if** та оператор-перемикач **switch**.

Оператор умовного переходу **if** використовується для розгалуження процесу обчислень на два напрямки і має такий два форми запису: повна та неповна.

Повна формамає вигляд

**if (*умова*) *інструкція1;* else *інструкція2;***

Слова **if** та **else** є зарезервованими, дужки навколо умови обов'язкові. Цій формі оператора відповідає блок-схема на рис. 1 а). В операторі ***умова*** — це вираз, який має логічне значення (**true** — «істина» або **false** — «хиба, неправда»).

Реалізується оператор **if** в цій формі таким чином: спочатку обчислюється вираз ***умова***; якщо значення виразу не дорівнює нулю («істина»), виконується **інструкція 1**, в протилежному випадку — **інструкція 2** і далі управління передається оператору, що є наступним за умовним оператором **if**. Приклад:

**if ( i < j ) і++;**  
**else**  
**{ j = i-3; i++; }**

Неповна формамає вигляд

**if (*умова*) *інструкція1;***

В цій формі оператора **if** друга частина (тобто **else**) може бути відсутня і тоді, якщо вираз приймає значення **false** («хиба, неправда»), виконується зразу наступний оператор програми, що розташований за умовним. Таку конструкцію називають “пропуск оператора”. Цій формі оператора відповідає блок-схема на рис. 1 б). Приклад:

**if ( i >= j ) j = i-3;**

**i++;**

**if ((a>0)&&(b>0)&&(c>0)&&(a+b>c)&&(a+c>b)&&(b+c>a))**

**соut << "a,b,c –сторони трикутника"<<endl;**

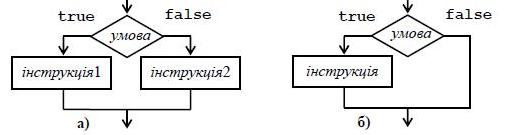


Рис.1 Блок-схеми двох форматів інструкції розгалуження

Коли у будь-якій гілці розгалуження необхідно виконати декілька операторів, їх слід розташовувати у блоці, інакше компілятор не зможе зрозуміти, де закінчується розгалуження. блок може включати різні оператори, у тому числі описи та інші умовні оператори, але не може складатися з одних описів. Потрібно ураховувати, що змінна, яка описана у такому блоці, за межами блока не існує. Синтаксис C++ припускає, що у випадку застосування вкладених умовних операторів кожне **else** відповідає найближчому до нього попередньому **if**. Наприклад:

**if (с > d** **&& (с > r** **|| с == 0)) d++;**  
**else**  
**{int b=10; d** **/= с; с = 0 }**

тут декілька умов слід об’єднати знаками логічних операцій, а сукупність операцій необхідно розмістити у блоці.

Обчислення найбільшого значення серед трьох змінних **а, b, с**:

**if (а > b)  
{ if (а > с) max = а;**  
 **else max = с; }**  
**else { if (b > c) max = b;**  
 **else max = c; }**

У цьому фрагменті програми фігурні дужки можуть бути відсутні, тому що компілятор відносить частину **else** до найближчого **if**;

**if(p++) ps++;** — вираз не використовує операцій відношення.

**Оператор-перемикач** **switch** називають оператором множинного розгалуження. Він використовується для вибору одного а багатьох варіантів рішення і має таку форму запису:

**switch (L)**

**{**

**case к.в.1: *інструкція 1*; [break;]**

**case к.в.2: *інструкція 2*; [break;]**

**..................................................**

**case к.в.n: *інструкція* n; [break;]**

**[default: *інструкція* n+1;]**

**}**

де **switch, case, default** — службові слова;

**break** — оператор (необов’язковий) здійснює вихід з оператopa **switch**;  
**L** — будь-який вираз одного з цілих типів; цей вираз називається **селектором варіантів**.

