Лінійні програми на Сі

Програма на С (C++) – це сукупність текстових файлів (зазвичай заголовочних та вхідних) (header and source files) які містять декларації ([declarations](https://en.cppreference.com/w/cpp/language/declarations)). Вони компілюються для того, щоб утворити виконуваний файл якщо C++ фали містять головну функцію ([main function](https://en.cppreference.com/w/cpp/language/main_function)). Якщо її немає, то програма може бути перетворена в бібліотеку.

Програма на мові Сі (Сі++) побудована за допомогою спеціальних символів (на кшталт {,}, # etc) та ключових слів ([keywords](https://en.cppreference.com/w/cpp/keyword)). Інші слова можуть служити як ідентифікатори ([identifiers](https://en.cppreference.com/w/cpp/language/identifiers)). Крім того, програма може містити коментарі ([Comments](https://en.cppreference.com/w/cpp/comment)) які ігнорують під час компіляції. Деякі символи можуть бути представлені в програмі за допомогою ([escape sequences](https://en.cppreference.com/w/cpp/language/escape)).

Ідентифікатори ([identifiers](https://en.cppreference.com/w/c/language/identifier)), на Сі можуть ідентифікувати об’єкти ([objects](https://en.cppreference.com/w/c/language/object)), функції ([functions](https://en.cppreference.com/w/c/language/functions)), структури ([struct](https://en.cppreference.com/w/c/language/struct)), об’єднаня ([union](https://en.cppreference.com/w/c/language/union)), перерахування([enumeration](https://en.cppreference.com/w/c/language/enum)) та їх члени, типи, що визначені користувачем ([typedef](https://en.cppreference.com/w/c/language/typedef) names), мітки ([labels](https://en.cppreference.com/w/c/language/statements" \l "Labels)) та макроси([macros](https://en.cppreference.com/w/c/preprocessor/replace)).

Сутності представлені деклараціями ([declarations](https://en.cppreference.com/w/cpp/language/declarations)), які представляють їх за іменами ( [names](https://en.cppreference.com/w/cpp/language/name)) та визначають їх властивості. Декларації визначають всі властивості що потрібні для використання сутності як визначення ([definitions](https://en.cppreference.com/w/cpp/language/definition)). Програма має містити лише одне визначення будь-якої не підставляємої (non-inline) функції або змінної.

Визначення функції складається з послідовності тверджень ([statements](https://en.cppreference.com/w/cpp/language/statements)), деякі з яких включають вирази ([expressions](https://en.cppreference.com/w/cpp/language/expressions)), які визначають які обчислення повинна виконати програма.

Імена, що виникають в програмі співставленні з визначеннями, що представляють ними за допомогою визначення імен ([name lookup](https://en.cppreference.com/w/cpp/language/lookup)). Кожне ім’я діє лише в середині частини програми що зветься областю визначення ([scope](https://en.cppreference.com/w/cpp/language/scope)). Деякі імена можуть мати зв’язок ([linkage](https://en.cppreference.com/w/cpp/language/storage_duration)) що визначає посилання до тих самих сутностей коли вони посилаються на ті самі сутності коли вони з'являютсья в різних областях визначення або translation units.

Декларації та вирази створюють, знищують, отримують доступ та маніпулюють об’єктами. Кожен об’єкт, функція та вираз в Сі асоційовані з певним типом.

Кожен об’єкт, посилання, функція, вираз в C++ асоційований з типом ([type](https://en.cppreference.com/w/cpp/language/type)), який може бути базовим ([fundamental](https://en.cppreference.com/w/cpp/language/types)), составним (compound), або визначений користувачем ([user-defined](https://en.cppreference.com/w/cpp/language/classes)), повним (complete) або неповним ([incomplete](https://en.cppreference.com/w/cpp/language/incomplete_type)), і так далі.

Визначені об’єкти (declared objects) або визначені посилання (declared references) які не є не статичними членами даних ([non-static data members](https://en.cppreference.com/w/cpp/language/data_members)) звуться змінними (*variables)*.

Приклад програми на Сі

/\*

Optional, but recommended: Documentation section

Необов’язкова але рекомендована частина

i.e. comments, that describe the program, like:

Заголовочні коментарі

First C-style program - Перша програма:

Обчислення синуса

\*/

#include <stdio.h> // Link section - заголовочний файл,

//бібліотека стандартного вводу-виводу

#include <math.h> // Link section заголовочний файл,

//бібліотека математичних функцій

//Definition Section and Global declaration section

//Тут можуть бути визначення макросів

//Тут можуть бути визначення глобальних змінних та констант

//Тут можуть бути визначення або декларація функцій

int main() // головна функція (main function): точка входу (entry point)

{

// Визначення та/або ініціалізація локальних змінних

float x; //визначаємо дійсну (одинарної точності) змінну 'x'

scanf("%F",&x); // введення змінної 'x'

double y=sin(x); /\* Вираз (expression): виклик функції sin,

обчислення виразу та

ініціалізація дійсної змінної (подвійною точності) ‘y’

\*/

printf("Result y=%f\n",y); // виведення значення змінної y

}

Ще один приклад, що не підключає математичну бібліотеку, але використовує стандартне введення-виведення

/\*

Second C-style program - Обчислення температури по Фаренгейту

\*/

#include <stdio.h> // Бібліотека стандартного вводу-виводу

//(без неї нема printf та scanf)

int main(){ // точка входу

// Визначення та/або ініціалізація локальних змінних

float F, C; //визначаємо відразу дві дійсні – змінні ‘F’ та ‘C’

printf(“F=”); // виводимо підказку для користувача

scanf("%f", &F); // введення змінної ‘F’

C=(F-32)\*5/9; /\* Обчислення за формулою \*/

printf("Celsius C=%e\n",С); /\* виведення значення змінної ‘C’ в науковому форматі \*/

}

Ці файли потрібно зберігати з розширенням “.с”, для того щоб система та компілятор зрозуміли що це файл для компіляції Сі-компілятором. Для того, щоб цю програму можна було відкомпілювати компілятором с++ потрібно трохи модифікувати код та зберегти його з розширенням “.cpp”:

#include <cstdio> // Link section: , бібліотека стандартного

// вводу-виводу С інтегрована як С++ бібліотека

#include <cmath> // заголовочний файл,

//бібліотека математичних функцій з С в С++

int main() // головна функція (main function): точка входу (entry point)

{

float x; //визначаємо дійсну (одинарної точності) змінну 'x'

scanf("%F",&x); // введення змінної 'x'

double y=sin(x); /\* Вираз (expression): виклик функції sin,

обчислення виразу та

ініціалізація дійсної змінної (подвійною точності) ‘y’

\*/

printf("Result y=%f\n",y); // виведення значення змінної y

}

Відмітимо, що мова Сі – чутлива до регістрів(кейс-сенсітів), тобто регістр символів має значення.

Щодо **коментарів** в коді. В сучасній мові Сі існує два типи коментарів, тобто ділянок коду що не компілюються:

- ***Коментарій ділянки коду*,** що може бути як на часину рядка так і на декілька рядків:

/\* якийсь текст тут \*/

/\*

текст,

знову текст

….

