Лінійні програми на Сі

Програма на Сі - що це таке з точки зору стандарту Сі.
Приклад програми на Сі. З чого вона складається.
Компіляція програми на Сі.
Структурні частини програми на Сі: токени, крапки з комою, коментарі, ключові слова, пробіли.
Введення/виведення. С++ введення/виведення. Форматоване введення. Форматоване виведення. Бібліотека stdio.h
Визначення змінної в Сі. Декларація та ініціалізація.
Локалізація. Бібліотека locale.h
Використання математичної бібліотеки math.h
Дійсні типи даних.
Використання float.h

Програма на Сі: що це таке з точки зору стандарту Сі

Програма на С (С++) – це сукупність текстових файлів, зазвичай заголовочних та вхідних (header та source files), які містять декларації (declarations). Декларації — це визначення змінних та функцій. Файли, що складають програму компілюються для того, щоб утворити виконуваний файл, якщо С++ файли містять головну функцію (main function). Якщо її немає, то програма може бути перетворена в бібліотеку.

Програма на мові Сі (Сі++) побудована за допомогою спеціальних символів (на кшталт {,}, # і т.п.) та ключових слів (keywords). Інші слова можуть служити як ідентифікатори (identifiers). Крім того, програма може містити коментарі (comments) які ігноруються під час компіляції. Деякі символи можуть бути представлені в програмі за допомогою послідовностей символів (escape sequences).

Ідентифікатори (identifiers), на Сі можуть ідентифікувати сутносту (об'єкти) (objects), функції (functions), структури (struct), об'єднаня (union), перерахування (enumeration) та їх члени, типи, що визначені користувачем (typedef names), мітки (labels) та макроси(macros).

Сутності представлені деклараціями (declarations), які представляють їх іменами (names) та визначають їх властивості. Декларації визначають всі

властивості що потрібні для використання сутності як визначення (definitions). Програма має містити лише одне визначення будь-якої не підставляємої (non-inline) функції або змінної.

Визначення функцій складається з послідовності тверджень (statements), деякі з яких включають вирази (expressions), які визначають обчислення що повинна виконати програма.

Імена, що виникають в програмі співставленні з визначеннями, що зберігають їх за допомогою області дії імен (name lookup). Кожне ім'я діє лише в середині частини програми що зветься областю визначення (scope). Деякі імена можуть мати зв'язок (linkage), що визначає посилання до визначених для цього звязку сутностей, коли вони посилаються на ті самі сутності що з'являються в різних областях визначення або одиницях трансляції (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 (type), який може бути базовим (fundamental), составним (compound), або визначений користувачем (user-defined), повним (complete) або неповним (incomplete), і так далі.

Визначені об'єкти (declared objects) або визначені посилання (declared references) які не є не статичними членами даних (non-static data members) звуться змінними (*variables)*.

Програма на Сі: як це виглядає практично

Приклад програми на Сі

```
/*
Optional, but recommended: Documentation section
Необов'язкова але рекомендована частина (краще її писати англійською)
i.e. comments, that describe the program, like:
 Заголовочні коментарі, як наприклад:
First C-style program - Перша програма:
Обчислення сінуса
*/
#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",C); /* виведення значення змінної ‘C’ в науковому форматі  */
 }
```

Ці файли потрібно зберігати з розширенням “.c”, для того щоб система та компілятор зрозуміли що це файл для компіляції Сі-компілятором. Для того, щоб цю програму можна було відкомпілювати компілятором c++ 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 і так далі.

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, які звуться разом побудовою програми (building):

1. **Препроцесінг (Preprocessing):** файли C++ проекту оброблюються таким чином, що текст програми замінюється на зрозумілі компілятору 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 цей пункт вказують як частину процесу компіляції) – створює файл як набір асемблерних (машинних) інструкцій (object file) таким чином, що код стає таким, що можна переміщувати.

4. **Лінковка (Linking):** використовує 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 <опції, на зразок –o hello –l<бібліотека>>

Зокрема для компіляції файлу з першого лістингу потрібно запустити:

gcc hello1.c -lm

Опція lm потрібна для того, щоб програма використовувала математичну бібліотеку math.h.