**к.в.1, …, к.в.n** — константні вирази, які не можуть повтoрюватися і не можуть містити змінних чи викликів функцій; зазвичай, це ціла або символьна константа. Ці значеннями називаються **мітками варіантів**;

**оператор 1; …** – будь-які оператори мови C++.

У процесі виконання цього оператора спочатку обчислюється значення виразу **L**, потім це значення порівнюється (послідовно зверху донизу) зі значеннями константних виразів **к.в.1, …, к.в.n.** У випадку збігу значень **L** і одного з цих константних виразів та, якщо наприкінці гілки розгалуження (варіанту) немає оператора **break**, виконуються всі оператори, починаючи з відповідної гілки. За наявності оператора **break** виконується тільки оператор, що знаходиться у відповідній гілці розгалуження, і керування передається оператору, який розташований за межами oпeратору **switch**. Якщо значення виразу **L** не збігається з жодним із значень константних виразів, то виконується гілка **default** і здійснюється вихід з оператора **switch**. У випадку, коли в операторі **switch** відсутня гілка **default** (вона необов’язкова) і значення **L** не збігається з жодним із значень константних виразів, відбувається вихід з оператора **switch**. Варіант / гілка з міткою **default** можна записати будь-де, але рекомендується записувати його останнім.

Приклад використанням оператора **switch**:

**int а=2;**

**switch (а)**

**{**

**case 1: func1( );**

**case 2: func2( );**

**case 0: …**

**case 4: func3( );**

**default: printf ("good bye \n");**

**}**

Тут оператор **switch** передбачає реалізацію функцій **func2(), func3()** і **default: printf (“good bye \n”);.**

Фрагмент програми можна записати по-іншому:

**int а=2;**

**switch (а)**

**{**

**case 1: funcl( ); break;**

**case 2: func2( ); break;**

**case 0: ...**

**case 4: func4( ); break;**

**default: printf ("good bye \n");**

**}**

Оператор-перемикач приводить до виконання тільки гілки

**case 2: func2( ); break;** після чого здійснюється вихід із **switch**.

Щоб написати кілька інструкцій там, де за правилами мови має бути одна, наприклад, як гілку в умовній інструкції, використовують **блок** – послідовність інструкцій у дужках **{}**. Він має такий загальний вигляд:

**{**

***інструкція***

**...**

***інструкція***

**}**

Виконання блоку полягає в послідовному виконанні інструкцій, записаних у ньому.

**Прості інструкції повторення обчислень**

Циклічні, тобто повторювані, обчислення задають за допомогою оператору циклу. Який має форми:

Інструкція циклу з передумовою **while**,

Інструкція циклу з післяумовою **do while**,

Інструкція циклу **for**.

Оператори циклу використовують для здійснення багаторазового повторення деякої послідовності дій. Кожен цикл складається з тіла циклу, тобто операторів, що виконуються декілька разів. Один прохід циклу називається ітерацією.

Інструкція циклу з передумовою **while** виконується, якщо умова перевіряється до початку циклу, і має вигляд:

**while (умова) *інструкція;***

Слово **while** є зарезервованим, дужки обов'язкові, **while (*умова*)** – це **заголовок циклу**, а ***інструкція*** – **тіло**.

Інструкція циклу виконується так. Спочатку обчислюється умова в заголовку. Якщо вона істинна, то виконується тіло циклу та знов обчислюється умова. Якщо вона істинна, то все повторюється. Виконання інструкції циклу закінчується, коли обчислено значення умови **false**, тобто хибність. Отже, в останньому циклі тільки обчислюється умова, а тіло не виконується. Якщо при першому обчисленні умова хибна, то тіло циклу не виконується жодного разу. **Ітерація циклу** складається з перевірки умови циклу та виконання після неї тіла циклу. Інструкції циклу з передумовою відповідає блок-схема на рис. 2.

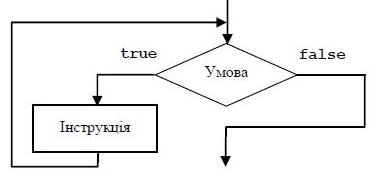


Рис. 2. Блок-схема інструкції циклу з передумовою

Умову в інструкції циклу називають **умовою продовження**, оскільки, якщо вона істинна, то виконання інструкції циклу продовжується. Цикл починається обчисленням умови, тому її ще називають **передумовою**. Інструкції циклу з передумовою застосовують зазвичай тоді, коли кількість повторень циклу наперед невідома.

