МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Київський національний університет будівництва і архітектури

Г.В. КРАСОВСЬКА

АЛГОРИТМІЗАЦІЯ ТА ПРОГРАМУВАННЯ

Основи програмування мовою Сі

Конспект лекцій

для студентів, які навчаються за напрямом
підготовки 6.050101 «Комп'ютерні науки»

Київ 2015

УДК 681.3.06
ББК 32.973-01
К78

Укладач Г.В. Красовська, канд. техн. наук, доцент

Рецензент І.М. Доманецька, канд. техн. наук, доцент

Затверджено на засіданні вченої ради факультету автоматизації та інформаційних технологій протокол №3 від 26 листопада 2014 року.

Красовська Г.В.

К78 Алгоритмізація та програмування. Основи програмування мовою Сі: конспект лекцій / Г.В. Красовська. – К.: КНУБА, 2015. – 52 с.

Викладено теми другого модуля дисципліни «Алгоритмізація та програмування», розглянуто основні поняття мови програмування Сі, структура програми, типи даних, опис констант та змінних, оператори, опис функцій, консольне та файлове введення/виведення даних, масиви даних, структуровані типи даних, покажчики та динамічний розподіл пам’яті.

Призначено для студентів, що навчаються за напрямом підготовки 6.050101 «Комп’ютерні науки».

УДК 681.3.06
ББК 32.973-01

ЗМІСТ

Вступ

Мову програмування Сі розробив на початку 70-х років минулого століття Деніс Рітчі (Dennis M. Ritchie), співробітник фірми Bell Telephone Laboratories (США) [4].

У передмові до першого видання книги «Мова програмування Сі», яка вийшла в США в 1978 р. (вперше перекладена російською в 1985 р.), він та Брайан Керниган писали: «Сі – це універсальна мова програмування з компактним способом запису виразів, сучасними механізмами управління структурами даних і різноманітним набором операторів. Сі не є ні мовою "дуже високого рівня", ні "великою" мовою, вона не розрахована і на якусь конкретну область застосування. Однак, завдяки широким можливостям і універсальності для вирішення багатьох завдань вона зручніша і ефективніша, ніж імовірно більш потужні мови. …І операційна система, і Сі-компілятор, і, по суті, всі прикладні програми системи UNIX (включаючи і ті, які використовувалися для підготовки тексту цієї книги) написані на Сі» [2].

Пройшло майже 40 років, а «мова програмування Сі, в якій поєднується потужність і гнучкість універсальних мов програмування з високою ефективністю виконавчого коду та можливістю безпосереднього доступу до апаратних ресурсів комп’ютера, є однією з фундаментальних і найбільш уживаних мов проблемно-орієнтованого та системного програмування. Тому глибоке знання і практичне володіння інструментальними засобами мови Сі обов’язкове для фахівців з програмного забезпечення комп’ютерів, комп’ютерних інформаційних технологій, систем автоматизованого керування та проектування, комп’ютерної інженерії, а також усіх, хто пов’язує свою діяльність з комп’ютером і бажає опанувати науку програмування» [4]

Зміст даного конспекту лекцій відповідає робочій навчальній програмі дисципліни «Алгоритмізація та програмування», що викладається на факультеті автоматизацій та інформаційних технологій Київського національного університету будівництва та архітектури для студентів першого курсу напрямку підготовки 6.050101 «Комп’ютерні науки». До матеріалів даного конспекту лекцій увійшли теми лекційного матеріалу другого модуля дисципліни «Алгоритмізація та програмування» (тематика першого модуля відображена в [5]).

Лекція 1. Основні поняття мови програмування Сі

Алфавіт мови

Алфавіт мови Сі складається з:

- великих та малих літер латинського алфавіту: ‘A’..’Z’, ‘a’..’z’;
- десяткових цифр 0, 1, 2, … , 9 та шістнадцяткових цифр 0, … , 9, A, B, …F;
- спеціальних символів: ~ ! @ # $ % ^ & * ( ) - + = { } [ ] : ; “ ‘ < > ? , . / | \ та символу підкреслення _ ;
- ескейп-послідовностей – спеціальних символів, які призначені для представлення неграфічних символів або символів, що важко представити, враховуючи семантику мови Сі. Зазвичай вони використовуються для управління курсором на екрані, наприклад таких дій як табулювання, переведення курсору на новий рядок т.і. [2]

Ескейп-послідовність складається зі зворотного слеша \, за яким слідує або буква, або цифра, або знак пунктуації. Ескейп-послідовність вважається за один символ і береться в одинарні лапки. Наприклад, ‘\n’, ‘\a’ тощо. Список ескейп-послідовністей наведено в табл. 1.