Для компіляції Сі++:

g++ hello1.cpp

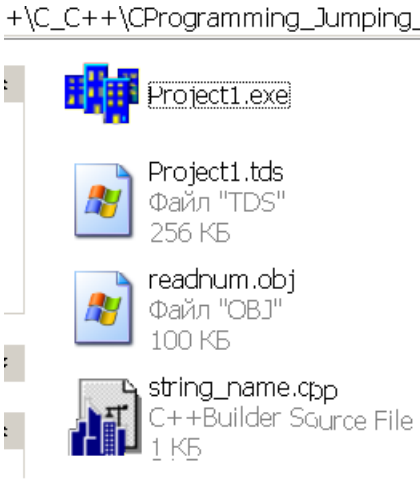
Результатом повинні бути виконуваний файл **hello1** та файл ***hello1.o*** – бібліотека (об'єктний файл)

Більш докладна команда, що показує проміжні етапи видасть результат

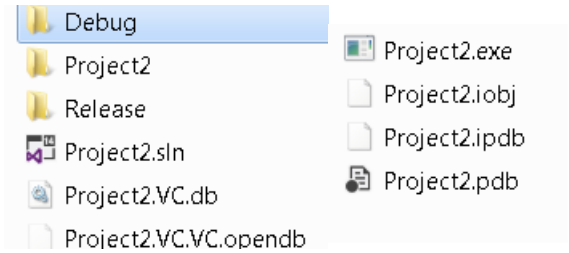
$gcc –Wall –save-temps filename.c –o filename

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 С

На мові С програма складається з різних токенів, які є або ключовими словами (keyword), ідентифікаторами (identifier), константами (constant), рядковими одиницями (string literal) або спецсимволами (symbol). Наприклад, printf("Hello, World! \n"); складається з наступних токенів

- printf — команда;
- (- спецсимвол(відкриваючи дужки);
- "Hello, World! \n" – рядкова константа;
-) - спецсимвол (закриваючи дужки);
- ; - спецсимвол (крапка з комою).

Крапки з комою

В мові С 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 С 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 самого стилю 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 С – 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).

Важливо памятати, що мова Сі не дозволяє символів @, $, та % всередині ідентифікаторів.

Крім того, Сі - **case-sensitive** програмна мова. Тобто, *Power* та *power* – два різні ідентифікатори в С. 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 С. Ці слова не можуть бути використані як ідентифікатори чи імена модулів.

Таблиця 1.1

auto	float	signed	_Alignas (з C11)
break	for	sizeof	_Alignof (з C11)
case	goto	static	_Atomic (з C11)
char	if	struct	_Bool (з C99)
const	inline (з C99)	switch	_Complex (з C99)
continue	int	typedef	_Generic (з C11)
default	long	union	_Imaginary (з C99)
do	register	unsigned	_Noreturn (з C11)
double	restrict (з C99)	void	_Static_assert (з C11)
else	return	volatile	_Thread_local (з C11)
enum	short	while	
extern			

Деякі ключові слова починаються з нижнього підкреслення:

Таблиця 1.2

Ключове слово	Використовується як	Визначено в
_Alignas (з C11)	alignas	stdalign.h
_Alignof (з C11)	alignof	stdalign.h
_Atomic (з C11)	atomic_bool, atomic_int, ...	stdatomic.h
_Bool (з C99)	bool	stdbool.h
_Complex (з C99)	complex	complex.h
_Generic (з C11)	(no macro)	
_Imaginary (з C99)	imaginary	complex.h
_Noreturn (з C11)	noreturn	stdnoreturn.h
_Static_assert (з C11)	static_assert	assert.h
_Thread_local (з C11)	thread_local	threads.h

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

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

if ifdef include

elif	ifndef	line
else	define	error
endif	undef	pragma
defined		

Цей специфічний токен розпізнається, якщо він знаходиться зовні макросів:

_Pragma(з C99)

Наступні два токена вважаються доповненнями до мови Сі:

asm
fortran

Пробіли в С

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

Пробілами (Whitespace) в Сі 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;
   int y = x*2+1;
   std::cout<<"y="<<y;
}
```

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>**:

printf (<керуючий рядок>, <список аргументів>);

Керуючий рядок береться у лапки і вказує компілятору вигляд інформації, що виводиться. Вона може містити специфікації перетворення та 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 1.3 наведено основні формати функції друку.

Таблиця 1.3

Формат	Тип інформації, що виводиться
%d	десяткове ціле число
% i	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;
int 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,"y.e");
printf ("x=%-8.4f s=%5d%8.2f ", x, s, x);
}
```

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

x=12.3450 s= -777 12.34.

