**Тема 2. Загальні поняття. Елементи мови С++ - константи, змінні, операції, перетворення типів.**

**Постановка завдання**

В Постановці задачі потрібно розкрити такі аспекти:

* Мета розробки ПЗ, призначення ПЗ
* Вхідні дані.
* Вихідні дані, очікувані результати.
* Функції обробки інформації (коротка технологічна схема за принципом введення інформації → контроль інформації → оброблення → формування результатів ), які відображують через які процеси вхідні дані трансформуються у вихідні.

Вхідні дані повинні бути чітко визначені. Доцільно подавати їх у вигляді такої таблиці:

|  |  |  |  |
| --- | --- | --- | --- |
| №№ | Назва атрибута | Формат | Обмеження |
| 1 | 2 | 3 | 4 |
|  |  |  |  |

Порядок визначення формату та обмежень залежить від встановлених в організації – розробника стандартів. Наприклад, можна надавати визначення відповідно обраної мови програмування або виразами "текст" чи "число", або визначенням формату таким як "ХХХ" - ціле, "ХХ.ХХ" –десяткове з 2 знаками після коми, "ДД.ММ.РРРР" – дата.

За аналогічною схемою визначаються і вихідні дані.

Функції обробки інформації зазвичай описуються словесно та схематично. Прикладом такої схеми може бути блок-схема.

**Алгоритм рішення**

Алгоритм - сукупність правил, однозначно визначующих процес перетворення вхідних і проміжних даних у результат рішення задачі. Опис алгоритму являє собою загальну схему рішення задачі. Алгоритмічний процес це - процес послідовного перетворення конструктивних об'єктів, що проходить дискретними кроками (тобто зміна відбувається стрибкоподібно), кожний крок полягає в зміні одного конструктивного об'єкта іншим. Алгоритми характеризуються обчислювальною складністю і ємкісною складністю. За видом використовуваної обчислювальної моделі алгоритми діляться на послідовні (або детерміновані), паралельні (або недетерміновані), розподілені та ін.

Алгоритм може бути реалізований в обчислювальних системах, якщо він містить тільки елементарні команди, тобто не потребуючими деталізації, можна вважати такі команди або операції:

1) початок, кінець;

2) список даних;

3) введення, виведення;

4) обчислювальні операції, реалізовані оператором присвоювання

Для алгоритму характерні наступні властивості:

- детермінірованість, чи визначеність, тобто однозначність –його розуміння для будь-якого виконавця, що приводить до точного виконання однієї і тієї ж послідовності дій;

- результативність, чи спрямованість , тобто властивість досягнення за кінцеве число досить простих кроків шуканого результату розглянутої задачі;

- масовість, тобто придатність для рішення будь-якої задачі з деякого класу задач.

Розрізняють наступні способи представлення алгоритмів: текстуальний,

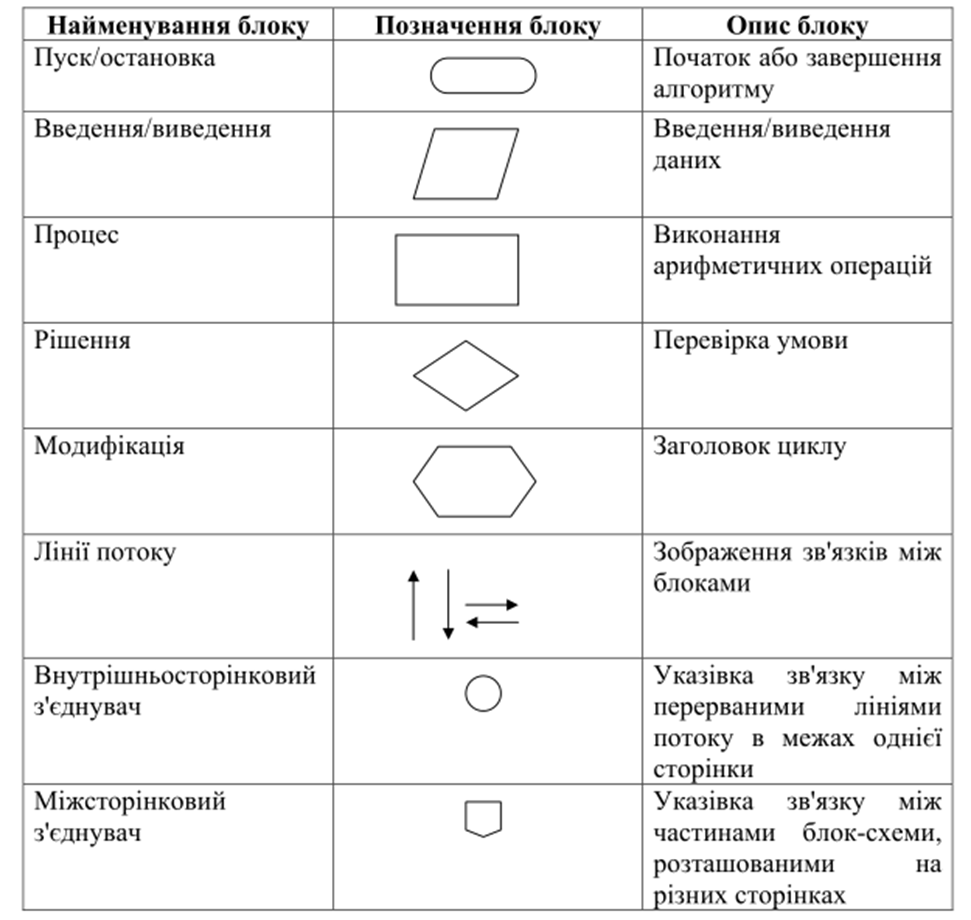
операторний і графічний.

Найбільше поширення в даний час одержав графічний спосіб, при якому обчислювальний процес розчленовується на окремі операції, що відображаються у виді умовних графічних символів (блоків). Серед графічних способів найпростішим є блок-схема.

**Побудова блок- схем**

Для побудови блок-схеми використовуються графічні символи представлені в табл.1. Блоки з'єднуються між собою в визначеної послідовності лініями чи стрілками. Усередині блоків у виді формул чи тексту вказується інформація, що пояснює, характеризує виконувані ними дії. Блоки звичайно мають наскрізну нумерацію. Номер ставиться у верхньому лівому куті блоку в розриві. ліній.

Таблиця 1. Символи для створення блок-схеми



Блоки з'єднуються між собою в визначеної послідовності лініями чи стрілками. Усередині блоків у виді формул чи тексту вказується інформація, що пояснює, характеризує виконувані ними дії. Блоки звичайно мають наскрізну нумерацію. Номер ставиться у верхньому лівому куті блоку в розриві ліній.

Теорія структурного програмування доводить, що алгоритм будь-якого ступеня складності можна побудувати за допомогою основного базового набору структур:

1) послідовна (лінійна) структура (рис.1);

2) структура, що розгалужується (рис. 2);

3) циклічна структура (рис. 3).

Найпростішими для розуміння і використання є лінійні структури (рис. 1). Лінійним називається алгоритм (фрагмент алгоритму), у якому окремі розпорядження виконуються незалежно від значень вихідних даних і проміжних результатів

**Алгоритм лінійної структури**

Лінійним називається алгоритм (фрагмент алгоритму), в якому окремі команди виконуються послідовно друг за другом, не залежно від значень вхідних даних і проміжних результатів. Приклад алгоритму лінійної структури (рис. 1).