Приклад: наступна послідовність операторів обчислює суму квадратів перших **n** натуральних чисел:

**int і=0, sum=0; n=100;**  
**while (і < n) sum += ++і \* і;**

Інструкція циклу з **післяумовою**, або **do**-інструкція, має загальний вигляд

**do *інструкція* while (умова);**

**Оператор циклу з післяумовою do while** звичайно застосовується у випадках, коли тіло циклу виконується хоча б один раз. Слово **do** (виконувати) є ключовим. Інструкція циклу з післяумовою виконується так: спочатку виконується тіло циклу, потім обчислюється умова. Якщо вона хибна, то цикл завершується, інакше повторюється тіло й знову обчислюється умова. На відміну від інструкції циклу з передумовою, цикл *починається діями в тілі циклу* та закінчується обчисленням умови. Умова перевіряється після виконання тіла циклу, тому її називають **післяумовою**. Тіло циклу, заданого **do**-інструкцією, виконується обов'язково хоча б один раз (на відміну від **while**-інструкції). Інструкцію циклу з післяумовою використовують, коли потрібно спочатку виконати тіло циклу, і лише потім перевіряти умову продовження. Циклу з післяумовою відповідає блок-схема на рис. 3.

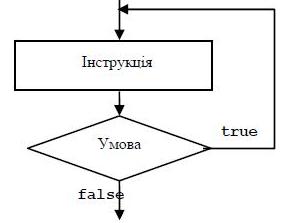


Рис. 3. Блок-схема інструкції циклу з післяумовою

**Приклад.** Потрібно з клавіатури ввести ціле число від 10 до 99. Якщо користувач набрав число за межами цього діапазону, то слід *повторити спробу*. Отже, спочатку треба вводити число, а потім перевіряти умову того, що число є двозначним.

**do {**

**cout << "Enter one integer in [10,99]>";**

**cin >> k;**

**} while (!(10<=k && k<=99)); // 10<=k && k<=99**

Інструкція циклу **for** або **for**-інструкція, має загальний вигляд

**for (*початкова дія*; *умова*; *перехідна дія*)**

***основна дія***

Слово **for** зарезервоване, дужки та два знаки; усередині дужок є обов'язковими. Початкова дія, умова й перехідна дія є *виразами* (кожен із них може бути порожнім), основна дія – *інструкцією*. Тілом циклу **for** називають його основну дію. Інструкція **for** виконується так само, як і інструкції вигляду

***початкова дія*;**

**while (*умова*)**

**{**

***основна дія*;**

***перехідна дія*;**

**}**

Оператор циклу **for** реалізується таким чином:

* виконується початкова дія - вираз ініціювання (виконання цієї нотації може бути здійснено до оператора **for**);
* обчислюється вираз-умова;
* якщо умовний вираз приймає значення «істина» — виконуються оператори циклу;
* обчислюється вираз ітерації;
* знову перевіряється умова;
* як тільки умова прийме значення **0** («хиба, неправда»), керування передається оператору, що розташований за оператором циклу **for**.

Оскільки в операторі **for** перевірка виразу-умови відбувається перед циклом, то у випадку помилкової умови цикл може жодного разу не виконуватися.

Важливі зауваження

Якщо блок умова залишити порожнім, тоді результат аналізу умови завжди буде вважатися true і ми отримаємо "безконечний цикл".

Змінні, визначені всередині циклу for, мають спеціальний тип області видимості, область видимості циклу.

Такі змінні існують лише всередині циклу та недоступні за його межами.

Подання циклу на блок-схемі надано на рис.4.