ще щось ….

\*/

- ***Коментарій до кінця рядку:***

// текст до кінця рядка ігнорується компілятором

Можна побачити також, що всі команди на Сі завершуються крапкою з комою, а блоки програми виділяються фігурними дужками.

На відміну наприклад від Python немає різниці як розташовувати пробіли та табуляції в текстів програми. Але *вкрай бажано використовувати певний стиль програмування*, схожий до того що використовується в Python – одна команда на рядок, табуляція при виділенні програмних блоків, фігурні дужки на окремих рядках ().

Отриманий файл програми тепер потрібно відкомпілювати та запустити.

Компіляція

Оскільки мова Сі – це компілятор, то середовище програмування повинно перетворити цю програму (або пакет програм) на файл, що може виконуватися. Для цього вона має виконати наступні дії (building):

1. **Препроцесінг (Preprocessing):** файли C++ проекту оброблюються таким чином, що текст програми замінюється на зрозумілі компілятори символи, замість #include, #define та інших препроцесорних команд підставляється код відповідних файлів та команд, видаляються коментарі. Результатом препроцесінгу стає "чистий" C++ файл – проміжне представлення (intermediate representation).
2. **Компіляція (Compilation):** компілятор перетворює проміжний файл (translation unit, intermediate compiled output file), у файл що готовий для компіляції асемблером даного компілятору на даній платформі. Отриманий результат може бути включений до статичних бібліотек.
3. **Збирання (Асемблювання) (Assembly)** (часто вказують як частину процесу компіляції) – створює файл як набір асемблерних (машинних) інструкцій (object file) таким чином що код стає тим що можна переміщувати.
4. **Лінковка (Linking)** об’єктні файли, що згенервані компілятором та генерує виконуваний файл або бібліотеку (статична чи динамічна лінковка).

Процес компіляції можна запустити в командному рядку або в інтегрованому програмному середовищі.

На Linux в командному рядочку:

Для компіляції Сі-файлу:

gcc hello1.c

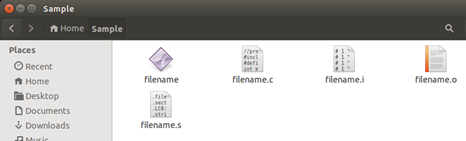
Для компіляції Сі++:

g++ hello1.cpp

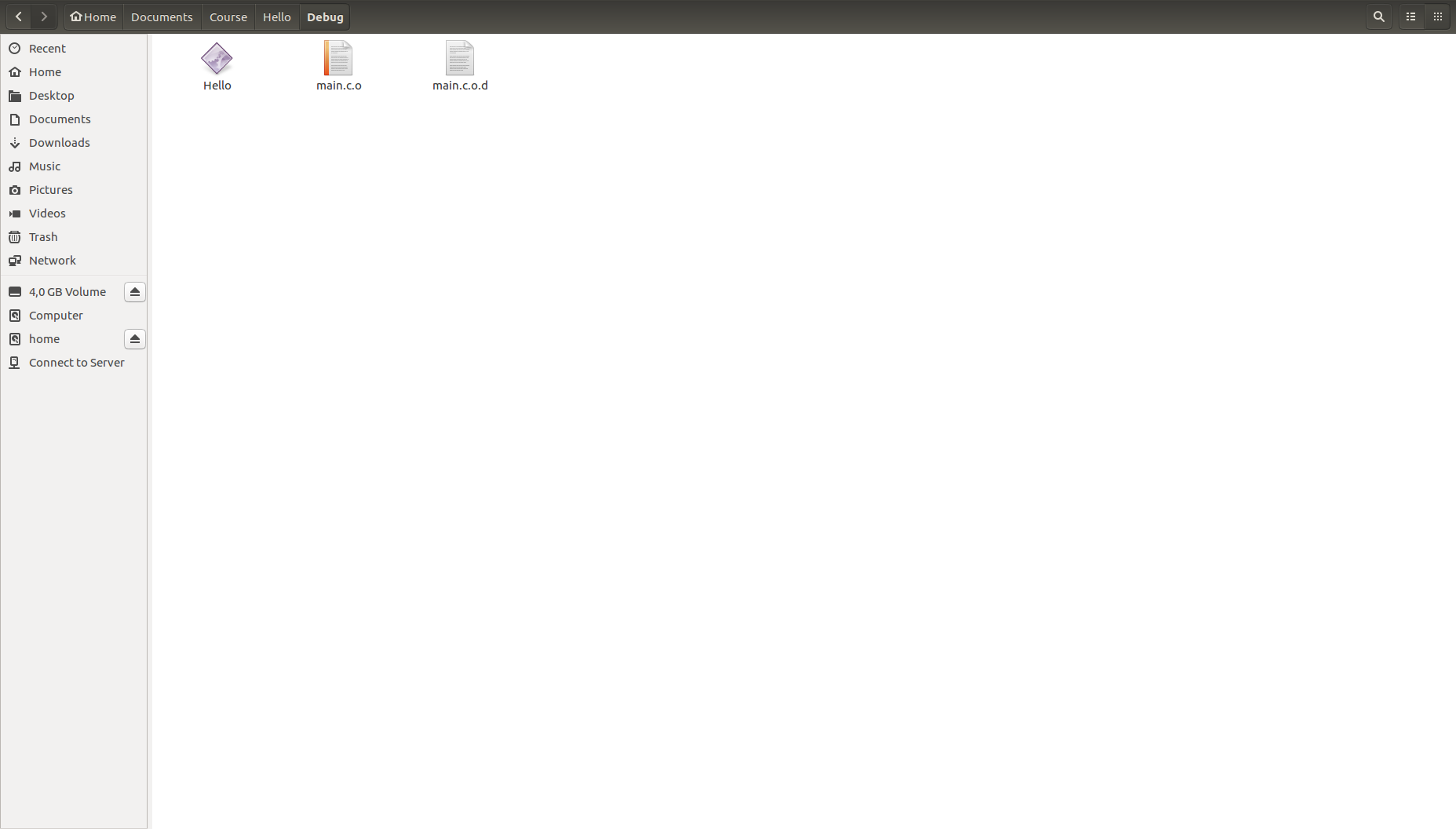
Результатом повинні бути виконуваний файл **hello1** та файл ***hello1.o –*** бібліотека (об’єктний файл)