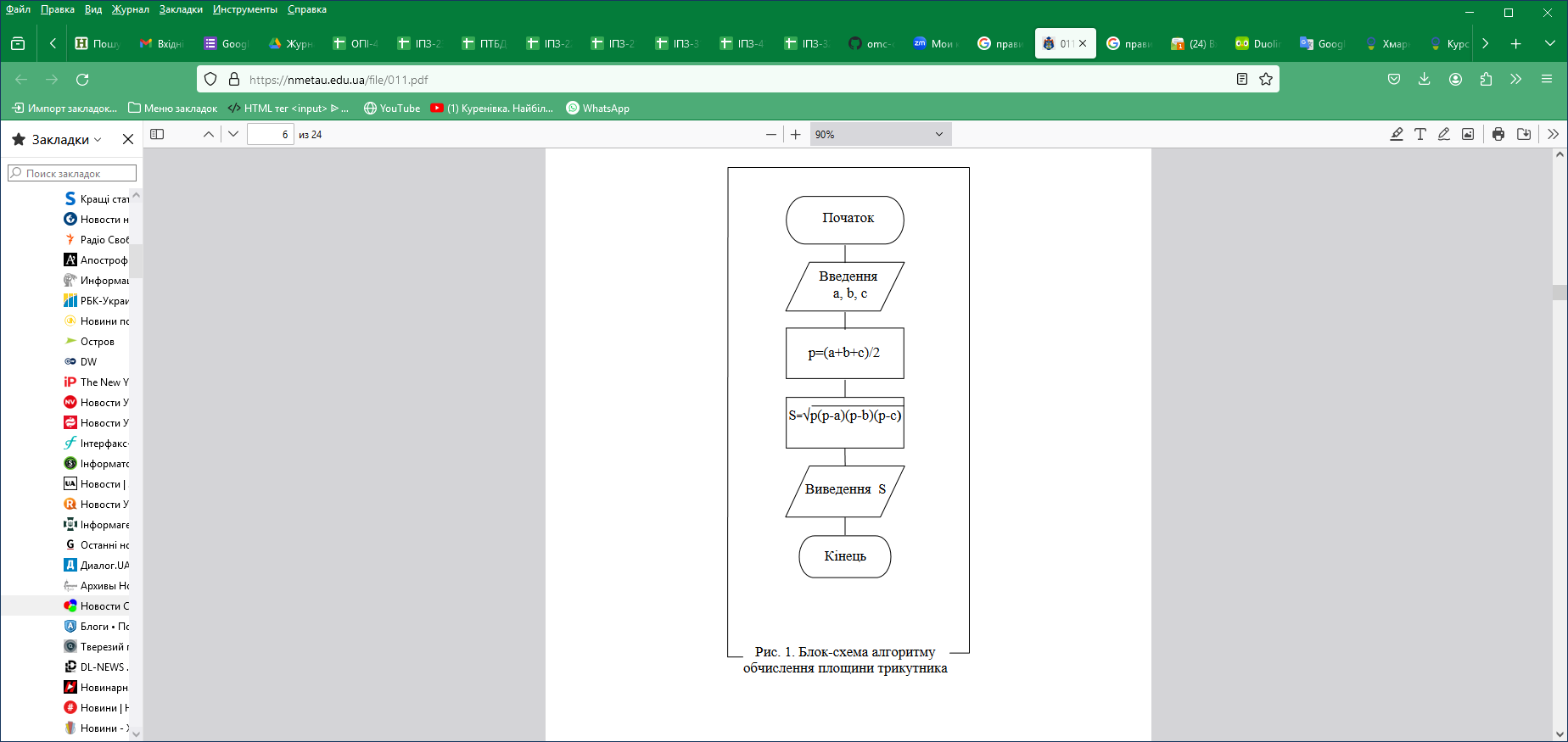


Рисунок 1 - Блок-схема лінійного алгоритму

**Блок-схеми процесів, що розгалужуються**

Часто для подальшої деталізації використовуються структури, що розгалужуються, тобто такі, в яких у залежності від вихідних даних або проміжних результатів алгоритм реалізується в одному з декількох, заздалегідь передбачених (можливих) напрямків. Такі напрямки часто називаються гілками. Кожна гілка може бути будь-якого ступеня складності, а може взагалі не містити команд, тобто бути виродженою. Вибір тієї або іншої гілки здійснюється в залежності від результату перевірки умови з конкретними даними. У кожному випадку алгоритм реалізується тільки по одній гілці, а виконання інших виключається. Приклад алгоритму, що розгалужується (рис. 2).

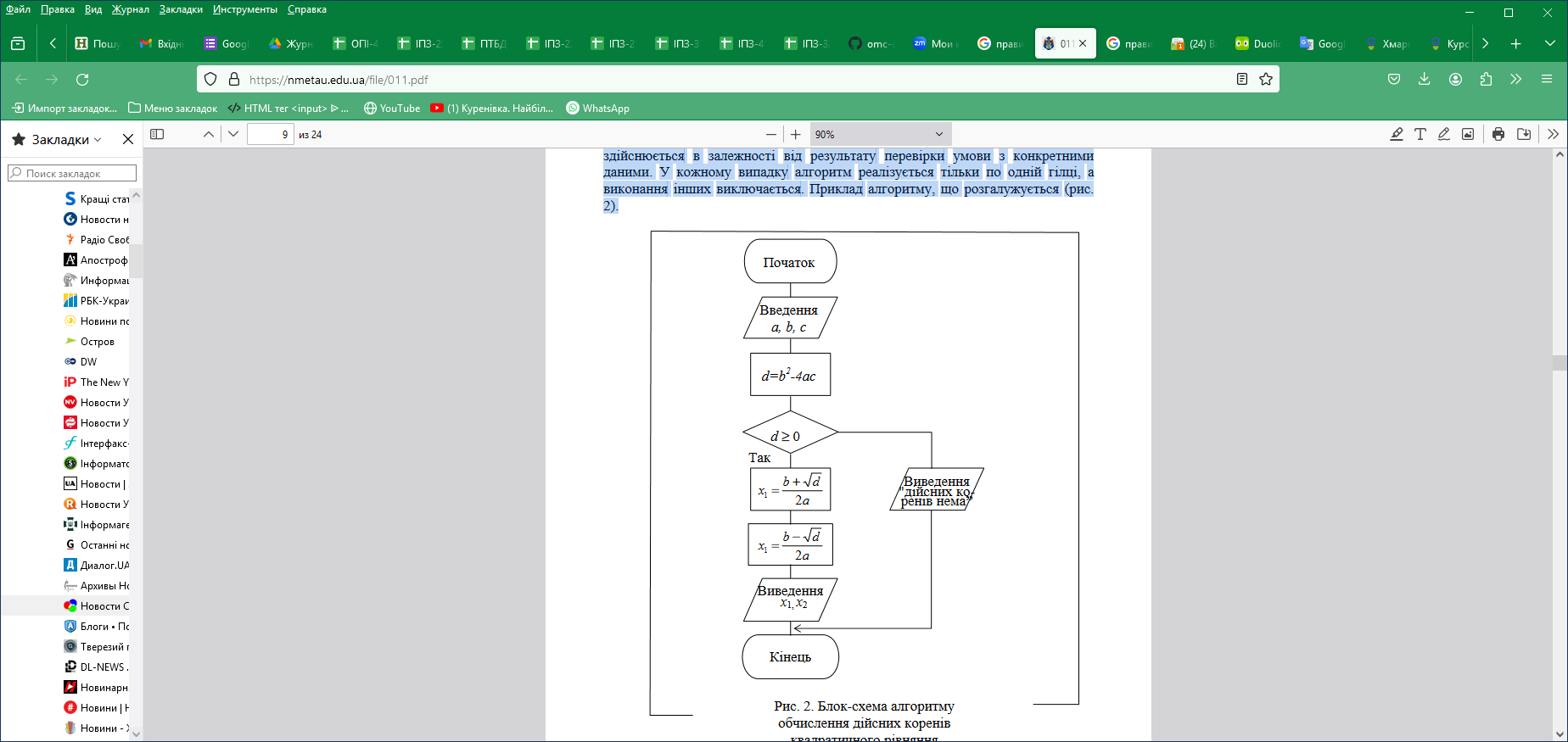


Рисунок 2 - Блок-схема алгоритму обчислення дійсних коренів квадратичного рівняння

**Блок-схеми циклічних обчислювальних процесів**

Виконання деякої частини алгоритму багаторазово при різних значеннях деяких змінних називається циклічним обчислювальним процесом. Циклами називаються повторювані ділянки обчислень. Для організації циклів необхідно: задати початкове значення змінної, що визначає цикл (параметр циклу), зміняти цю змінну перед кожним повторенням циклу, перевіряти умову продовження (закінчення) циклу.

У циклічних алгоритмах виконання деяких операторів (груп операторів) здійснюється багаторазово з тими ж або модифікованими даними У залежності від способу організації кількості повторень циклу розрізняють три типи циклів:

1) цикл із заданою умовою закінчення роботи (ЦИКЛ – ДО / DO - WHILE);

2) цикл із заданою умовою продовження роботи (ЦИКЛ – ПОКИ / WHILE);

3) цикл із заданою умовою повторень роботи (ЦИКЛ 3 ПАРАМЕТРОМ / FOR).

Приклади циклічного алгоритму (рис. 3- 8).

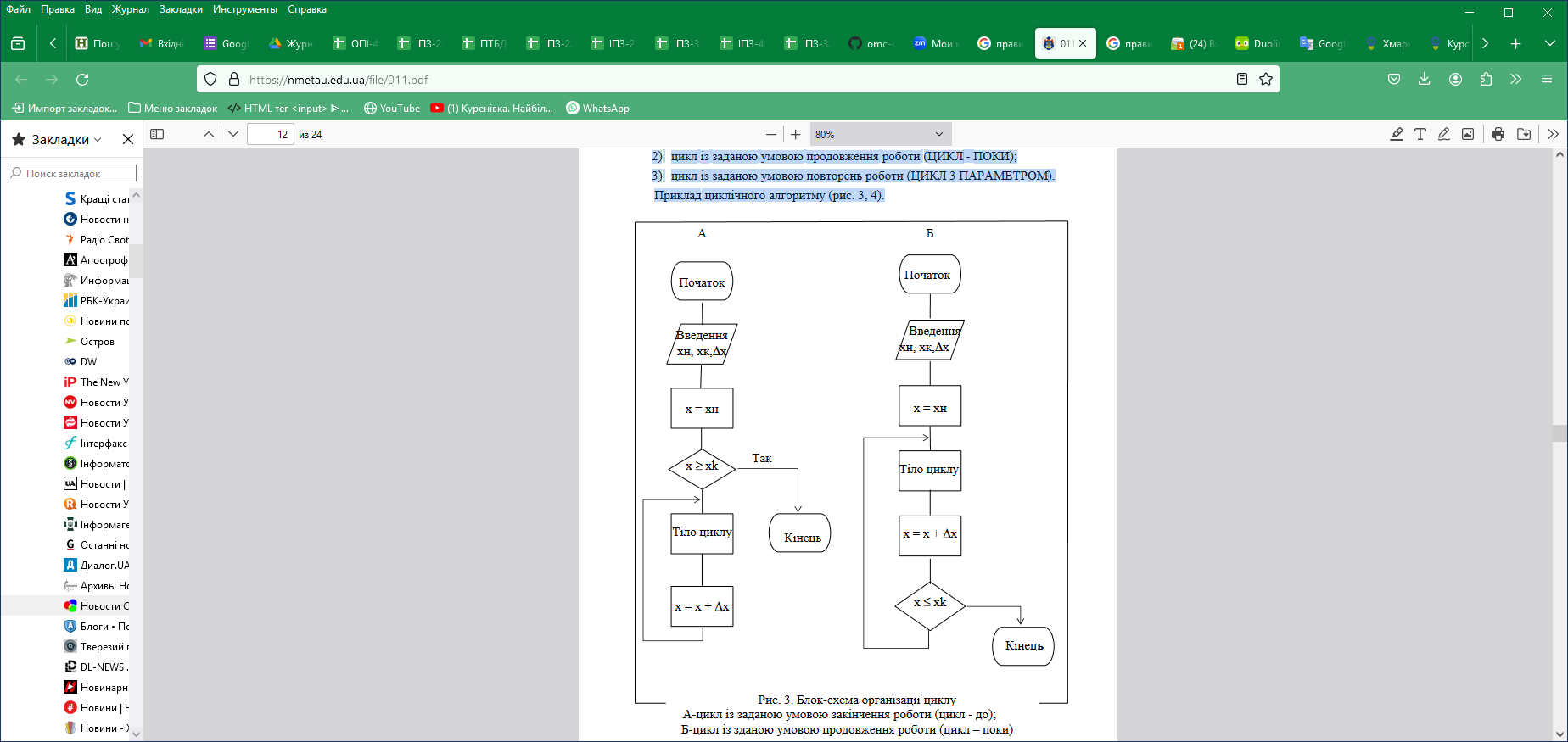


Рисунок 3 - Блок-схема організаціі циклу

А-цикл із заданою умовою закінчення роботи (цикл – до/DO);