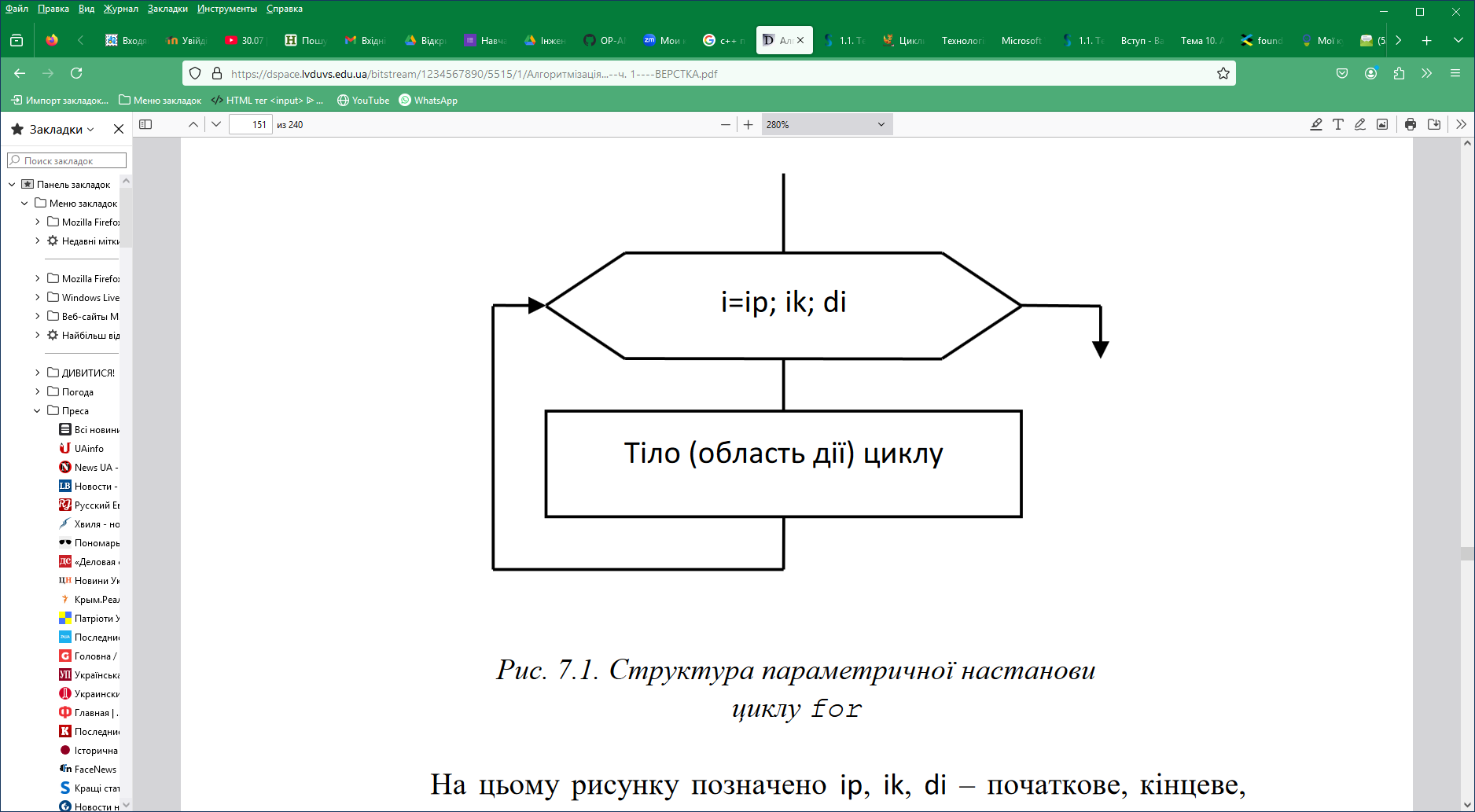


Рисунок 4 – Подання циклу на блок-схемі

Оператор **for** може використовувати декілька змінних, що керують циклом, а будь-які вирази можуть бути відсутніми, наприклад:

**int n, у;**

**for (int k = 0, n = 20;k <= n; k++, n--)**

**y = k \* n;**

**cout<<k;//не працює**

або

**int і=0;**

**for (; і < 4; i++)**

Перший фрагмент має два вирази ініціювання і два вирази ітерації. Спочатку відбувається присвоювання значень змінним **k = 0 і** **n = 20**, далі здійснюється порівняння **k <= n** і, якщо ця умова має значення «істина», то буде виконуватися тіло циклу, а потім вираз **k++ і** **n–-**, якщо ж умова не виконується, то цикл припиняє свою роботу.

C++ дозволяє поєднати ці дві дії в одному виразі – за допомогою операції послідовного обчислення. Операція зі знаком "**,**" позначає послідовне обчислення виразів, записаних через кому (в прикладі це **к = 0, n = 20;)**. Ця послідовність виразів розглядається як один вираз; його значенням є значення останнього виразу. Операція послідовного обчислення дозволяє на місці одного виразу записати кілька.