Більш докладна команда, що показує проміжні етапи видасть результат

$gcc –Wall –save-temps filename.c –o filename

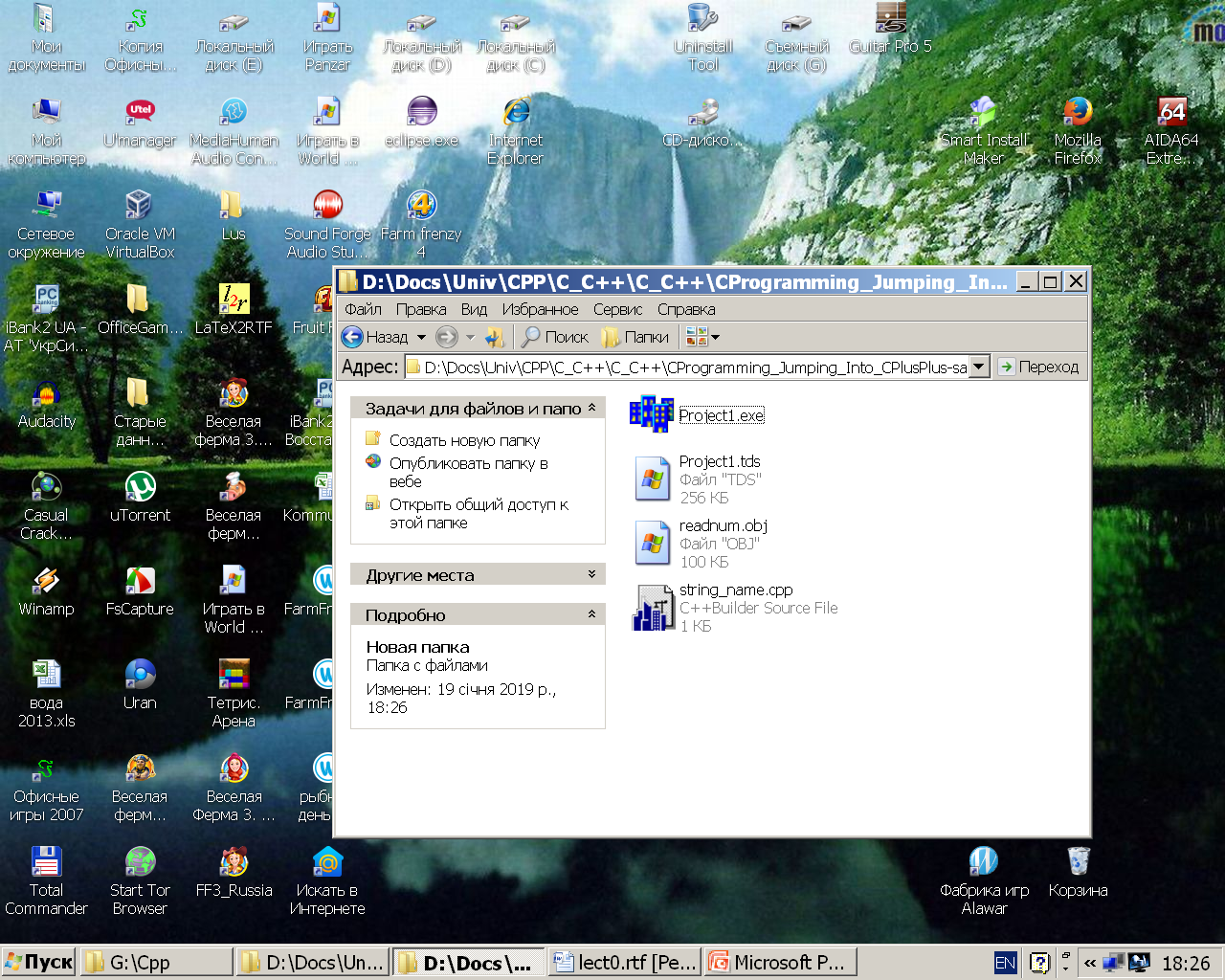


В середовищі CodeLite результат наступний

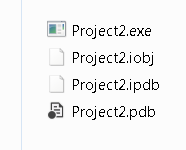
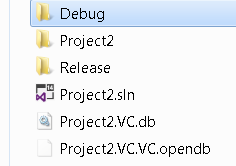


На Windows:

Borland Builder



А Visual Studio 15 зробить наступні папки та вміст ітогової папки Release:



Для запуску в командному рядку можна використати різні компілятори, зокрема:

MS Visual Studio: **cl hello1.c**

MinGW( Eclipse, CodeLite: **gcc hello1.c** )

Turbo C: **tc hello1.c**

Borland Builder C++: **bc hello1.c**

Іншим шляхом компіляції може бути створення проекту за допомогою середовища розробки, MS Visual Studio, Eclipse, CodeLite, Code:Blocks, Borland Builder і т.п. (в даному випадку потрібно створити консольний проект) та запустити відповідною кнопкою запуску. В цьому випадку середовище розробки само повинно згенерувати команду запуску та можливо створити автоматичний скріпт (більш докладно про це в наступних главах) для запуску програми.

Структурні частини програми на Сі

Токени C

На мові C програма складається з різних токенів, які є або ключове слово(keyword), ідентифікатор (identifier), константа (constant), рядкова одиниця (string literal), або спецсимвол (symbol). Наприклад, printf("Hello, World! \n"); складається з наступних токенів

printf - команда

( - дужки, спецсимвол

"Hello, World! \n" – рядкова константа

) - дужки, спецсимвол

; крапка з комою, спецсимвол

Крапки з комою

В мові C крапка з комою – кінець твердження. Таким чином, кожна інструкція повинна закінчуватися нею. Компілятор тоді розуміє що вона закінчилась і потрібно перевести її в машинний код.

Приклади інструкцій:

printf("Hello, World! \n");

return 0;

Коментарі

Коментарі ці допоміжний текст в C програмі який ігнорується компілятором. Вони починаються 2 символами /\* та закінчуються 2-ма символами \*/ як показано нижче −

/\* my first program in C \*/

Неможна коментувати таким стилем всередині коментарю такого стиля та всередині літералу.

Інший стиль коментарів що дозволений в сучасному С – це рядковий коментар який робиться підвійним символом слеша // як в прикладі:

float x=1.0; // ініціалізуємо дійсну (одинарної точності) змінну 'x'

Ідентифікатори

Ідентифікатор C – це ім’я, яке використовується для ідентифікації змінної (variable), функції (function) та інших ідентифікаторов що визначені користувачем. Ідентифікатор починається з латинської літери (або від A до Z, або від a до z) або з нижнього підкреслення '\_' за яким слідує 0 або більше літер, нижніх підкреслювань та цифр (від 0 до 9).

C не дозволяє символів @, $, та % всередині ідентифікаторів. C - **case-sensitive** програмна мова. Тобто, *Power* та *power* – два різні ідентифікатори в C. Ось приклади ідентифікаторів через кому:

Modx , zara, abc, move\_name, a\_123, myname50, \_temp, j, J, a23b9, retVal.

Ключові слова

В таблиці приведені ключові слова C. Ці слова не можуть бути використані як ідентифікатори чи імена модулів.