Локалізація

Нажаль, коли ми спробуємо вивести інформацію за допомогою іншого алфавіту, зокрема українського, можливе виникнення такої ситуації, 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 Сі - бібліотека <locale.h> (clocale.h в Сі++) із функцією

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

Ця функція змінює поведінку стандартних функцій Сі 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"; іншим системам треба давати більш точні вказівки типу "uk_UA.cp1251" чи "en_US.utf8" (у форматі мова_країна.кодування).

Для Сі++ була повністю перероблена бібліотека cctype (ctype.h), і тепер є бібліотека locale, що містить всі важливі функції з ctype і ще купу різних функцій.

Крім того, в сучасних стандартах була додана підтримка юнікоду: широкі символи (wchar_t), широкі 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 1.4

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 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 проблему переповнення буферу (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_s("%d", &a);
scanf("%d", &b); // scanf_s("%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_s("%d", &a);
scanf(" %c", &b); // scanf_s("%d", &b);
/* пропустить всі пробіли перед %d, потім введе символ.*/
```

Примітка 2. Якщо у вас більш-менш сучасний компілятор(C11), то краще використовувати замість scanf функцію scanf_s. Якщо ж потрібно забазпечити крос-платформеність, то краще записати власний варіант введення, що використовує scanf_s у випадку сумісного з C11 компілятора, а в іншому випадку, наприклад, форматоване введення рядку з потрібними специфічними опціями.

Визначення змінної в Сі

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 Сі включаючи char, w_char, int, float, double, bool або будь-яким визначеним користувачем (user-defined) типом чи об'єктом; а список змінних **variable_list** може складатись з одного 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), тобто їм може бути присвоєно початкове значення під час 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 статичним типом видимості (static storage duration) ініціалізовані нулем або 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 змінною без запиту про те чому ця змінна дорівнює. Це знання зберігається лише під час компіляції, під час лінковки потрібне також визначення цієї змінної.

Різниця між присвоєнням та ініціалізацією С:

Таблиця 1.5

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 1.6).

Таблиця 1.6

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 (++x)* збільшує операнд на 1 до його використання. *Постфіксна операція (x++)* 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 (--x) зменшує операнд на 1 до його використання. Постфіксна операція (x--) 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 дорівює

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: `x<0 ? -x : x; //обчислюється абсолютна величина.`

Операції присвоювання

Таблиця 1.7

Операція	Пояснення	Приклад
=	Присвоїти значення виразу-операнду з правої частини операнду лівої частини	`P=10.5-3*x`
*=	Присвоїти операнду лівої частини добуток значень обох операндів	`P*=2` еквівалентно `P=P*2`
/=	Присвоїти операнду лівої частини результат від ділення значення лівого операнда на значення правого	`P/=(2.2-x)` еквівалентно `P=P/(2.2-x)`
%=	Присвоїти лівому операнду залишок від	`P%=3` еквівалентно

	ділення цілочисельного значення лівого операнда на цілочисельне значення правого операнда	P=P%3
+=	Присвоїти операнду лівої частини суму значень обох операндів	A+=B еквівалентно A=A+B
-=	Присвоїти операнду лівої частини різницю значень лівого і правого операндів	X-=3.4-y еквівалентно X=X-(3.4-y)

Порядок виконання операцій регулюється наступною таблицею пріорітетів виконання операцій.

Пріоритет виконання операцій

Таблиця 1.8

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

15	=, +=, -=, *=, /=, %=, &=, ^=, |=, <<=,>>=	<<<
16	"," (кома)	>>>

Використання математичної бібліотеки

Звичайно, не хотілося б використанням стандартних операцій та арифметичних дій над змінними обмежуватись. Навіть у далеких від математики галузях, а тим більш при програмуванні штучного інтелекту, задачах шифрування та контроля чи аналізу каналів зв'язку часто-густо треба використовувати відомі математичні функції. Звичайно, не варто (хоча інколи й корисно вміти) кожен раз реалізовувати ці функції, а скористатись бібліотеками, які взодять в стандарт мови Сі.

Для того, щоб використовувати бібліотеку математичних функцій потрібно підключити біблотеку math.h. Іншим варіантом є бібліотека tgmath.h, яку варто розглянути окремо. Після того, як ця бібліотека є включеною в файлі програми за допомогою директиви #include можна використовувати найбільш розповсюджені математичні функції в цій програмі.

Основні математичні функції мови C/C++, опис яких міститься у файлі **<math.h>,**наведені у таблиці 1.8.

Таблиця 1.9

Математичний запис	Функція	Пояснення	Приклад
arccos(x)	acos	Повертає арккосинус кута, рівного x радіан	acos(x);
arcsin(x)	asin	Повертає арксинус аргументу x в радіанах	asin(x);
arctg (x)	atan	Повертає арктангенс аргументу x в радіанах	atan(x);
arctg(x/y)	atan2	Повертає арктангенс відношення параметрів x та y в радіанах	atan2(x, y);
]x[	ceil	Заокруглює дійсне значення x до найближчого більшого цілого і повертає його як дійсне	ceil(x);

cos(x)	cos	Повертає косинус кута, рівного x радіан	cos(x);
ch(x)	cosh	Повертає гіперболічний косинус аргументу, рівного x радіан	cosh(x);
e^x	exp	Повертає результат піднесення числа e до степені x	exp(x);
|x|	fabs	Повертає модуль дійсного числа x	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 x на y. Аналогічна операції %, але працює з дійсними числами	fmod(x, y);
ln (x)	log	Повертає значення натурального логарифму x	log(x);
lg (x)	log10	Повертає значення десяткового логарифму x	log10(x);
x^y	pow	Вираховує значення числа x у степені y	pow(x, y);
sin(x)	sin	Повертає синус кута, рівного x радіан	sin(x);
sh(x)	sinh	Повертає гіперболічний синус кута, рівного x радіан	sinh(x);
$\sqrt{x}$	sqrt	Визначає корінь квадратний числа x	sqrt(x);
tg (x)	tan	Повертає тангенс кута, рівного x радіан	tan(x);
tgh(x)	tanh	Повертає гіперболічний тангенс кута, рівного x радіан	tanh(x);

Приклади:

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

int main(){
   double x, y, z;
   scanf("%lf %lf",&x,&y);
```

```
    z = exp(x)*cos(y);
    printf("z=%lf",z);
}
```