Операторам циклів с параметром **for** потрібно віддати перевагу при організації циклів з лічильниками.

В циклі **for** можна використовувати інструкції **break** та **continue.** Інструкція **break** у тілі циклу **for** завершує його виконання, а інструкція **continue** завершує виконання лише тіла циклу; відразу після неї виконується перехідна дія.

**Приклад.** Дуже часто інструкція циклу **for** зустрічається у вигляді

**for (k=0; k<n; ++k) *інструкція***

й задає виконання *інструкції* за значень *k* = 0, 1, 2, …, *n*-1 або у вигляді

**for (k=1; k<=n; ++k) *інструкція***

й задає виконання *інструкції* за значень *k* = 1, 2, …, *n*, або у вигляді

**for (k=n; k>0; --k) *інструкція***

й задає виконання *інструкції* за значень *k* = *n*, *n*-1, …, 2, 1. Змінну **k** у цих ситуаціях інколи називають **лічильником циклу**.

**Переривання break та продовження циклу continue**

Виконання інструкції **break** всередині циклу будь-якого різновиду перериває й завершує цикл; далі виконуються дії, наступні за цим циклом. Якщо **break** записано в інструкції циклу, вкладеній в іншу інструкцію циклу, то виконання **break** завершує вкладений цикл, а зовнішній цикл продовжується.

Інструкція **continue** всередині циклу задає перехід на кінець тіла циклу. В інструкціях циклу з перед- і післяумовою після **continue** обчислюється умова продовження циклу.

**Приклад.** За допомогою клавіатури вводиться послідовність дійсних чисел. Потрібно підрахувати суму її додатних елементів, а за появи 0 видати накопичену суму й завершити роботу.

Запрограмуємо цикл, в якому вводиться й обробляється послідовність чисел. Уведене число зберігаємо в змінній **x**, а суму додатних елементів – у змінній **sum**. Якщо під час уведення трапилася помилка, то подальші дії з уведення не виконуються, а змінна **x** зберігає своє останнє значення. Тому умовою продовження циклу буде саме відсутність помилок (інакше можна отримати цикл, який ніколи не завершиться!). Цю умову задає значення виразу введення **cin>>x**, перетворене до логічного типу.

**#include <iostream>**

**using namespace std;**

**int main()**

**{ double x;**

**double sum=0;**

**cout<<"Enter reals:\n";**

**while (cin>>x){**

**if (x==0) break; //виходимо з циклу**

**if (x<0) continue; //пропускаємо від'ємні**

**sum+=x;**

**}**

**cout << "sum=" << sum << endl;**

**system("pause");**

**return 0;**

**}**

Використання інструкції **continue** в цій програмі є дуже штучним. Ще одним недоліком є те, що в кінці не повідомляється, чи були помилки під час уведення. Інструкції програми виконуються *в порядку їх запису в програмі*. Про таку програму кажуть, що вона **структурована**. Інструкції **break** і **continue** *порушують* цей порядок обчислень, заплутуючи текст програми. Тому, користуючися ними, програміст повинен ретельно відслідковувати точку програми, якою продовжуються обчислення. Інколи ці інструкції дійсно скорочують запис розгалужень у циклі, проте в більшості випадків ті ж самі дії *можна описати без них*. Тому краще не зловживати **break** і **continue**.

**Приклади використання циклів у програмуванні**

Цикли в програмуванні використовуються для автоматизації повторюваних дій.

1. **Обробка даних у великих об’ємах.**  Приміром, цикли використовуються в програмному забезпеченні для обробки великих об’ємів даних, наприклад, обробки транзакцій у банківських системах.
2. **Робота з масивами.** Цикли часто використовуються для обробки елементів масивів, наприклад, для підсумовування значень, знаходження мінімального або максимального значення, сортування елементів тощо.
3. **Ітерації за елементами користувацького інтерфейсу.** Цикли можуть використовуватися для обробки елементів користувацького інтерфейсу, таких як таблиці, списки, дерева тощо. Наприклад, для обробки всіх елементів таблиці або списку, додавання нових елементів або видалення наявних.
4. **Автоматизація процесів.** Цикли використовуються для автоматизації повторюваних процесів, таких як створення звітів, обробка файлів тощо. Наприклад, цикл може використовуватися для обробки всіх файлів у директорії, щоб виконати певну дію для кожного файлу.