Таблиця 1

|  |  |  |  |
| --- | --- | --- | --- |
| [auto](https://en.cppreference.com/w/c/keyword/auto) [break](https://en.cppreference.com/w/c/keyword/break) [case](https://en.cppreference.com/w/c/keyword/case) [char](https://en.cppreference.com/w/c/keyword/char) [const](https://en.cppreference.com/w/c/keyword/const) [continue](https://en.cppreference.com/w/c/keyword/continue) [default](https://en.cppreference.com/w/c/keyword/default) [do](https://en.cppreference.com/w/c/keyword/do) [double](https://en.cppreference.com/w/c/keyword/double) [else](https://en.cppreference.com/w/c/keyword/else) [enum](https://en.cppreference.com/w/c/keyword/enum) [extern](https://en.cppreference.com/w/c/keyword/extern) | [float](https://en.cppreference.com/w/c/keyword/float) [for](https://en.cppreference.com/w/c/keyword/for) [goto](https://en.cppreference.com/w/c/keyword/goto) [if](https://en.cppreference.com/w/c/keyword/if) [inline](https://en.cppreference.com/w/c/keyword/inline) (since C99) [int](https://en.cppreference.com/w/c/keyword/int) [long](https://en.cppreference.com/w/c/keyword/long) [register](https://en.cppreference.com/w/c/keyword/register) [restrict](https://en.cppreference.com/w/c/keyword/restrict) (since C99) [return](https://en.cppreference.com/w/c/keyword/return) [short](https://en.cppreference.com/w/c/keyword/short) | [signed](https://en.cppreference.com/w/c/keyword/signed) [sizeof](https://en.cppreference.com/w/c/keyword/sizeof) [static](https://en.cppreference.com/w/c/keyword/static) [struct](https://en.cppreference.com/w/c/keyword/struct) [switch](https://en.cppreference.com/w/c/keyword/switch) [typedef](https://en.cppreference.com/w/c/keyword/typedef) [union](https://en.cppreference.com/w/c/keyword/union) [unsigned](https://en.cppreference.com/w/c/keyword/unsigned) [void](https://en.cppreference.com/w/c/keyword/void) [volatile](https://en.cppreference.com/w/c/keyword/volatile) [while](https://en.cppreference.com/w/c/keyword/while) | [\_Alignas](https://en.cppreference.com/w/c/keyword/_Alignas) (since C11) [\_Alignof](https://en.cppreference.com/w/c/keyword/_Alignof) (since C11) [\_Atomic](https://en.cppreference.com/w/c/keyword/_Atomic) (since C11) [\_Bool](https://en.cppreference.com/w/c/keyword/_Bool) (since C99) [\_Complex](https://en.cppreference.com/w/c/keyword/_Complex) (since C99) [\_Generic](https://en.cppreference.com/w/c/keyword/_Generic) (since C11) [\_Imaginary](https://en.cppreference.com/w/c/keyword/_Imaginary) (since C99) [\_Noreturn](https://en.cppreference.com/w/c/keyword/_Noreturn) (since C11) [\_Static\_assert](https://en.cppreference.com/w/c/keyword/_Static_assert) (since C11) [\_Thread\_local](https://en.cppreference.com/w/c/keyword/_Thread_local) (since C11) |

Деякі ключові слова починаються з нижнього підкреслення:

Таблиця 2

|  |  |  |
| --- | --- | --- |
| Ключове слово | Використовується як | Визначено в |
| [\_Alignas](https://en.cppreference.com/w/c/keyword/_Alignas) (since C11) | [alignas](https://en.cppreference.com/w/c/types) | stdalign.h |
| [\_Alignof](https://en.cppreference.com/w/c/keyword/_Alignof) (since C11) | [alignof](https://en.cppreference.com/w/c/types) | stdalign.h |
| [\_Atomic](https://en.cppreference.com/w/c/keyword/_Atomic) (since C11) | [atomic\_bool, atomic\_int, ...](https://en.cppreference.com/w/c/atomic) | stdatomic.h |
| [\_Bool](https://en.cppreference.com/w/c/keyword/_Bool) (since C99) | [bool](https://en.cppreference.com/w/c/types/boolean) | stdbool.h |
| [\_Complex](https://en.cppreference.com/w/c/keyword/_Complex) (since C99) | [complex](https://en.cppreference.com/w/c/numeric/complex/complex) | complex.h |
| [\_Generic](https://en.cppreference.com/w/c/keyword/_Generic) (since C11) | (no macro) |  |
| [\_Imaginary](https://en.cppreference.com/w/c/keyword/_Imaginary) (since C99) | [imaginary](https://en.cppreference.com/w/c/numeric/complex/imaginary) | complex.h |
| [\_Noreturn](https://en.cppreference.com/w/c/keyword/_Noreturn) (since C11) | [noreturn](https://en.cppreference.com/w/c/types) | stdnoreturn.h |
| [\_Static\_assert](https://en.cppreference.com/w/c/keyword/_Static_assert) (since C11) | [static\_assert](https://en.cppreference.com/w/c/error/static_assert) | assert.h |
| [\_Thread\_local](https://en.cppreference.com/w/c/keyword/_Thread_local) (since C11) | [thread\_local](https://en.cppreference.com/w/c/thread/thread_local) | threads.h |

Також є спецсимволи диграфи <%, %>, <:, :>, %:, та %:%: що представляють альтернативи звичайним токенам.

Директиви, що відповідають макросам (перед ними стоїть символ #):

|  |  |  |
| --- | --- | --- |
| [if](https://en.cppreference.com/w/c/preprocessor/conditional) [elif](https://en.cppreference.com/w/c/preprocessor/conditional) [else](https://en.cppreference.com/w/c/preprocessor/conditional) [endif](https://en.cppreference.com/w/c/preprocessor/conditional) [defined](https://en.cppreference.com/w/c/preprocessor/conditional) | [ifdef](https://en.cppreference.com/w/c/preprocessor/conditional) [ifndef](https://en.cppreference.com/w/c/preprocessor/conditional) [define](https://en.cppreference.com/w/c/preprocessor/replace) [undef](https://en.cppreference.com/w/c/preprocessor/replace) | [include](https://en.cppreference.com/w/c/preprocessor/include) [line](https://en.cppreference.com/w/c/preprocessor/line) [error](https://en.cppreference.com/w/c/preprocessor/error) [pragma](https://en.cppreference.com/w/c/preprocessor/impl) |

Цей специфічний токен розпізнається якщо він знаходиться зовні макросів:

|  |
| --- |
| [\_Pragma](https://en.cppreference.com/w/c/preprocessor/impl)(since C99) |

Наступні два токена вважаються доповненнями до мови Сі:

|  |
| --- |
| [asm](https://en.cppreference.com/mwiki/index.php?title=c/language/asm&action=edit&redlink=1) [fortran](https://en.cppreference.com/w/c/keyword/fortran) |

Пробіли в C

Лінія що містить пробіли(whitespace), можливо з коментарієм,зветься порожньої лінією та ігнорується компілятором C .

Пробілами (Whitespace) в C також звуть порожні лінії, табуляцію, символ переходу на новий рядок та коментарі. Пробіли також відділяють елементи однієї інструкції від іншої та дозволяють йому зрозуміти елементи інструкції. Таким чином в інструкції

int age;

повинен бути хоча б один роздільник (пробіл) між int та age щоб компілятор розрізнив їх. З іншого богу в інструкції

fruit = apples + oranges; // get the total fruit

непотрібні пробіли між fruit та =, або між = та apples, хоча їх бажано там ставити для гарного вигляду коду.