Б-цикл із зданою умовою продовження роботи (цикл – поки/WHILE)

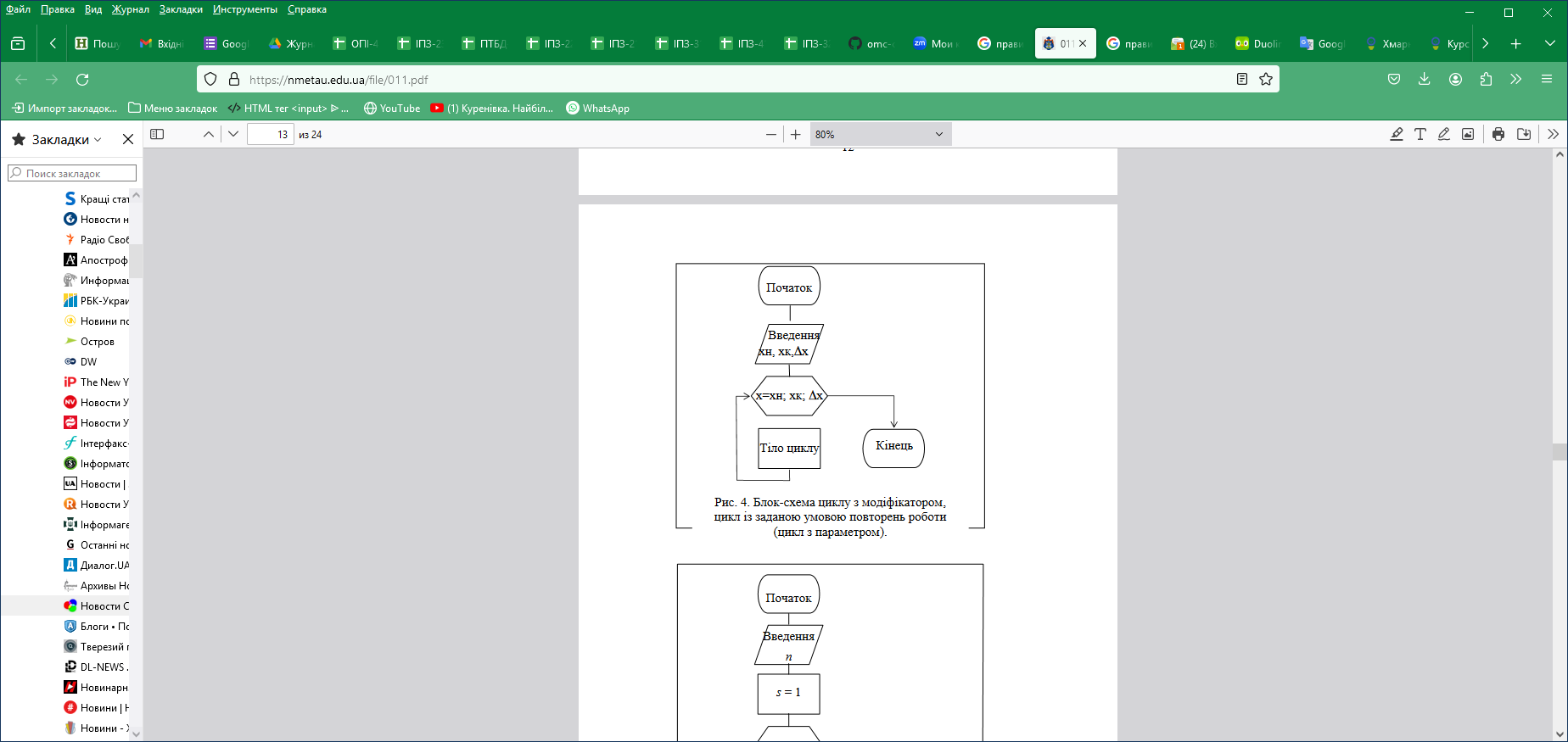


Рисунок 4 - Блок-схема циклу з модіфікатором, цикл із заданою умовою повторень роботи (цикл з параметром / FOR).