**Контроль циклів у програмуванні**

Важливо правильно вибирати тип циклу і контролювати виконання, щоб уникнути нескінченного виконання та інших проблем.

Один зі способів контролю циклів – використання умовних операторів, таких як if, які перевіряють умови виходу з циклу. Наприклад, можна використовувати умову виходу з циклу при досягненні певного значення або при виконанні умови.

Також можна використовувати лічильник, щоб обмежити кількість ітерацій циклу. Лічильник збільшується на кожній ітерації, і коли він досягає певного значення, цикл завершується.

Ще один спосіб контролю циклів – це використання винятків. Якщо відбувається помилка під час виконання циклу, можна використати виняток для завершення циклу.

Загалом, контроль циклів – це частина розробки програмного коду, яку потрібно враховувати під час написання будь-якого циклу в програмі. Але важливо уникати нескінченного виконання циклів, щоб уникнути збоїв програми та витоку ресурсів.

**Оптимізація циклів**

***Оптимізація циклів*** – важлива частина оптимізації програмного коду загалом, оскільки цикли можуть споживати багато ресурсів і сповільнювати виконання програми. Ось деякі методи оптимізації циклів:

1. Зменшення кількості ітерацій. Якщо ви знаєте заздалегідь, скільки разів потрібно виконати цикл, можна використовувати цикл for, щоб зменшити кількість ітерацій.
2. Використання більш ефективних операцій. Під час роботи з масивами даних краще використовувати спеціальні операції, методи, оптимізовані для пошуку, додавання або видалення елементів.
3. Уникнення зайвих обчислень. Можна уникнути повторних обчислень, зберігаючи результати і використовуючи їх знову, а не перераховувати щоразу.
4. Робота з пам’яттю. Можна зменшити кількість операцій з пам’яттю шляхом використання більш ефективних алгоритмів і структур даних, які мінімізують кількість операцій запису та читання з пам’яті.
5. Використання паралелізму. Можна використовувати паралельні обчислення, щоб виконати цикли на кількох ядрах процесора одночасно, що значно прискорить виконання програми.

**Оптимізація циклів дає змогу підвищити продуктивність програми та зменшити споживання ресурс**

**Збільшення та зменшення**

У циклічних обчисленнях дуже часто використовуються присвоювання вигляду **x=x+1** та **x=x-1**. Їх можна задати в скороченій формі за допомогою одномісних операторів **збільшення** (інкременту) **++** і **зменшення** (декременту)º**--**. Ці оператори (і відповідні операції) мають **префіксну (++x, --x**) і **постфіксну** (**x++, x--**) **форми**.

Вираз із постфіксним оператором **x++** або **x--** змінює значення змінної **x** на **1**, але значенням самого виразу є значення **x** *перед зміною*. Вираз із префіксним оператором **++x** або **--x** теж змінює **x** на **1**, але значенням виразу є значення **x**, отримане *після зміни*. Ці відмінності виявляються, коли оператори **++** та **--** застосовуються всередині виразів.

Операції **++** та **--** виконуються швидше ніж відповідні присвоювання вигляду **x=x+1** та **x=x-1**, тому рекомендується використовувати саме їх. Операції **++** та **--** застосовні до змінних будь-якого з базових типів, хоча найчастіше їх використовують із цілими змінними.

Скрізь, де немає необхідності використовувати старе значення змінної, рекомендується з виразів вигляду **n++** та **++n** вибирати **++n**, оскільки він виконується швидше й простіше.

Спосіб і порядок обчислення виразу залежить від компілятора, тому краще записувати операції збільшення або зменшення в окремих виразах або інструкціях, а не у складі інших виразів. Наприклад, значення виразів **(n++)\*(n++)** та **(++n)\*(++n)** у різних системах програмування навіть можуть відрізнятися. Гарантовано лише те, що до значення змінної **n** двічі додається 1.