## Введення/виведення

На С++ введення та виведення можна робити за допомогою команд

std::cin>> // команда введення

std::cout<< // команда виведення

Приклад:

#include <iostream>

int main(){

int x;

std::cin>>x;

std::cout<<”x=”<<x;

}

## Форматоване виведення

Розглянемо класичну запропоновану одним з співавторів Сі Керніганом програму привітання “Hello, world!”. На Сі вона виглядатиме наступним чином:

#include <stdio.h>

int main(){

printf(“Hello, world!\n”);

}

Вона виводить текст “Hello, world!” та переводить курсор на новий рядок за допомогою спецсимволу ‘\n’.

Команда виведення інформації на консоль - printf. Як і деякі інші команди вводу/виводу в форматованому стилі для С, її опис міститься у заголовочному файлі**<stdio.h>**:

**рrintf** (<керуючий рядок>, <список аргументів>);

Керуючий рядок береться у лапки і вказує компілятору вигляд інформації, що виводиться. Вона може містити специфікації перетворення і керуючи або escape-символи.

Специфікація перетворення має такий вигляд:

**%**<**флаг**> <**розмір поля. точність**>**специфікація**,

де **флаг** може набувати наступних значень:

**-** вирівнювання вліво числа, що виводиться (за замовчуванням виконується вирівнювання вправо);

**+** виводиться знак додатного числа;

**розмір поля** – задає мінімальну ширину поля, тобто довжину числа. Якщо ширини поля недостатня, автоматично виконується його розширення;

**точність** – задає точність числа, тобто кількість цифр його дробової частини;

**специфікація** вказує на вигляд інформації, що виводиться. У таблиці 3 наведено основні формати функції друку.

Таблиця 3

|  |  |
| --- | --- |
| Формат | Тип інформації, що виводиться |
| %d | десяткове ціле число |
| % і | для виведення цілих чисел зі знаком (printf (“a=%i”, -3)); |
| %u | для виводу беззнакових цілих чисел (printf(“s=%u”, s)) |
| %c | один символ |
| %s | рядок символів |
| %e | число з плаваючою крапкою (експоненційний запис) |
| %f | число з плаваючою крапкою (десятковий запис) (printf(“b=%f\n, c=%f\n, d=%f\n”, 3.55, 82.2, 0.555 )); |
| %u | Десяткове натуральне число |

**Керуючий рядок** може містити наступні **керуючі символи:**

**\n** – перехід на новий рядок;

**\t** – горизонтальна і **\v** – вертикальна табуляція;

**\b** – повернення назад на один символ;

**\r** – повернення на початок рядка;

**\a** – звуковий сигнал;

\” –лапки;

\? – знак питання;

\\ - зворотний слеш.

**Список аргументів** – об’єкти, що друкуються (константи, змінні). Кількість аргументів та їх типи повинні відповідати специфікаціям перетворення в керуючому рядку.

*Приклад 1.*

#include <stdio.h>

const float pi = 3.14158f;

void main()

{

int number=5, cost=11000, s=-777;

float bat=255, x=12.345;

printf ("%d students drank %f bottles.\n", number, bat);

printf ("Value of pi is%f.\n", pi);

printf ("The price is %d%s\n", cost,"у.е");

printf ("x=%-8.4f s=%5d%8.2f ", x, s, x);

}

В результаті виконання останньої функції **printf**() на екрані буде виведено:

х=12.3450 s= -777 12.34.

Локалізація

Нажаль, коли ми спробуємо вивести інформацію за допомогою іншого алфавіту, зокрема українського, можливе виникнення того, що замість символів алфавіту виведуться якийсь незрозумілі символи. Це пов’язано з тим, що Сі та навіть Сі++ розроблявся в ті часи, коли розробка програмного забезпечення велося цілком на англійському, а кодування та відображення символів було залишено на програміста, стандартних функцій для роботи з кодуваннями не було, юнікоду також.

Звісно, таке становище дуже шкодило переносимості програм, і були розроблені стандартні засоби для роботи з різними кодуваннями, для C - бібліотека <locale.h> (clocale в C++) із функцією

char\* setlocale (int category, const char\* locale);

Ця функція змінює поведінку стандартних функцій C для роботи з рядками у відповідності до локалі і категорії, причому категорії бувають:

LC\_COLLATE - впливає на функції strcoll and strxfrm

LC\_CTYPE - впливає на функції з cctype, крім isdigit and isxdigit

LC\_MONETARY - впливає на інформацію про грошові одиниці з функції localeconv.

LC\_NUMERIC - впливає на форматування чисел при вводі-виводі (зокрема, на десяткову кому) і включає LC\_MONETARY

LC\_TIME - впливає на strftime

LC\_ALL - включає все попереднє

Другий параметр - назва локалі - залежить від ОС; Windows та Linux розуміє [прості назви на кшталт "Ukrainian" чи "Russian"](http://msdn.microsoft.com/en-us/library/ms912047(v=winembedded.10).aspx); іншим системам треба давати більш точні вказівки типу "uk\_UA.cp1251" чи "en\_US.utf8" (у форматі мова\_країна.кодування).  
Для C++ була повністю перероблена бібліотека cctype (ctype.h), і тепер є бібліотека locale, що містить всі важливі функції з ctype і ще купу різних .  
Крім того, в сучасних стандартах була додана підтримка юнікоду: довгі символи (wchar\_t), довгі рядки (L"日本語" буде кодовано в wchar\_t, а не в char), бібліотеки <cwchar> (wchar.h) та <cwctype> (wctype.h), класс std::wstring і т.д.

Нажаль, при використанні старих версій Сі незважаючи на зміні в цій бібліотеці все одно можуть виникнути проблеми, тому краще користуватись свіжими компіляторами для уникнення цих проблем.

Для того щоб встановити власну локаль корисно користуватись наступними автоматичними локалями

Таблиця 4

|  |  |
| --- | --- |
| Ім’я локалі | Опис локалі |
| "C" | Мінімальна локаль "C" |
| "" | Локаль яка прописана в системі |

Приклад:

|  |
| --- |
| */\* setlocale example \*/*  #include <stdio.h> */\* printf \*/*  #include <locale.h> */\* struct lconv, setlocale, localeconv \*/*  *int* main ()  {  printf ("Locale is: %s\n", setlocale(LC\_ALL,NULL) );  setlocale (LC\_ALL,"");  struct lconv \*lc;  lc = localeconv ();  printf ("Currency symbol is: %s\n-\n",lc->currency\_symbol);  setlocale (LC\_ALL,"Ukrainain");  printf ("наша валюта: %s\n-\n",lc->currency\_symbol);  return 0;  } |

За допомогою структури lconv можна також змінювати можливість вводу виводу дійсних чисел у форматі крапки чи коми, параметри виводу часу і т.п., а функції [localeconv](http://www.cplusplus.com/localeconv) повертає в програму відповідну структуру для роботи з нею.