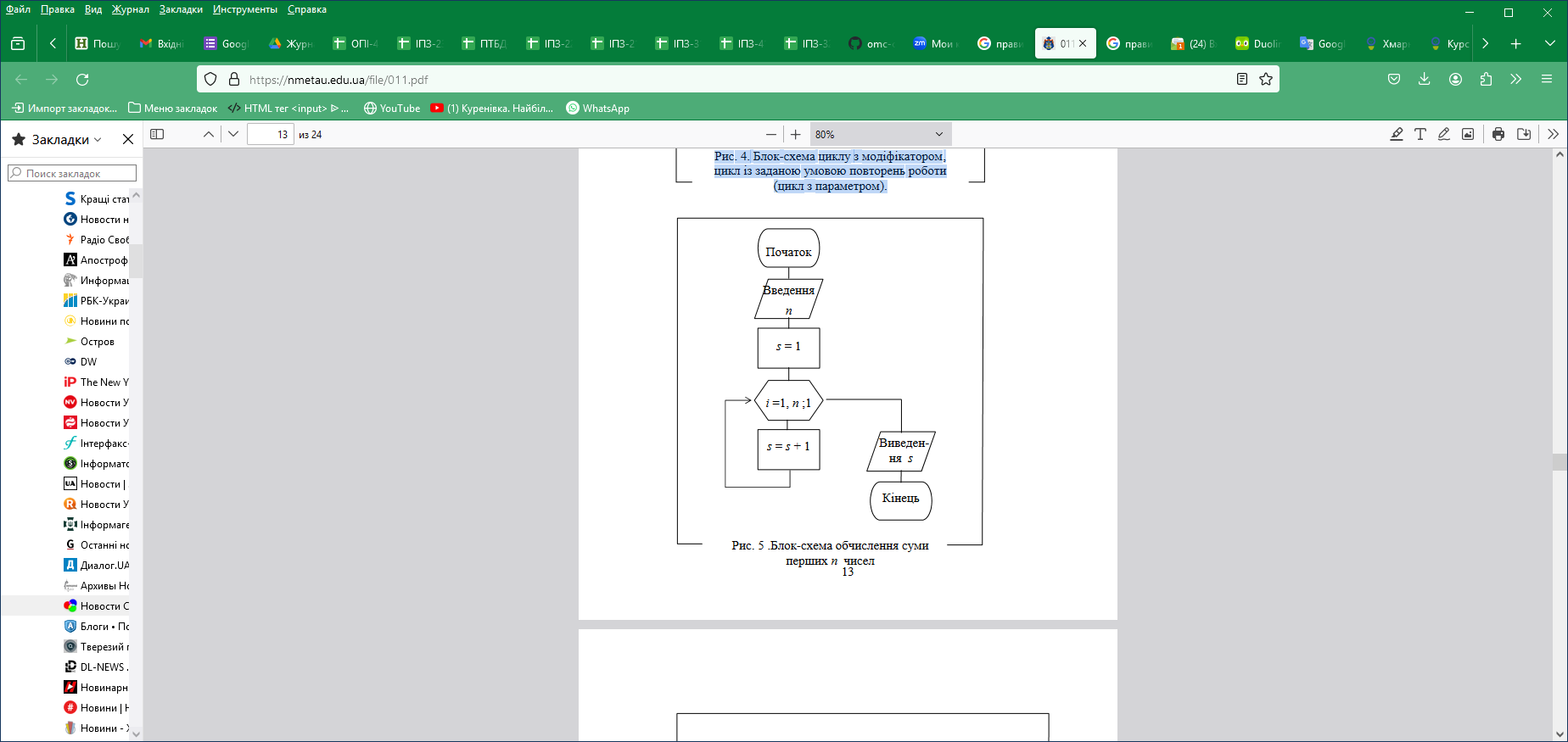


Рисунок 5 - Блок-схема обчислення суми перших n чисел

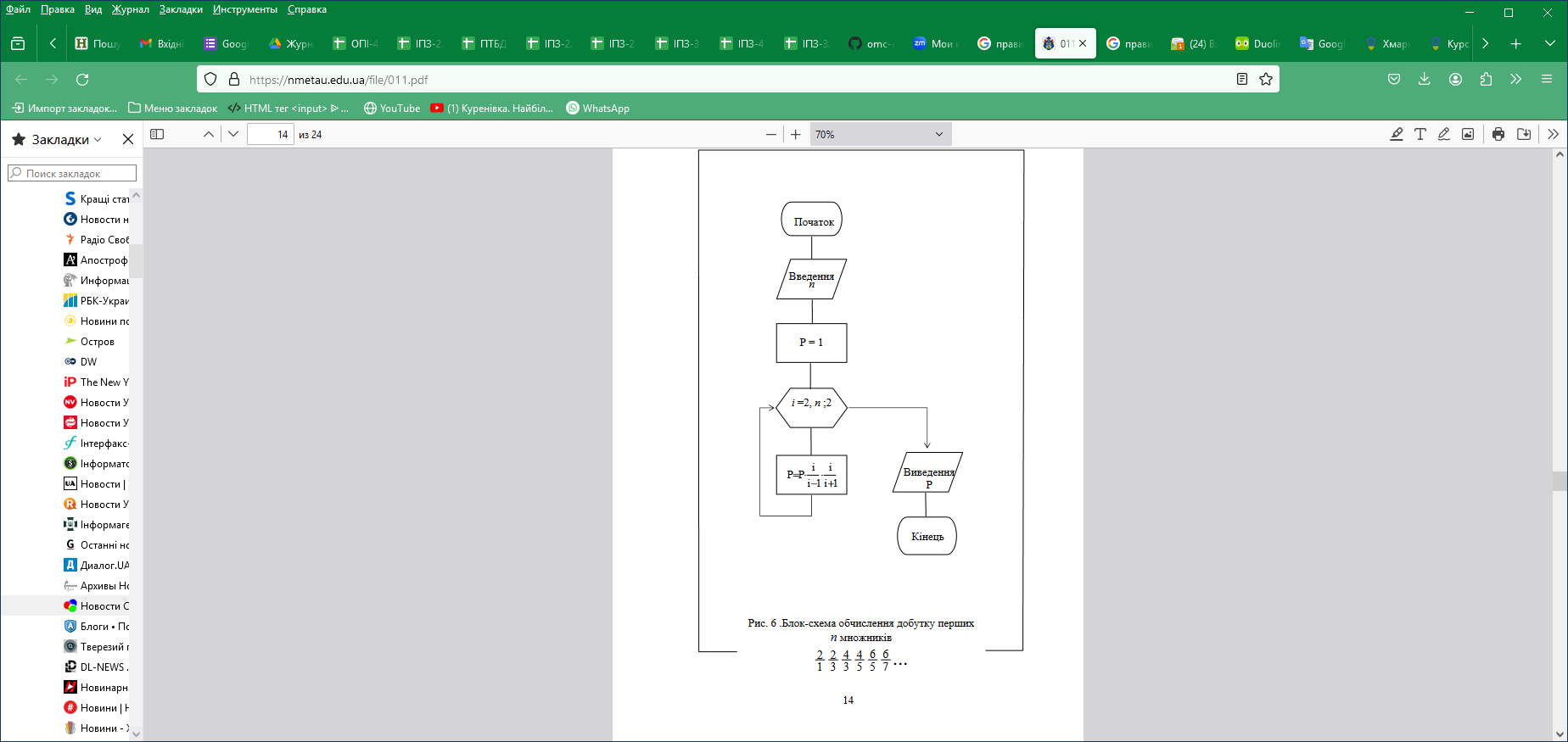


Рисунок 6 - Блок-схема обчислення добутку перших n множників

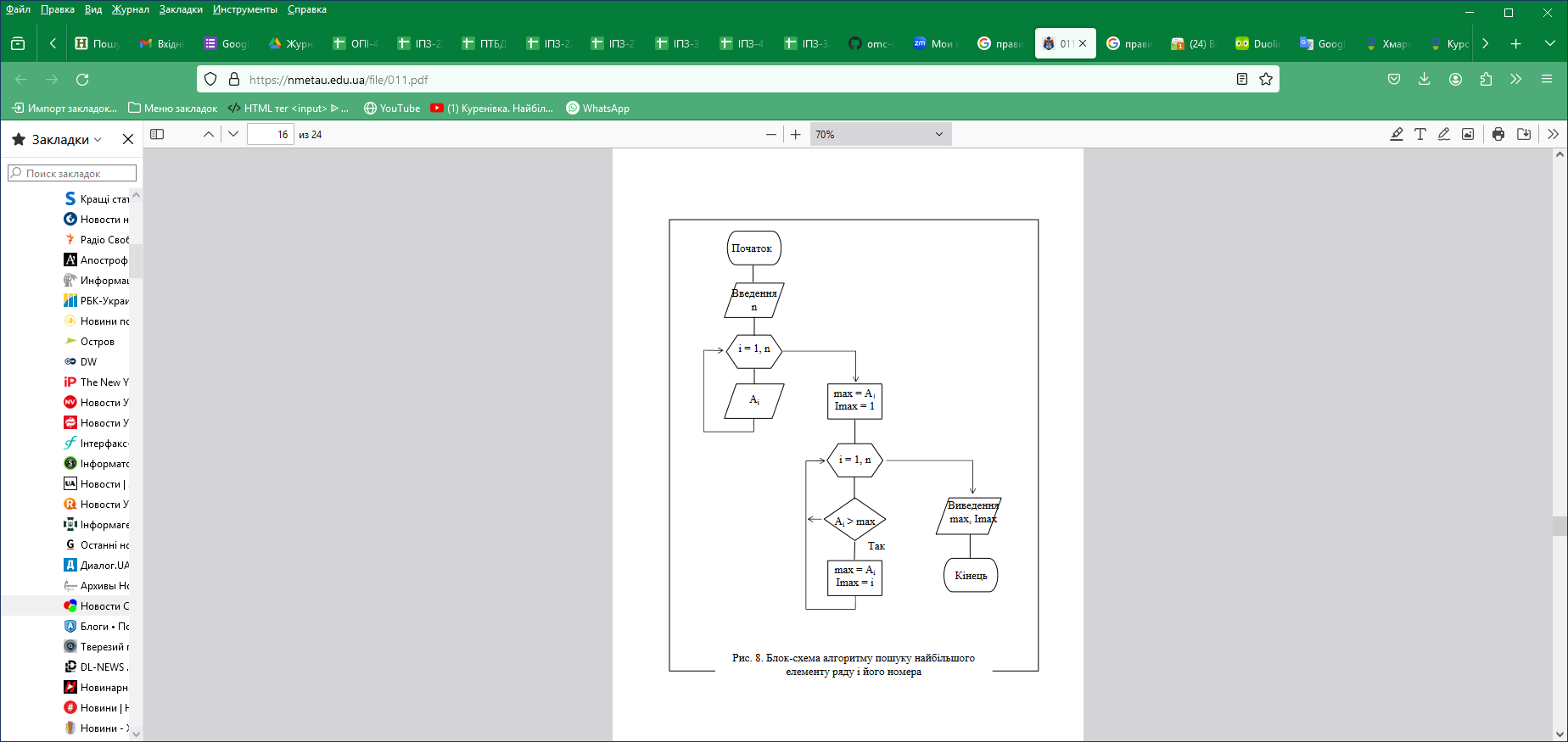


Рисунок 8 - Блок-схема алгоритму пошуку найбільшого елементу ряду і його номера

**Загальні поняття мови С++**

1972 році співробітник фірми **Bell Laboratories** Деніс Рітчі створив нову алгоритмічну мову — С. В її основу було закладено багато особливостей мови **Assembler**. Мова С є універсальною, придатною для розв’язання будь-якого типу задач, хоча спочатку була задумана як мова системного програмування (у 1973 році на мові С Деніс Рітчі реалізував операційну систему **Unix**).

У середині 80-х років Б’ярн Страуструп розробив мову «С з класами», що надалі стали називати мовою C++. Ця мова дозволяє працювати не тільки зі змінними, але і з їх адресами, розміщувати дані як у пам’яті, так і в регістрах, використовувати непряму адресацію (завдання адреси комірки, в якій зазначена адреса даного), автоматично змінювати адресу. Об’єктний код, що формується компіляторами мови C++, займає приблизно стільки ж пам’яті, скільки і відповідна програма на **Assembler**.

Мову C++ можна розглядати як надмножину мови С, бо вона зберігає усі можливості, що надає мова С, і доповнює їх засобами об’єктно-орієнтованого програмування. C++ є універсальною алгоритмічною мовою, яка використовується для розробки системних та складних прикладних програм. Це не тільки найпоширеніша мова програмування, але й мова спілкування програмістів, оскільки більшість програм алгоритмів написані на C++.

Мова C++ є мовою високого рівня і основою багатьох систем програмування, наприклад **Visual Studio**, **Eclipse**. Найбільш популярною з них вважається **Visual Studio**. За допомогою цієї системи візуального об’єктно-орієнтованого програмування як користувач-початківець, так і програміст-професіонал мають можливість створювати інтерфейс користувача до прикладних програм різноманітних класів, що виглядає однаково професійно.

У природній мові спілкування виділяють чотири основні елементи: символ, слово, словосполучення та речення. Подібні елементи існують і в алгоритмічній мові, тільки слова мають назву *лексеми*, словосполучення — *вирази*, а речення — *оператори*. Лексеми створюються із символів, вирази — із лексем та символів, оператори — з символів, виразів і лексем.