Для компіляції файла з математичною бібліотекою деякі компілятори вимагають вказання що компілювати потрібно з цією бібліотекою, наприклад, для gcc під Linux:

>>>gcc examle.c –lm

або якщо потрібно вказати імя вихідного виконувано файлу:

>>> gcc examle.c –o example –lm.

Дійсні типи даних

В наступній таблиці приведено дійсні типи даних на Сі:

Таблиця 1.10

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 64-бітному 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, згідно специфікації IEEE-754. 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        /* Помилка- Illegal: incomplete exponent */
210f        /* Помилка- Illegal: no decimal or exponent */
.e55        /* Помилка- Illegal: missing integer or fraction */
```

Введення/виведення дійсних чисел

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);   // введення в десятковому форматі
scanf("%Le", &r2); // введення в науковому форматі з маленькою e
scanf("%LE", &r3); // введення в науковому форматі з великою E
```

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

```
printf("%f", f1);  // виведення в десятковому форматі
printf("%le", f2);  // введення в науковому форматі з маленькою e
printf("%lE", f3);  // введення в науковому форматі з великою E
printf("%g", f3);  // введення в десятковому форматі з найменшою кількістю знаків
```

```
printf("%lf", d1);  // виведення в десятковому форматі
printf("%le", d2);  // введення в науковому форматі з маленькою е
printf("%lE", d3);  // введення в науковому форматі з великою Е
printf("%g", d1);  // введення в десятковому форматі з найменшою кількістю знаків

printf("%Lf", r1); // виведення в десятковому форматі
printf("%Le", r2); // введення в науковому форматі з маленькою е
printf("%LE", r3); // введення в науковому форматі з великою Е
printf("%Lg", r1);  // введення в десятковому форматі з найменшою кількістю знаків
```

Помітимо, що при введенні дійсних чисел важливо не переплутати специфікатори, програма з прикладу наступного вигляду може підрахувати невірне значення:

```
#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);
}
```

На Сі++ можна не враховувати такі нюанси:

```
#include<iostream>
#include<math>

int main(){
  double x, y, z;
  std::cin>>x>>y;
  z = exp(x)*cos(y);
  std::cout<<z;
  printf("z=%lf",z); // в Сі++ можна використовувати С функції вводу-виводу
}
```

Література

1. Об'єктно-орієнтоване програмування мовою С++ / Ю.І. Грицюк, Т.Є. Рак
2. Програмування мовою С++ / Ю.І. Грицюк, Т.Є. Рак
3. Алгоритмічні мови та основи програмування: мова С / 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 с.
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
16. https://en.cppreference.com/w/c/io/fscanf
17. http://www.c-cpp.ru/content/scanf
18. http://www.cplusplus.com/reference/cstdio/scanf/