Зупинка терміналу

Ще одна проблема, яка може виникнути при запуску програмного застосування – це те що після запуску програми вікно терміналу, що з’явиться після виконання програми після виконання закриється і не дасть насолодитися красотою виведеного тексту. Деякі середовища та платформи залишають термінал чекати спеціального його закриття, деякі дозволяють тримати його в фоні і дають можливість побачити, а деякі відразу закриваються після виконання. В останньому варіанті може допомогти декілька варіантів трюків. Перший – якщо середовище дозволяє робити відлагодження (дебаггінг, debug) програми, то можна поставити точку зупинки (брейкпойнт) перед останнім return і тоді дочекатись коли програма зупиниться перед виходом. Іншим варіантом є додати останню команду вводу символу з клавіатури getchar – ця команда чекає натиснення на клавіатуру:

#include <stdio.h>

int main(){

printf(“Hello, world!\n”);

getchar(); // або int c = getchar();

}

Форматоване введення

Функція **scanf** передбачена для форматного вводу інформації довільного вигляду. Загальний вигляд функції:

**scanf** (<керуючий рядок>, < список адрес>);

На відміну від функції виводу **printf**(), **scanf**() використовує у списку адреси змінних, для одержання яких перед іменем змінної ставиться символ ”&”, що позначає унарну операцію одержання адреси. Для вводу значень рядкових змінних символ ”&” не використовується. При використанні формату %s рядок вводиться до першого пропуску. Вводити дані можна як в одному рядку через пропуск, так і в різних рядках.

Дану особливість ілюструє відповідна частина програми:

int course;

float grant;

char name[20];

printf ( "Вкажіть ваш курс, стипендію, ім’я \n");

scanf ( "%d%f", &course, &grant);

scanf ( "%s", name); /\* ”&” відсутній при зазначенні масиву символів \*/

Для введення в форматованому вигляді використовуються майже ті самі флагі та специфікації, що використовуються і для функції **printf.**

В середовищі Visual Studio можливе виникнення наступної проблеми: при компілюванні або зборці програми цей компілятор видасть помилку:

Error 1 error C4996: 'scanf': This function or variable may be unsafe. Consider using scanf\_s instead. To disable deprecation, use \_CRT\_SECURE\_NO\_WARNINGS. See online help for details.

Ця помилка пов’язана з тим, що команда scanf() допускає так звану Buffer Overflow Problem, що дозволяє хакерам ломати комерційні застосування, а Visual Studio в свою чергу вимагає більш безпечного коду від програміста. Для компіляції коду в цьому середовищі можливі наступні опції:

1. Поставити при компіляції флаг \_CRT\_SECURE\_NO\_WARNINGS у властивості проекту, або макрос #define \_CRT\_SECURE\_NO\_WARNINGS на початок проекту
2. Поставити макрос #pragma warning(disable:4996) на початок проекту
3. Скористатись як радить компілятор функцією форматованого вводу scanf\_s. Зауважимо, що ця функція увійшла в формат Сі починаючі з версії С11.

#include <stdio.h>

int main(){

int i, result;

float fp; char c, s[81];

wchar\_t wc, ws[81];

result = scanf\_s( "%d %f %c %C %80s %80S", &i, &fp, &c, &wc, s, ws );

// C4996 warning consider using scanf\_s

printf( "The number of fields input is %d\n", result );

printf( "The contents are: %d %f %c %C %s %S\n", i, fp, c, wc, s, ws);

}

### Примітка: Більшисть специфікаторів враховує пробіли всередині scanf(). Послідовні команди

scanf("%d", &a);

scanf("%d", &b);

зчитують два цілих числа в різних рядках (друге %d вставить новий рядок після вводу) або на той самій лінії відокремлені пробілами або табуляціями (другий %d пропустить пробіли та табуляції). Специфікатори які не пропускають пробіли такі як %c, можна примусити приймати їх за допомогою пробілів перед специфікатором:

scanf("%d", &a);

scanf(" %c", &c); // пропустить всі пробіли перед %d, пітім символ

Визначення змінної в C

Визначення ( означення, декларація) змінної призначено для того, що компілятор визначив де та скільки потрібно пам’яті виділити під цю змінну. Для цього в визначенні потрібно вказати тип даної змінної та вказати одну або більше ідентифікаторів змінних через кому.

**type variable\_list;**

Тут, **type** повинен бути типом мови C включаючи char, w\_char, int, float, double, bool, та будь-який визначений користувачем (user-defined) об’єкт; а список змінних **variable\_list** може складатись або більше ідентифікаторів змінних через кому. Приклади декларацій змінних:

int i, j1, k\_2;

char c, ch;

float f, salary;

double d;

Рядок **int i, j1, k\_2;** визначає цілі змінні i, j1, та k\_2; ця інструкція каже компілятору створити змінні що звуться i, j1 та k\_2 типу int.

Змінні можуть бути **ініціалізовані** (initialized) (присвоєно початкове значення) під час свого визначення. Ініціалізатор складається зі знаку рівне (equal sign), за яким йде вираз:

type variable\_name = value;

Приклади:

extern int d = 3, f = 5; // ініціалізація цілих змінних d і f.

int d = 3, f = 5; // ініціалізація d і f.

byte z = 22; // ініціалізація z.

char x = 'x'; // змінній x присвоєно значення 'x'.

Якщо змінна визначена без ініціалізації: змінні з static storage duration ініціалізовані NULL (всі байти мають значення 0); значення всіх інших змінних невизначено, тобто може бути яким завгодно.

**Присвоєння** змінної приводить до того, що компілятор гарантує, що існує змінна даного типу та іменем, так що компілятор може проводити операції з цієї змінної без запиту. Це знання зберігається лише під час компіляції, під час лінковки потрібне також визначення цієї змінної.

Різниця між присвоєнням та ініціалізацією C:

Таблиця 5

|  |  |
| --- | --- |
| Присвоєння | Ініціалізація |
| Присвоєння повідомляє компілятору про тип даних та розмір змінної. | Визначення визначає розмір пам’яті під змінну |
| Змінна може бути перевизначена декілька разів | Може відбутись лише 1 раз. |
| Присвоєння значення та властивостей змінній |  |

Вираз: Операції С та С++

***Вираз*** за допомогою якого компілятор розуміє, що потрібно виконати певне обчислення або операцію над змінною складається з ідентифікаторів, що позначають змінні та позначок операцій, що відповідають певним діям, що компілятор повинен вміти з ними робити.

***Позначки*** ***операцій*** – це один або декілька символів, що визначають дію над операндами. Операції поділяють на унарні, бінарні та тернарні за кількістю операндів, які беруть участь в операції (таблиця 6).