**Принципи структурного програмування**

Структурне програмування це методологія й технологія розробки програмних комплексів, заснована на наступних принципах:

*- програмування* повинне здійснюватися зверху-*униз;*

- увесь проект повинен бути розбитий на *модулі* з одним *входом* і одним *виходом* (оптимальний розмір модуля — кількість *рядків* на екрані *дисплея);*

- логіка *алгоритму* й *програми* повинна допускати тільки три основні структури: *послідовне виконання, розгалуження* й *повторення.* Неприпустимий *оператор* передачі керування в будь-яке місце *програми;*

- при розробці *документація* повинна створюватися одночасно із *програмуванням,* у вигляді коментарів до програми.

Ціль структурного програмування — підвищення надійності програм, забезпечення *супроводу* й модифікації, полегшення й прискорення розробки. У програмах з використанням структурного програмування добре простежується основний алгоритм, вони більш зручні в налагодженні і менш чутливі до помилок програмування. Ідеї структурного програмування з'явилися на початку 70-х рр. XX в. у компанії *IBM,* у їхній розробці брали участь відомі вчені Є. Дейкстра, X. Милі, Є. Батіг, С. Хоор.

Існує декілька важливих моментів структурного програмування:

1. Вихідний код повинен мати модульну структуру. Тобто, програма розділяється на дрібніші одиниці – процедури і функції. Ці частини або підпрограми, можуть викликатися з будь-якого місця у ній. Процедури – окремі ділянки коду, які виконують певні дії, задані алгоритмом та мають власну назву. Функції також можуть обчислювати деякі змінні,  мають значення, яке повертається, і можуть використовуватись в основній частині програми і в інших підпрограмах. Деякі підпрограми можуть мати рекурсивну структуру, тобто виклик з «самої себе». Це може допомогти вирішити задачу, але і призвести до зациклювання.
2. Кодування програми повинно виконуватися зверху-вниз чи знизу вгору. Схема «зверху-вниз» добре зрозуміла для дослідження написаної програми і пошуку помилок. Схема «знизу-вгору» використовується, коли алгоритм програми не розроблений, але вже написані деякі підпрограми, які реалізують певні дії.
3. Наявність керуючих елементів. У структурному підході використовуються цикли, умови і послідовності.

Структурне програмування поліпшило загальне сприйняття коду та сприяло легшому написанню програм. За методами структурного програмування, алгоритм програми стає універсальним і за рахунок цього будь-який інший розробник зможе його змінити або використати у своїй програмі.

*Для самостійного вивчення (2 години)*: Вивчення лекційного матеріалу та додаткових джерел. Розгляд запитань і виконання завдань для самостійної роботи, запропонованих на лекції.

*Література*

1. Ковалюк Т. В. Алгоритмізація та програмування: Підручник. — Львів: «Магнолія 2006», 2013. — 400 с., ил.
2. Єжова Л. Ф. Алгоритмізація і програмування процедур обробки інформації: Навч.-метод. посіб. для самост. вивч. дисц. К.: КНЕУ, 2000.
3. Вступ до програмування мовою С++. Організація обчислень: навч. посіб. / Ю. А. Бєлов, Т. О. Карнаух, Ю. В. Коваль, А. Б. Ставовський. – К. : Видавничо-поліграфічний центр "Київський університет", 2012. – 175 с. с.: іл. ISBN (укр.) . URL: <http://csc.knu.ua/uk/library/books/belov-24.pdf>
4. Джейс Либерти Освой самостоятельно С++ за 21 день: 3-е изд. пер. с англ.: Уч. пос. – М.: Издательский дом „Вильямс”, 2001. – 816 с.: ил..

*Контрольні запитання для самоперевірки*.

1. Які оператори реалізують розгалуження у програмі?
2. Як діє умовний оператор if?
3. Як працює оператор-перемикач switch?
4. Які оператори циклу використовуються у C++?
5. Як працює оператор циклу for?
6. Пояснити на прикладах використання циклу з передумовою і циклу з післяумовою.
7. Які оператори з розглянутих є операторами керування у мові C++?

*Контрольні запитання для надання письмових відповідей*.

1. Наведіть власні приклади умовного оператору **if** у двох формах.
2. Наведіть власний приклад оператор-перемикача **switch**.
3. Наведіть власні приклади всіх форм оператору циклу.
4. Наведіть власні приклади використання операторів переривання **break** та продовження циклу **continue.**

**До кожного прикладу надайте короткий опис завдання щодо виконання оператору.**