**Алфавіт** мови C++ включає:

* великі (**A-Z**) і малі (**a—z**) літери латинського алфавіту та символ підкреслення ( **\_** );
* арабські цифри від **0** до **9**;
* знаки арифметичних дій **+, -, \*, /, %, ++, —**;
* знаки побітових операцій **<<, >>, &, |, ~, ^**;
* знаки відношень **<, <=, ==, !=, >, >=**;
* знаки логічних операцій **&&, ||, !**;
* розділові знаки **, ; : пропуск**;
* спеціальні знаки **., =, ->, ?, \, $, #, ', "**;
* символи дужок **(, ), [, ], {, }**.

Інші символи, а також літери кирилиці не використовуються для побудови базових елементів мови або для їх розділу, але вони можуть застосовуватись у символьних константах та коментарях.

**Лексеми**, тобто базові елементи мови з певним самостійним значенням, складаються із символів алфавіту. До них відносять

* ідентифікатори,
* ключові слова,
* знаки операцій,
* константи,
* роздільники (дужки, крапка, кома, символи пропуску).

Межі лексем визначаються іншими лексемами-роздільниками або знаками операцій.

***Ідентифікатором***, тобто ім’ям програмного об’єкта, називається будь-яка послідовність літер латинського алфавіту, цифр i символу підкреслення за умови, що першою стоїть літера або символ підкреслення, а не цифра.

Існує два різновиди ідентифікаторів:

* *стандартні*, наприклад, імена всіх вбудованих у мову функцій;
* *користувальницькі*.

Мова C++ чуттєва до регістру літер, тому компілятор розпізнає великі і малі літери латинського алфавіту як різні символи. Це дає можливість створювати ідентифікатори, що однаково читаються, але відрізняються написом одного або декількох символів. Наприклад, ідентифікатори **«Sigma»**, **«sigma»** і **«sigMa»** вважаються різними.

Ідентифікатори можуть мати будь-яку довжину, але значимими є не більше 31 символу від початку ідентифікатора, a в деяких компіляторах це обмеження ще більш суворе (не більше 8 символів). Імена програмних об’єктів створюються на етапі оголошення даних, після цього їх можна використовувати в різних операторах програми.

**Рекомендації щодо вибору ідентифікатора**:

* рекомендується не жалкувати часу на створення ідентифікаторів переважно за їх змістовим призначенням;
* ім’я програмного об’єкта повинно легко розпізнаватись і, бажано, не мати символів, які можна переплутати між собою;
* ідентифікатор не повинен збігатися з ключовими словами, а також з іменами стандартних об’єктів мови C++;
* не слід починати ідентифікатори із символу підкреслення, бо вони можуть співпадати з іменами системних функцій або змінних;
* для розділу частин імені можна використовувати символ підкреслення;
* всередині ідентифікатора не можна розміщати символи пропуску.

***Ключовими*** (службовими) словами називають ряд зарезервованих ідентифікаторів, що вживаються для побудови конструкцій мови і мають фіксоване значення. За смисловим навантаженням службові слова поділяються на такі основні групи:

* специфікатори типів — **char, int, long, typedef, short, float, double, enum, struct, union, signed, unsigned, void;**
* квалифікатори типів — **const і volatile;**
* класи пам’яті — **auto, extern, register, static;**
* для побудови операторів — **for, while, do, if, else, switch, case, continue, goto, break, return, default, sizeof.**

Як роздільники лексем застосовуються такі символи: пропуск, табуляція, символ нового рядка, коментар. Між будь-якими двома лексемами допускається довільна кількість символів-роздільників. Крім того, деякі лексеми (**«\*», «+», «,», « », «(», «->»** тощо) самі є роздільниками і відділяти їх від інших лексем символами-роздільниками необов’язково.

Таблиця 1. Список ключових слів мови С++

****

**Спеціальні символи**

|  |  |
| --- | --- |
| Символ | Опис |
| **\n** | Новий рядок (LF) |
| **\t** | Горизонтальна табуляція (Н\_TAB) |
| **\r** | Повернення каретки (CR) |
| **\f** | Нова сторінка (FF) |
| **\a** | Звуковий сигнал |
| **\'** | Апостроф |
| **\”** | Лапки |
| **\\** | Обернена коса риска (back slash) |

**Структура програми**

Основними частинами типової структури програми на С++ є такі:

директиви препроцесорної обробки;

опис зовнішніх змінних (вихідних даних і результатів) та функцій;

функції програми;

головна функція — програми **main()**, що має вигляд:

main()

{

опис змінних; виконавчі оператори;

}

У загальному випадку програма складається з декількох функцій, що не перетинаються (тобто «вкладення» однієї функції в іншу неприпустиме). Перед функціями і між ними можуть бути присутні оголошення об’єктів даних і оператори препроцесорної обробки. Функції користувача, які викликаються у головній функції main(), слід обов’язково описати до їх використання. Приклад простої програми:

**//..............фрагмент простої програми на С++**

**#include <iostream.h>**

**#include <соnio.h> //директиви препроцесора**

**const int n=20;**

**void main()**

**{**

**float mas[n]; //опис одновимірного масиву**

**//........... ..введення елементів масиву**

**for (int i=0; i < n; i++)**

**cin >> mas[i];**

**// виконання перетворень та виведення перетвореного масиву**

**for(int i=0; i < n; i++)**

**{ mas[i]= mas[i]\*2;**

**cout << " " << mas[i] << " ";}**

**getch (); // затримка результату на екрані**

**}**

**Коментарі** необхідні для пояснень призначення тих чи інших частин програми і їх текст завжди ігнорується компілятором. Мова С++ використовує два різновиди коментарів:

* *// текст* — **однорядковий коментар**, який починається з двох символів «/» («коса риска») і закінчується символом переходу на новий рядок;
* */\* текст \*/* — **багаторядковий коментар**, що розташовується між символами-дужками «/\*» і «\*/».

Багаторядкові коментарі не можуть вкладатися один в один, а однорядкові коментарі можна вкладати в багаторядкові коментарі. Багаторядкові коментарі доцільно застосовувати для тимчасового виключення блоків при налагодженні програми.

Рекомендації щодо раціонального складання коментарів:

* коментарі повинні бути добре складеними реченнями, мати правильну пунктуацію та містити тільки потрібну для супроводу інформацію;
* пропуск — один з найбільш ефективних коментарів, що значно поліпшує розуміння програми;
* штрихові лінії коментаря або порожні рядки застосовуються для поділу функцій та інших логічно завершених фрагментів програм.

Директива препроцесора **#include<iostream>** забезпечує підключення до програми засобів зв’язку зі стандартними потоками введення-виведення даних. Ці засоби знаходяться у заголовному файлі **iostream.h**, де **і (*input*)** — введення, **о (*output*)** виведення, **stream** — поток, **h (*head*)** — заголовок. Оскільки середовище **Borland C++** не забезпечує затримку результатів на екрані, у текстах прикладів програм тут і далі використано стандартний потік введення **сіn** (бібліотека **iostream**), функцію введення **getch()** (бібліотека **conio.h**) тощо.

При створенні програми враховують такі основні вимоги:

* усі використані константи, змінні, функції та нестандартні типи повинні бути оголошеними (описаними) до їхнього першого використання, і ці оголошення можна розміщати в будь-якому місці програми, але для кращого сприйняття змісту краще робити це на початку;
* кожний оператор мови закінчується символом **«;»;**
* фігурні дужки (**« { »** та **« } »**) виділяють складений оператор і все, що подано між такими дужками, синтаксично сприймається як один оператор;
* для кращого сприйняття змісту вкладені блоки повинні мати відступ у 3-4 символи, при цьому блоки одного рівня вкладеності слід вирівняти за вертикаллю.

# Типи даних

Обробка даних різного типу є головною метою будь-якої програми. Кожне з даних характеризується класом пам’яті, ім’ям, типом і значенням. *Імена* дозволяють ідентифікувати дані, тобто відрізняти їх між собою. Програміст обирає тип кожної величини, що використовується для подання реальних об’єктів. *Тип* задає множину можливих значень даних і способи їх зберігання, перетворення та використання.

**Обов’язкове оголошення типу даних дозволяє** компілятору робити перевірку допустимості різних конструкцій програми.

Усі типи даних мови C++ можна розділити на **основні** (базові) і **складені**. Основні типи визначені для представлення цілих, дійсних, символьних і логічних даних. На основі цих типів вводиться опис складених типів, до яких належать масиви, перелічення, функції, структури, посилання, покажчики, об’єднання і класи.

**Основні типи даних** (*див. табл. 2, 3*) часто називають арифметичними, тому що їх можна використовувати в арифметичних операціях. Для опису основних типів мови C++ використовують такі службові слова:

* **int** (цілий);
* **char** (символьний);
* **bool** (логічний);
* **float** (дійсний);
* **double** (дійсний з подвійною точністю);
* **void** (порожній, не має значення).

Типи **int, char, bool** називають ***цілими***, а типи **float** та **double** — ***дійсними з плаваючою крапкою***. Код, що формує компілятор для обробки цілих величин, відрізняється від коду для величин з плаваючою крапкою.

Для уточнення внутрішнього подання та діапазону значень стандартних типів **мова C++ використовує чотири специфікатори типу**:

* **short** (короткий);
* **long** (довгий);
* **signed** (знаковий);
* **unsigned** (беззнаковий).