Таблиця 6.

|  |  |  |
| --- | --- | --- |
| Операція | | Короткий опис |
| **Унарні операції** | | |
| & | Операція одержання адреси операнда | |
| \* | Звернення за адресою (розіменування) | |
| - | Унарний мінус – змінює знак арифметичного операнда | |
| ~ | Порозрядове інвертування внутрішнього двійкового коду (побітове заперечення) | |
| ! | Логічне заперечення (НЕ) значення операнда. Цілочисельний результат 0 (якщо операнд ненульовий, тобто істинний) або 1 (якщо операнд нульовий, тобто хибний). Таким чином: !1 дорівнює 0; !2 дорівнює 0; !(-5)=0; !0 дорівнює 1. | |
| ++ | Інкремент (збільшення на одиницю):  *Префіксна операція (++х)* збільшує операнд на 1 до його використання.  *Постфіксна операція (х++)* збільшує операнд на 1 після його використання.  int m=1, n=2;  int a=(m++)+n; // a=3, m=2, n=2  int b=m+(++n);// b=6, m=2, n=3 | |
| -- | Декремент (зменшення на одиницю):  Префіксна операція (--х) зменшує операнд на 1 до його використання.  Постфіксна операція (х--) зменшує операнд на 1 після його використання. | |
| sizeof | Обчислення розміру (в байтах) об’єкта того типу, який має операнд. Має дві форми:  1) sizeof (вираз);  sizeof(1.0); // Результат - 8,  Дійсні константи за замовчуванням мають тип double;  2) sizeof (тип)  sizeof (char); // Результат – 1. | |
| **Бінарні операції** | | |
| *Арифметичні операції* | | |
| + | Бінарний плюс (додавання арифметичних операндів) | |
| - | Бінарний мінус (віднімання арифметичних операндів) | |
| *Мультиплікативні* | | |
| \* | Добуток операндів арифметичного типу | |
| Операція | Короткий опис | |
| / | Ділення операндів арифметичного типу (якщо операнди цілочисельні, абсолютне значення результату заокруглюється до цілого, тобто 20/3 дорівнює 6) | |
| % | Одержання залишку від ділення цілочисельних операндів (13%4 = 1) | |
| *Операції зсуву (визначені лише для цілочисельних операндів)* | | |
| << | Зсув вліво бітового представлення значення лівого цілочисельного операнда на кількість розрядів, рівну значенню правого операнда (4<<2 дорівнює 16, тобто код 4 100, а звільнені розряду обнуляються, 10000 – код 16) | |
| >> | Зсув вправо бітового представлення значення правого цілочисельного операнда на кількість розрядів, рівну значенню правого операнда | |
| *Порозрядні операції* | | |
| & | Порозрядна кон’юнкція (І) бітових представлень значень цілочисельних операндів | |
| | | Порозрядна диз’юнкція (АБО) бітових представлень значень цілочисельних операндів | |
| ^ | Порозрядне виключне АБО бітових представлень значень цілочисельних операндів | |
| *Операції порівняння* | | |
| < | Менше, ніж | |
| > | Більше, ніж | |
| <= | Менше або рівне | |
| >= | Більше або рівне | |
| = = | Рівне | |
| != | Не рівне | |
| Операція | Короткий опис | |
| *Логічні бінарні операції* | | |
| && | Кон’юнкція (І) цілочисельних операндів або відношень, цілочисельний результат (0) або (1) | |
| || | Диз’юнкція (АБО) цілочисельних операндів або відношень, цілочисельний результат (0) або (1) (умова 0<x<1 мовою С++ записується як 0<x && x<1) | |
| Тернарна операція | | |
| *Умовна операція* | | |
| ? : | Вираз1 ? Вираз2 : Вираз3;  Першим вираховується значення Виразу1. Якщо воно істинне, тоді обчислюється значенняВиразу2, яке стає результатом. Якщо при обчисленніВиразу1одержуємо 0, тоді в якості результату береться значенняВиразу3.  Наприклад:  х<0 ? -x : x; //обчислюється абсолютна величина. | |

Пріоритет виконання операцій

Таблиця 7

|  |  |  |
| --- | --- | --- |
| Ранг | Операції | Напрямок виконання |
| 1 | () (виклик функції), [], ->, "." | >>> |
| 2 | !, ~, +, - (унарні), ++, --, \*, (тип), sizeof, (new,delete – Сі++) | <<< |
| 3 | .\* , ->\* - Сі++ | >>> |
| 4 | \*, /, % (бінарні) | >>> |
| 5 | +, - (бінарні) | >>> |
| 6 | <<, >> | >>> |
| 7 | <, <=, =>, > | >>> |
| 8 | ==, != | >>> |
| 9 | & (порозрядна) | >>> |
| 10 | ^ | >>> |
| 11 | | (порозрядна) | >>> |
| 12 | && (логічна) | >>> |
| 13 | || (логічна) | >>> |
| 14 | ?: (тернарна) | <<< |
| 15 | =, +=, -=, \*=, /=, %=, &=, ^=, |=, <<=,>>= | <<< |
| 16 | "," (кома) | >>> |

Основні математичні функції мови С/С++, опис яких міститься у файлі **<math.h>,**наведені у таблиці 8.

**Таблиця 8**

|  |  |  |  |
| --- | --- | --- | --- |
| Математичний запис | Функція | Пояснення | Приклад |
| arccos(x) | acos | Повертає арккосинус кута, рівного х радіан | acos(x); |
| arcsin(x) | asin | Повертає арксинус аргументу х в радіанах | asin(x); |
| arctg (x) | atan | Повертає арктангенс аргументу х в радіанах | atan(x); |
| аrctg(x/у) | atan2 | Повертає арктангенс відношення параметрів х та у в радіанах | atan2(x, y); |
| ]x[ | ceil | Заокруглює дійсне значення х до найближчого більшого цілого і повертає його як дійсне | ceil(x); |
| cos(x) | cos | Повертає косинус кута, рівного х радіан | cos(x); |
| ch(x) | cosh | Повертає гіперболічний косинус аргументу, рівного х радіан | cosh(x); |
| ex | exp | Повертає результат піднесення числа е до степені х | exp(x); |
| |x| | fabs | Повертає модуль дійсного числа х | fabs(x); |
| [x] | floor | Заокруглює дійсне число до найближчого меншого числа і повертає результат як дійсний | floor(x); |
| x mod y | fmod | Повертає залишок ділення х на у. Аналогічна операції %, але працює з дійсними числами | fmod(x, y); |
| ln (x) | log | Повертає значення натурального логарифму х | log(x); |
| lg (x) | log10 | Повертає значення десяткового логарифму х | log10(x); |
| xy | pow | Вираховує значення числа х у степені у | pow(x, y); |
| sin(x) | sin | Повертає синус кута, рівного х радіан | sin(x); |
| sh(x) | sinh | Повертає гіперболічний синус кута, рівного х радіан | sinh(x); |
|  | sqrt | Визначає корінь квадратний числа х | sqrt(x); |
| tg (x) | tan | Повертає тангенс кута, рівного х радіан | tan(x); |
| tgh(x) | tanh | Повертає гіперболічний тангенс кута, рівного х радіан | tanh(x); |