Таблиця 2. Дані цілого типу

|  |  |  |  |
| --- | --- | --- | --- |
| *Тип даних* | *Розмір,*  *байт* | *Нижня межа діапазону* | *Верхня межа діапазону* |
| [signed] char | 1 | -128 | 127 |
| unsigned char | 1 | 0 | 255 |
| [signed] short | 2 | -32 768 | 32 767 |
| unsigned short | 2 | 0 | 65 535 |
| [signed] int | 2 | -32 768 | 32 767 |
| 4 | -2 147 483 648 | 2 147 483 647 |
| unsigned int | 2 | 0 | 65 535 |
|  | 4 | 0 | 4 294 967 295 |
| [signed] long | 4 | -2 147 483 648 | 2 147 483 647 |
| unsigned long | 4 | 0 | 4 294 967 295 |

Таблиця 3. Дійсні типи даних

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Тип даних* | *Розмір,байт* | *Точність* | *Нижня межа діапазону* | *Верхня межа діапазону* |
| float | 4 | 1 | 3.4Е-38 | 3.4Е+38 |
| double | 8 | 15 | 1.7Е-308 | 1.7Е+308 |
| long double | 10 | 15 | 3.4Е-4932 | 1.1Е+4932 |

Розмір однакового типу даних може відрізнятися на комп’ютерах різних платформ, а також може залежати від застосованої операційної системи. Тому при оголошенні тієї чи іншої змінної потрібно чітко уявляти, скільки байт вона буде займати в пам’яті ЕОМ, щоб запобігти проблемам, пов’язаним з переповненням і неправильною інтерпретацією даних. Діапазони кожного з типів повинні бути перевірені для конкретного комп’ютера.

# Змінні

Кожна програма потребує виконання різноманітних обчислень, для здійснення яких використовуються вирази, що складаються з операндів, знаків операцій і дужок. Операнди задають дані для обчислень, а операції задають дії, які потрібно виконати над цими даними. Операнд є виразом, що в окремому випадку може бути константою або змінною.

**Змінна** — це іменована область пам’яті, у якій зберігаються дані визначеного типу. Змінна має ім’я, розмір та інші атрибути, такі як видимість, час існування тощо. *Ім’я* *змінної* служить для звертання до області пам’яті, у якій зберігається її значення. Кожна змінна повинна мати своє ім’я, причому в одному блоці не може бути двох змінних з однаковим ім’ям. ***Перед використанням будь-яка змінна повинна бути описана***, при цьому для неї резервується деяка область пам’яті, розмір якої залежить від конкретного типу змінної. Під час виконання програми змінна може приймати різні значення. Загальний вигляд опису змінних:

**[клас пам’яті] [const] тип ім’я [ініціювання];**

де необов’язковий *клас пам’яті* може приймати одне зі значень — **auto, extern, static** чи **register** (тут і далі при описі синтаксису об’єктів програмування необов’язкові частини синтаксичних конструкцій мови подано у квадратних дужках «[]»);

модифікатор **const** вказує, що змінна не може змінювати своє значення, у цьому випадку її називають **типізованою (іменованою) константою** або просто **константою**;

**ініціювання** — це присвоювання змінній при описі початкового значення, яке записується зі знаком рівності — **= значення** або в круглих дужках — (**значення**). Константа ***повинна бути ініційована при описі***. Один оператор може містити опис декількох змінних одного типу, розділяючи їх комами, наприклад:

**const int n = 20, m = 5, k = 4;** — ініціювання констант **n, m, k** цілого типу;

**float h = 17.5, d(5.5), sum**; — опис дійсних змінних **h, d, sum**, ініціювання **h і d**;

**char sf = 'f', st[ ] = "Knowledge is power."**; — ініціювання символьних змінних.

Якщо **тип** значення, що ініціюється, не збігається з типом змінної, то виконуються перетворення типу.

**Областю дії ідентифікатора змінної** є частина програми, в якій його можна використовувати для доступу до зв’язаної з ним області пам’яті. Залежно від області дії змінна може бути локальною або глобальною.

***Локальна змінна*** визначена всередині блока (блок розташований між фігурними дужками). Область її дії обмежена початком опису змінної та кінцем блока, включаючи усі вкладені блоки. Змінна, визначена поза будь-яким блоком, називається глобальною, і областю її дії вважається файл, у якому вона визначена від початку опису до його кінця.

**Клас пам’яті** визначає час існування та область видимості програмного об’єкта, тобто змінної. Якщо клас пам’яті не зазначений явно, то він визначається компілятором, виходячи, з контексту оголошення.

***Час існування*** змінної може бути постійним (протягом виконання програми) і тимчасовим (протягом виконання блока).

***Областю видимості*** ідентифікатора називається частина тексту програми, з якої можна здійснити звичайний доступ до зв’язаної з ідентифікатором області пам’яті. Найчастіше область видимості збігається з областю дії. Винятком є ситуація, коли у вкладеному блоці описана змінна з таким же ім’ям. У цьому випадку зовнішня змінна у вкладеному блоці невидима, хоча він і входить до її області дії. Проте до цієї змінної, якщо вона глобальна, можна звернутися, застосовуючи операцію доступу до області видимості — **“::“**.

Клас пам’яті задають такі специфікатори:

* **auto** — автоматична змінна, для якої пам’ять виділяється у стеку і за необхідності ініціюється кожного разу при виконанні оператора, що містить її визначення. Звільнення пам’яті відбувається при виході з блока, де описана змінна. Час її існування — з моменту опису до кінця виконання блока. Для глобальних змінних цей специфікатор не використовується, а для локальних він приймається за замовчуванням, тому можна не задавати явно;
* **extern** означає, що змінна визначена в іншому місці програми (в іншому файлі або далі по тексту) і використовується для створення змінних, доступних в усіх модулях програми, де вони оголошені. При ініціюванні змінної у тому ж операторі, спеціфікатор **extern** ігнорується;
* **static** — статична змінна, що має постійний час існування. Вона ініціюється один раз при першому виконанні оператора, що містить визначення змінної. Залежно від розташування оператора, описані статичні змінні можуть бути глобальними і локальними. Глобальні статичні змінні видимі тільки у тому модулі, в якому вони описані;
* **register** — аналогічний до специфікатора **auto**, але пам’ять виділяється по можливості в регістрах процесора і за відсутності такої можливості у компілятора змінні обробляються як **auto**.

Приклад фрагменту програми з використанням розглянутих вище понять:

**int d;** //1 — глобальна змінна **d**

**int main()**

**{**

**int b;** //2 — локальна змінна **b**

**extern int y;** //3 — змінна **у** визначена в іншому місці програми

**static int s;** //4 — локальна статична змінна **s**

**d = 1;** //5 — присвоювання значення глобальній змінній

**int d;**   //6 — локальна змінна **d**

**d = 10;** //7 — присвоювання значення локальній змінній

**::d = 3;**  //8 — присвоювання значення глобальній змінній

**return 0;**

**}**

**int у = 4;**       // 9 — визначення та ініціювання змінної **у**

У цьому прикладі глобальна змінна **d** визначена поза всіма блоками. Пам’ять для неї виділяється в сегменті даних на початку роботи програми, областю дії є вся програма. Область видимості — вся програма, крім рядків 6-8, тому що в першому з них визначається локальна змінна з тим же ім’ям, область дії якої починається з початку її опису і закінчується при виході з блока. Змінні **b i s** — локальні, область їх видимості — блок, але час існування різний: пам’ять під **b** виділяється в стеку при вході у блок і звільняється при виході з нього, а змінна **s** розташована у сегменті даних та існує увесь час роботи програми. Якщо початкове значення змінних явно не задається, компілятор присвоює глобальним і статичним змінним нульове значення відповідного типу. Автоматичні змінні не ініціюються. ***Початкове ініціювання змінних не є обов’язковим, проте все ж його бажано здійснювати***.

Опис змінної може виконуватися у формі оголошення або визначення. Оголошення інформує компілятор про тип змінної і класи пам’яті, а **визначення** містить, крім цього, вказівку компілятору про виділення пам’яті відповідно до типу змінної. У C++ більшість оголошень є одночасно і визначеннями (у наведеному вище програмному фрагменті тільки опис **extern int у**; є оголошенням, але не визначенням). ***Змінна може бути оголошена багаторазово, а визначена тільки в одному місці програми***, оскільки оголошення тільки описує властивості змінної, а визначення зв’язує її з конкретною областю пам’яті.

**Основні типи змінних.**

***Цілі змінні*** (типу **int, long, short**) необхідні для збереження цілих значень і можуть бути знаковими і беззнаковими. Знакові змінні застосовують для подання як додатних, так і від’ємних чисел, при цьому один біт (найстарший) виділяється під знак. Для оголошення беззнакової змінної, тобто змінної, що приймає тільки додатні значення, необхідно використовувати ключове слово **unsigned**. За замовчуванням будь-який цілий тип вважається знаковим, і тому немає потреби у використанні ключового слова **signed**.

***Символьний тип*** даних **char** застосовується у випадку, коли змінна містить інформацію про код ASCII або для побудови таких більш складних конструкцій, як рядки, символьні масиви тощо. Дані типу **char** також можуть бути знаковими і беззнаковими.

***Змінна типу* bool** займає 1 байт і використовується, насамперед, у логічних операціях, тому що може приймати значення **0** (**false** — «неправда») або відмінне від нуля (**true** — «істина»). У випадку перетворення до цілого типу **true** має значення 1.

Стандарт C++ визначає три типи даних для збереження ***дійсних значень змінних***: **float, double** та **long double** (типи ***з плаваючою крапкою***). Тип **float**, як правило, використовують для збереження не дуже великих дробових чисел.

***Змінна типу* void** не має значення, оскільки множина значень цього типу порожня. Такі змінні необхідні для узгодження синтаксису. Тип **void** використовується для визначення функцій, що не повертають значення, для вказівки порожнього списку аргументів функції, а також як базовий тип для покажчиків i в операції приведення типів. Наприклад, якщо немає потреби у використанні поверненого значення функції, перед ім’ям функції ставиться тип **void**:

**void minmax(int\*x, int k, int\*min, int&max);**.

# Константи

Константи являють собою фіксовані значення, що не можуть змінюватися впродовж виконання всієї програми.

Спосіб визначення кожної константи залежить від її типу. Константи мови С++ слід поділяти на літеральні та типізовані.

**Літеральна константа** — це лексема, що являє собою зображення фіксованого числового, рядкового або символьного значення. Такі константи бувають **цілі, дійсні, символьні та рядкові**

|  | Літеральні константи мови |  |
| --- | --- | --- |
| Константа | Формат | Приклади |
| Цiла | Десятковий: послідовність десяткових цифр  (0, 1, 2, 3, 4, 5, 6, 7, 8, 9), що починається не з нуля, якщо це не число нуль | 9, 0, 217925 |
|  | Вісімковий: нуль, за яким розташовані вісімкові цифри (0, 1, 2, 3, 4, 5, 6, 7) | 02, 050, 07245 |
|  | Шістнадцятковий: 0х чи 0Х, за яким розташовані шістнадцяткові цифри (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, А, В, С, D, Е, F) | 0x1B9, 0X00FF |
| Дійсна | Десятковий:[цифри].[цифри]  Експоненціальний: [цифри][.][цифри]{Е|е}[+|-][цифри] | 9.7, .001, 87. 0.7Е6, .15е-3 9.2, 920 е-2, 92.Е-1, .92Е1 |
| Символьна | Один чи два символи, що подаються в апострофах | ‘А’ , ‘ю’ , ‘\* ‘ , ‘db’ , ’\0′, ‘\n’, ’\012′, ‘\х07\х07’ |
| Рядкова | Послідовність символів, що подають­ся в лапках | “RESULT”, |

**Цілі константи** можуть бути десятковими, вісімковими та шiстнадцятковими.

Довгі цілі константи (**long**) мають літеру **l** або **L** в кінці, наприклад: 32768L; 0777777l; 0XFL. Для завдання константи, без знака (**unsigned**) застосовується літера **u (U)**,наприклад 65535u. Довгі константи без знака записуються з використанням двох літер відразу: **(ul, UL)** або **(lu, LU)**.

**Дійсні числа**у мовах програмування мають дві форми подання: десяткову (природну) та експоненціальну (показникову).

***Десяткова форма***дійсного числа — це звичайний десятковий формат запису дійсного числа, тільки частина дійсного числа відділяється від дробової крапкою, а не комою, наприклад 10.123, 1.0123, 1012.3, 0.0010123.

**Експоненціальна форма**дійсного числа використовується для запису дуже великих або дуже малих чисел, для яких задавати зайві нулі не зовсім зручно, наприклад: 1 .0123\*1020,1.0123\*10-10. У цій формі запису числа можна виділити такі основні характеристики: знак числа, мантису числа, знак порядку та порядок числа. Зазначені характеристики дійсного числа зберігаються у пам’яті комп’ютера. Число у показниковій формі може бути представлено, наприклад, так: 1.0123Е-10. Мантиса записується ліворуч від знаку експоненти(***Е чи е***), порядок — праворуч. Символ **Е (е)** означає основу ступеня 10, і компілятор розпізнає цей запис як форму представлення дійсного числа. Символ пропуску всередині числа не допускається, а для відділення цілої частини від дробової використовується не кома, а крапка. При додатних значеннях числа і мантиси знак «+» можна не вказувати.

Як десяткова, так і екcпоненціальна форми запису допускають відсутність або цілої частини, або дробової, але не двох одразу.

За замовчуванням всі дійсні константи мають тип **double** — подвійну точність, що найчастіше займає в пам’яті 64 біти, тобто 8 байтів. Але у випадку, якщо програміста не влаштовує тип за замовчуванням, його можна вказати явно за допомогою спеціальних літер. Так, додавши літеру **f чи F**, константі надають дійсний тип **float** зі звичайною точністю, наприклад, 8.5f. Якщо в представленні константи використовується літера **L чи l**, то вона має тип **long double**.

Зображення від’ємної цілої чи дійсної константи вважається константним виразом, що складається зі знаку унарної операції зміни знаку (-) та константи, наприклад: -273, -2730.е-1, -273L.

***Символьні константи***мають один або два символи, що подаються в апострофах. Односимвольні константи займають у пам’яті один байт і мають стандартний тип **char** (***character-символ***). Двосимвольні константи займають два байти і мають тип **int**. Символьні константи мають цілий тип і їх можна ви­користовувати як цілочислові операнди у виразах.

**Послідовності.**

Послідовності починаються зі знаку «\», їх називають **керуючими**або **escape-послідовностями.**Символ зворотної косої риски «\» використовується для запису кодів, що не мають графічного зображення, для запису символiв, а також для виведення символьних констант, якщо їх коди заданi у вісімковому та шістнадцятковому вигляді (*табл. 4*).

*Таблиця 4* **Керуючі послідовності мови С++**

|  |  |
| --- | --- |
| \а | звуковий сигнал |
| \b | повернення на крок |
| \f | переведення сторінки (формату) |
| \n | новий рядок |
| \r | повернення каретки |
| \t | горизонтальна табуляція |
| \v | вертикальна табуляція |
| \\ | символ \ — зворотна коса риска |
| \' | символ ‘ — апостроф |
| \" | символ ” — лапки |
| \0 | нуль-символ |
| \? | знак питання |
| \0ddd | вісімковий код символу |
| \0xddd | шістнадцятковий код символу |

**Рядкова константа *(рядковий літерал)***— це послідовність cимволів, що подається в лапках (тобто в символах **«"»**)і зберігається у неперервній ділянці пам’яті, наприклад: **"**As busy as a bee**"**. У кінець кожного рядкового літералу компілятором додається нуль-символ, що предствляється керуючою послі­довністю **«\0».** Тому довжина рядка завжди на одиницю більше кількості символів у його записі. Таким чином, порожній рядок (” “) має довжину 1 байт. Слід звернути увагу на різницю між рядком з одного символу, наприклад, “С” і символьною конс­тантою ‘С’ . Порожня символьна константа неприпустима.

Керуючі послідовності можуть також застосовуватись у рядкових константах. Так, якщо всередині рядка потрібно записати лапки, то перед ними слід розташувати зворотну косу риску («\»), за якою компілятор відрізняє їх від лапок, що обмежують рядок:

Рядки, що записані у програмі підряд або через символи пропуску, при компіляції конкатенуються («склеюються»). Тобто послідовність двох рядків

**"** **As the tree, so the fruit** **"**

**“** **- Яблуко від яблуні недалеко падає. ”**

цілком еквівалентна рядку:

**“ As the tree, so the fruit - Яблуко від яблуні недалеко падає. ”.**

Довгу рядкову константу можна розмістити також на декількох рядках. У цьому випадку ставиться зворотна коса риска і натискається клавіша **Enter**. Наприклад:

**“Програма виконує те, \**

**що ви їй наказали робити, а не те, \**

**що ви хотіли, щоб вона робила.”**

Існує інша можливість завдання констант — з використанням директиви препроцесора **#define**,при цьому оголошення має вигляд:

**#define ім’я константи значення константи**

і наприкінці такого запису символ **«;»** не ставиться, тобто:

**#define max 65532**

**#define km 1000.**

Директива **#define** визначає ідентифікатор (*ім’я константи*) і послідовність символів (*значення константи*), яка замінює ідентифікатор у тексті програми.

**Нульовий покажчик** (*NULL-покажчик*)— єдина неарифметична константа мови C++.

При застосуванні великої кількості логічно взаємозалежних констант C++ доцільно користуватися константами перелічення. Тип перелічення має вигляди:

**enum {список іменованих констант}; — неiменоване перелічення,**

**enum [ім’я] {список іменованих констант}; — iменоване перелічення.**

де **enum** — службове слово (**enumerate** — перелічувати);

**ім’я** — ім’я списку констант;

**список іменованих констант** — розділена комами послідовнiсть iдентифікаторів або іменованих констант вигляду:

**ім’я константи = значення константи.**

Наприклад:

**enum {Anton, Ivan, Piter};**

**enum Months {January = 1, February, Marth, April, May, June, July, August, September, October, November, December};**

Якщо значення константи перелічення не визначено, то воно на одиницю більше значення попередньої константи. За замовчуванням перша константа має значення **0**. Тоді у першому прикладі константи одержать значення: **Anton = 0, Ivan = 1, Piter = 2**, а у другому — значення: J**anuary = 1, February = 2, Marth= З** тощо. Іменовані перелічення задають унікальний цілочисловий тип і можуть використовуватися як специфікації типу для визначення змінних.

# Операції

Для здійснення маніпуляцій з даними мова C++ застосовує широкий набір операцій (див. *табл. 5*), що виконують формування і, відповідно, подальше обчислення виразів. Вирази містять **одну** або **декілька** операцій, об’єкти яких називають операндами. Операції являють собою деяку дію, що виконується над **одним** (***унарна***) або **декількома** (***бінарна, триарна***) операндами, і мають позначення (наприклад, операція перевірки на рівність — позначення «==», операція обчислення залишку від ділення цілих чисел — позначення «%» тощо).