Операції присвоювання

Таблиця 9

|  |  |  |
| --- | --- | --- |
| Операція | Пояснення | Приклад |
| = | Присвоїти значення виразу-операнду з правої частини операнду лівої частини | Р=10.5-3\*х |
| \*= | Присвоїти операнду лівої частини добуток значень обох операндів | Р\*=2 еквівалентно Р=Р\*2 |
| /= | Присвоїти операнду лівої частини результат від ділення значення лівого операнда на значення правого | Р/=(2.2-х) еквівалентно  Р=Р/(2.2-х) |
| %= | Присвоїти лівому операнду залишок від ділення цілочисельного значення лівого операнда на цілочисельне значення правого операнда | Р%=3 еквівалентно Р=Р%3 |
| += | Присвоїти операнду лівої частини суму значень обох операндів | А+=В еквівалентно А=А+В |
| -= | Присвоїти операнду лівої частини різницю значень лівого і правого операндів | Х-=3.4-у еквівалентно Х=Х-(3.4-у) |

Приклади:

#include<stdio.h>

#include<math.h>

int main(){

double x, y, z;

scanf(“%f %f ”,&x,&y);

z = exp(x)\*cos(y);

printf(“z=%lf”,z);

}

## Дійсні типи даних

В наступній таблиці приведено дійсні типи даних на Сі:

Таблиця 10

|  |  |  |  |
| --- | --- | --- | --- |
| Тип | Розмір типу | Межі значень | Точність |
| float | 4 байти | 1.2E-38 - 3.4E+38 | 6 десяткових знаків |
| double | 8 байтів | 2.3E-308 - 1.7E+308 | 15 десяткових знаків |
| long double | 10 байтів | 3.4E-4932 - 1.1E+4932 | 19 десяткових знаків |

В заголовочному файлі float.h визначені точність та інші деталі представлення дійсних типів та їхніх значень:

#include <stdio.h>

#include <float.h>

int main() {

printf("Storage size for float : %lld \n", sizeof(float));

printf("Minimum float positive value: %E\n", FLT\_MIN );

printf("Maximum float positive value: %E\n", FLT\_MAX );

printf("Precision value: %d\n", FLT\_DIG );

return 0;

}

На Linux наприклад буде наступний результат:

Storage size for float : 4

Minimum float positive value: 1.175494E-38

Maximum float positive value: 3.402823E+38

Precision value: 6

### Константи дійсних типів

Константи дійсних типів мають цілу частину, дробову частину та можуть мати експоненту. Вони можуть бути представлені як в десятковому так і експоненційному (науковому) форматі.

При представленні в десятковій формі потрібно записати цілу частину, десяткову крапку та дробову частину, а потім для типу float можна записати в кінці літеру(суфікс) f або F. Для типу long double в кінці додається суфікс L.

Приклади:

float X1 = 123.567;

float X2 = 3.576f;

double X3 =0.2342;

long double x4 =33.56L;

Для представлення в експоненційному (науковому) форматі записується спочатку мантиса таким же чином як і десяткове дійсне число, потім літера e або E і після чього значення експоненти як ціле число, можливо зі знаком.

Приклади:

double y1 = 3.14159e0;

double y2 = 314159E-5;

float y3 = 0.314e-1f;

long double y4 = 5.46e235L;

Наступні варіанти запису дійсних чисел некоректні:

510E /\* Illegal: incomplete exponent \*/

210f /\* Illegal: no decimal or exponent \*/

.e55 /\* Illegal: missing integer or fraction \*/

Для введення дійсного числа можна використати наступні варіанти викликів функцій scanf:

float f1,f2,f3;

scanf(“%f”, &f1); // введення в десятковому форматі

scanf(“%e”, &f2); // введення в науковому форматі з маленькою e

scanf(“%E”, &f3); // введення в науковому форматі з великою E

double d1,d2,d3;

scanf(“%lf”, &d1); // введення в десятковому форматі

scanf(“%le”, &d2); // введення в науковому форматі з маленькою e

scanf(“%lE”, &d3); // введення в науковому форматі з великою E

long double r1, r2,r3;

scanf(“%Lf”, &r1); scanf(“%llf”, &r1); // введення в десятковому форматі

scanf(“%Le”, &r2); scanf(“%lle”, &r2); // введення в науковому форматі з маленькою e

scanf(“%LE”, &r3); scanf(“%llE”, &r3); ); // введення в науковому форматі з великою E

Для виведення дійсного числа можна використати наступні варіанти викликів функцій printf:

float f1,f2,f3;

printf(“%f”, f1); // виведення в десятковому форматі

printf(“%e”, &f2); // введення в науковому форматі з маленькою e

printf(“%E”, &f3); // введення в науковому форматі з великою E

printf(“%g”, &f3); // введення в десятковому форматі з найменшою кількістю знаків

double d1,d2,d3;

printf(“%lf”, d1); // виведення в десятковому форматі

printf(“%le”, d2); // введення в науковому форматі з маленькою e

printf(“%lE”, d3); // введення в науковому форматі з великою E

printf(“%g”, d1); // введення в десятковому форматі з найменшою кількістю знаків

long double r1, r2,r3;

printf(“%Lf”, r1); printf(“%llf”, r1); // виведення в десятковому форматі

printf(“%Le”, r2); printf(“%lle”, r2); // введення в науковому форматі з маленькою e

printf(“%LE”, r3); printf(“%llf”, r1); // введення в науковому форматі з великою E

printf(“%g”, r1); // введення в десятковому форматі з найменшою кількістю знаків

## Література

1. Об'єктно-орієнтоване програмування мовою C++ / Ю.І. Грицюк, Т.Є. Рак
2. Програмування мовою C++ / Ю.І. Грицюк, Т.Є. Рак
3. Алгоритмічні мови та основи програмування: мова C / В.Ю. Вінник
4. С++. Основи програмування. Теорія та практика / О.Г. Трофименко
5. Ю.А.Бєлов, Т.О.Карнаух, Ю.В.Коваль, А.Б. Ставровський. Вступ до програмування мовою С++. Організація обчислень. Навчальний посібник
6. <http://www.cplusplus.com/>
7. <https://purecodecpp.com/uk/>
8. <http://cpp-info-ua.blogspot.com/>
9. ANSI 89 – American National Standards Institute, American National Standard for Information Systems Programming Language C, 1989.
10. Kernighan 78 – B. W. Kernighan and D. M. Ritchie, The C Programming Language, Prentice-Hall: Englewood Cliffs, NJ, 1978. Second edition, 1988.
11. Thinking 90 – C\* Programming Guide, Thinking Machines Corp. Cambridge Mass., 1990.
12. Річі К. Мова програмування Сі
13. Керниган, Б. Язык программирования Си. Задачи по языку Си / Б. Керниган, Д. Ритчи, А. Фьюэр. – М. : Финансы и статистика, 1985. – 279 c.
14. С у задачах і прикладах : навчальний посібник із дисципліни "Інформатика та програмування" / А.П. Креневич, О.В. Обвінцев. – К. : Видавничо-поліграфічний центр "Київський університет", 2011. – 208 с.
15. Уэйт М. Язык Си / М. Уэйт, С. Прата, Д. Мартин. – М. : Мир, 1988. – 512 с