Операції поділяються на:

* ***унарні*** або ***одномісні*** — **&, \*, -, +, ~, !, ++, –-, sizeof;**
* ***бінарні*** або ***двомісні*** — **+, -, \*, /, %, <<,  >>, &, :, ^, <, >, <=, ==, >=, !=, &&, ||, =,\*=, /=, %=, +=, -=, <<=, >>=, &=, |=, ^=, ., ->, ,, (), [];**
* умовну ***триарну*** або ***тримісну*** операцію — **?: .**
* **Основні операції мови C++**

|  |  |  |
| --- | --- | --- |
| № | Операції | Порядок виконання |
| 1 | ()  ,  {}  ->  . | Л -> П |
| 2 | !   ~   ++  —  &  \*  (type) | П -> Л |
| 3 | sizeof | П -> Л |
| 4 | \*   /   % | Л -> П |
| 5 | +   – | Л -> П |
| 6 | <<   >> | Л -> П |
| 7 | <   <=   >   >= | Л -> П |
| 8 | ==   != | Л -> П |
| 9 | & | Л -> П |
| 10 | ^ | Л -> П |
| 11 | | | Л -> П |
| 12 | && | Л -> П |
| 13 | || | Л -> П |
| 14 | ?: | П -> Л |
| 15 | =   +=   \*=   -=   /=   %= | П -> Л |
| 16 | , | Л -> П |

Порядок застосування операції визначається **пріоритетом операції** (яка операція виконується раніше, а яка пізніше) та **асоціативністю** (виконується зліва направо або справа на ліво). У першу чергу реалізуються операції з найвищим пріо­ритетом.

У табл.5 літерою **«Л»** позначено величину, що стоїть ліворуч від знака операції, літерою **«П»** — величину, яка розташована праворуч від знака операції, а символом «->» напрямок виконання операції. Розглянемо основні операції.

***Арифметичні операції:***

**+** — додає величину **П** до **Л**;

* + — віднімає **П** із **Л**;
  + — унарна операція зміни знаку величини **П**;

**\***     — множення **П** і **Л**;

**/**  — ділення **Л** на **П**;

**%**  — залишок від ділення величини **Л** на величину **П** (для цілих чисел), наприклад, якщо **int g = 12;**, то операція **g = g % 9;** надасть результат: **g = 3;**

**++** — унарна операція інкремент. Якщо змінна розташовується праворуч від знаку операції (префіксна форма), то значення збільшується на 1 до використання. Якщо ж змінна знаходиться ліворуч від знаку операції (постфіксна форма), то її значення збільшується на 1 після використання, наприклад:

**int d;**

**++d;** — префіксний інкремент,

**d++;** — постфіксний інкремент;

-- — унарна операція декремент аналогічно інкременту має двi форми: префіксную (змінна розташована праворуч від знаку операцii) — зменшення значення змінної на 1 відбувається до її використання; постфіксну (змінна знаходиться ліворуч від знака операції) — зменшення значення змінної на 1 після її використання.

**Операції присвоювання:**

**=** — присвоювання значення **П** змінній **Л**;

**+=**  — додає величину **П** до змінної **Л**;

**–=** — віднімає величину **П** від змінної **Л**;

**\*=**  — множення змінної **Л** на величину **П**;

**/=** — ділення **Л** на **П**;

**%=** — видає залишок від ділення **Л** на **П**.

Просте присвоювання здійснює операція «=». Допускається одночасне зчіплювання декількох операцій присвоювання за умови, що всі операнди мають однаковий тип, наприклад:

int і, j, с;

i **= j = с = 0;**

Операції **«+=», «-=», «\*=», «/=»** виконують складні присвоювання і дозволяють записувати вирази коротше, наприклад:

**s += 7;**       //s=s + 7;

**і \*= j + 5;**   //i=i\*(j +5);

g%=9;        //g=g%9;.

**Операції відношення** порівнюють значення **Л** зі значенням **П**:

< — менше;

<= — менше або дорівнює (не перевищує);

== — дорівнює;

> — більше;

>= — більше або дорівнює (не менше);

!= — не дорівнює.

У мові C++ «істина» — це ненульова величина, «неправда» — це нуль **(0)**. У більшості випадків одиниця **(1)** викориcтовується як ненульове значення.

Операції відношення повертають ціле значення 1, якщо умова вірна, або 0, якщо умова помилкова.

**Логічні операції** оперують з цілими розмірами або з розмірами, які можна перетворити на цілі. Обчислення зупиняється, які тільки визначиться, чи є вираз правдивим («істина») або помилковим («неправда»). При цьому, як і для операцій відношення, значенням «істина» відповідає 1, а значенням «неправда» — 0.

&& — логічне **«AND»** (кон’юнкція);

|| — логічне **«OR»** (диз’юнкція);

!= — логічне **«NOT»** (заперечення).

Результат операції «&&» є «істина» **(1)**, якщо обидва її oпeранди правдиві (не рівні 0). Результат операції «||» — «істина» **(1)**, якщо хоча б один з її операндів є «істина». Логічне заперечення «!=» перетворює свій операнд на «істину» **(1)**, якщо він дорівнює **0**, і на «неправду» **(0)**, якщо він не дорівнює **0**.

З  використанням логічних операцій та операцій відношення записуються різні умовні вирази, наприклад, умова **3 < х < 5** ма­тиме вигляд: **х > 3 && х < 5**.

**Операції обробки окремих бітів** застосовують для обробки даних як послідовностей бітів (розрядів), кожний з яких набуває значення **0** або **1**.

& — операція бітового множення (кон’юнкція);

| — операція бітового додавання (диз’юнкція);

^ — додавання за модулем 2 (результат дії дорівнює 1, якщо число складаємих одиничних бітів непарне, якщо ж їх число парне, то результат дорівнює 0.);

~ — інвертування (побітове **NOT)**;

>> — зсув праворуч;

<< — зсув ліворуч.

**Змінна-покажчик** зберігає значення, що є адресою об’єкта в пам’яті комп’ютера. Через покажчик можна звертатися до об’єкта.

**Операції з адресами та покажчиками:**

& — одержання адреси: видає адресу змінної, ім’я якої роз­ташоване праворуч від позначення операції;

\*  — непряма адресація (розіменування): видає значення, записане за адресою, на яку посилається покажчик.

**Додаткові операції:**

**sizeof()** — знаходить розмір (у байтах) операнду, розташованого праворуч від назви операції;

**(type)** — операція приведення типу перетворює наступне за нею значення в тип, визначений ключовим словом, укладеним у круглі дужки, наприклад:

**i = i+(int)\*3.14;**

**?** —  **триарна** (з трьома операндами)     операція, що має вигляд:

**вираз1? вираз2 : виразЗ;,**

тут, якщо результат обчислення першого операнду **(вираз1)** не дорівнює **0** («істина»), то результатом операції буде значення другого операнду **(вираз2)**, інакше — третього операнду **(виразЗ)**. Наприклад, знаходження найбільшої з двох величин **а** і **b**, можливо здійснити операцією: **max = (b > а)? b : а;**.

**Перетворення типів**

***Правила приведення типів:***

1. Автоматично здійснюються тільки ті перетворення, які перетворюють операнди з меншим діапазоном значень в операнди з більшим діапазоном значень, наприклад:

**int i\_var = 5;**

**float f\_var = 2.5, summa;**

**…**

**summa = i\_var + f\_var;**

2. Вирази, що не мають змісту (наприклад, число з плаваючою комою в ролі індексу), не пропускаються компілятором ще на етапі трансляції:

***float f;***

***…***

***mas [f]=25; // викликає помилку трансляції (Error)***

3. Вирази, в яких могла б втрачатися інформація (наприклад, при присвоєнні довгих цілих коротшим або дійсних цілим), можуть викликати попередження (Warning), але вони допустимі:

***int i;***

***float f=3.2;***

***i=f; // попередження (Warning) при трансляції***

***Для самостійного вивчення .*** Історія створення та розвитку мови С++. Сумісність та перетворення типів [1, с.53-56]. Поглибити матеріал лекції за наданою літературою. Вивчення лекційного матеріалу та додаткових джерел. Розгляд запитань і виконання завдань для самостійної роботи, запропонованих на лекції.

***Контрольні запитання*.**

1. Що включає в себе алфавіт мови С++?
2. Що таке лексеми, як вони складаються, що до них відносять?
3. Дайте визначення ідентифікатору, яким чином він утворюється?
4. Що називають ключовими словами?
5. Які символи можуть бути роздільниками лексем?
6. Які коментарі використовує мова C++? Надайте поради до їх створення, наведіть приклади.
7. Що являє собою структура програми на C++? Наведіть основні вимоги, які слід ураховувати при створенні програм мовою C++.
8. Які основні та складені типи даних має мова С++?
9. Що таке змінна і як здійснюється її опис та визначення?
10. Що таке «область дії ідентифікатора» та «клас пам’яті»?
11. Які константи налічує С++? Охарактеризуйте їх застосування.
12. Що таке пріоритет операції? Наведіть приклади арифметичних та логічних операцій.
13. Які операції присвоювання та операції відношення налічує С++?
14. Що реалізують логічні операції та операції обробки окремих бітів?
15. Які операції над покажчиками та додаткові операції має С++?

***Література***

1. 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>
2. Джейс Либерти Освой самостоятельно С++ за 21 день: 3-е изд. пер. с англ.: Уч. пос. – М.: Издательский дом „Вильямс”, 2001. – 816 с.: ил..
3. Бондарев В. М. Программирование на С++: Учеб. пособие. — Харьков: СМИТ, 2004г. — 294 с. URL: <https://www.rulit.me/author/bondarev-v-m/programmirovanie-na-c-get-161082.html>
4. Липпман С. Б., Лажойе Ж. Язык программирования С++: Вводный курс. — М.: ДМК, 2001. URL: <http://www.insycom.ru/html/metodmat/inf/Lipman.pdf>
5. Дейтел Х., Дейтел П. Основы программирования на С++. – М.: Бином, 1999. – 1024 с.