Таблиця 1

Ескейп-послідовності мови С [2]

Позначення	Опис
\ a	звуковий сигнал з динаміка
\ b	повернення курсору на крок назад
\ f	переведення сторінки
\ n	переведення курсору на новий рядок
\ r	повернення курсору на початок рядка
\ t	горизонтальне табулювання
\ v	вертикальне табулювання
\ \	виведення зворотного слешу
\ ?	виведення знака питання
\ ‘	виведення одинарної лапки
\ “	виведення подвійної лапки
\ 000	вісімковий код (цифри від 0 до 7)
\ xhh	шістнадцятковий код

Ключові слова

Ключові слова – зарезервовані ідентифікатори, що використовуються під час написання програми. Змінювати призначення ключових слів неможна. Основні ключові слова мови Сі такі:

auto	***double***	***int***	***struct***
break	***else***	***long***	***switch***
case	***enum***	***register***	***typedef***
char	***extern***	***return***	***union***
const	***float***	***short***	***unsigned***
continue	***for***	***signed***	***void***
default	***goto***	***sizeof***	***volatile***
do	***if***	***static***	***while***

Коментарі

Коментарі – частина тексту програми, що слугує для пояснення операторів програми або алгоритму її роботи. Коментарі не впливають на виконання програми. Текст коментарів записують у дужках:

```
/* це коментар */
```

Коментарі записуються в програмі на новому рядку або наприкінці рядка після оператора. Наприклад:

```
/* фрагмент програми мови Сі */
int x = 0; /* оголошення та ініціалізація змінної */
/* надання змінній нового значення */
x = 25 * x;
```

Структура програми мовою Сі

Програма мовою Сі умовно може бути розподілена на три частини:

- блок препроцесування;
- оголошення змінних на зовнішньому рівні та опис функцій;
- головна функція.

Кожна з цих частин може бути відсутня. Але, якщо є блок препроцесування, то він розташовується на початку програми. Після блоку препроцесуваня послідовність і наявність розділів програми може бути змінена.

Узагальнена структура програми подана на рис. 1.

Блок препроцесування обробляється і виконується до початку компіляції основної програми і складається з директив. Директиви починаються зі знака #. Виконання директив препроцесування в більшості випадків призводить до модифікації тексту програми. Наприклад, директива підключення бібліотек:

#include < назва файлу >
або
#include “ назва файлу “

```
/* блок препроцесування */
#include < назва файлу 1>
#include “ назва файлу 2 “
/* інші директиви препроцесування */
```

```
/* опис змінних на зовнішньому рівні */
   int a;
   float b=0.5;
тип_результату ім’я_функції (список параметрів)
                                                    /* заголовок функції */
{ /* початок функції */
   /* опис змінних на внутрішньому рівні */
   /* тіло функції */
   return результат; /* повернення результату в точку виклику */
} /* кінець функції */

/* опис змінних на зовнішньому рівні */
…
/* опис інших функцій */
…
/* опис змінних на зовнішньому рівні */
…
```

```
/* головна функція */
int main (int argc, char* argv[])          /* заголовок головної функції */
{ /* початок головної функції */
      /* опис змінних на внутрішньому рівні */
      /* тіло головної функції */
return 0;
} /* кінець головної функції */
```

Рис. 1. Узагальнена структура Сі-програми

означає, що до програми необхідно приєднати текст програми із зазначеного після неї файлу. Файли, які приєднуються директивою include мають розширення .h і називаються файлами-заголовками (header-файлами). У них описуються константи, змінні або функції стандартної бібліотеки мови Сі або створені програмістом. Усі стандартні бібліотеки мови Сі розташовуються в службовому директорії INCLUDE.

Отже, якщо необхідно звернутися до файлу-заголовку, що знаходиться в службовому директорії (каталозі) INCLUDE, його ім'я вказується в < > . Наприклад:

#include <math.h>

Якщо файл розташовується в іншому директорії, тоді його ім'я вказується в " ", в яких також може бути записано шлях до файлу:

#include "c:\\stud\\iust\\myfile.h"

Інші директиви блоку препроцесування поступово будуть розглядатися в наступних темах.

Будь-яка програма призначена для обробки даних, які подаються в програмі як змінні або константи. Опис змінних може здійснюватися:

- на внутрішньому рівні – в тілі будь-якої функції програми;
- на зовнішньому – поза будь-якими функціями.

У термінах мови Сі головна програма називається головною функцією, яка може мати:

- тип результату int (повертає цілочисельне значення) або void (не повертає жодного значення);
- зарезервоване ім'я main;
- список параметрів int argc, char* argv[], які використовуються для доступу до елементів командного рядка, що задаються під час запуску програми на виконання (argc – кількість аргументів, argv – посилання на список аргументів). Список параметрів може бути порожнім, тоді він позначається як void.

У функціях рядок **return** призначений для повернення визначеного результату в зовнішню функцію, яка викликала дану. У випадку головної функції результат повертається операційній системі, під управлінням якої дана програма була запущена на виконання. Повернення з функції main нуля (*return 0;*) означає завершення програми без помилок.

Організація Сі-програм у складі декількох файлів

Коли розробляється велика за обсягом програма доцільно тексти програм зберігати в декількох файлах. У Сі-програмах є різні способи організації таких програм. Найпростіший і найпоширеніший з них – опис файлів-заголовків (h-файлів), які призначені для централізованого опису констант, змінних, типів та заголовків функцій, які будуть спільно використовуватися всіма файлами, з яких складається Сі-програма.

Фактично h-файл – окремий файл, який містить декларації для декількох файлів-реалізації. Узагальнена структура Сі-програми, що складається з декількох файлів, подана на рис. 2.

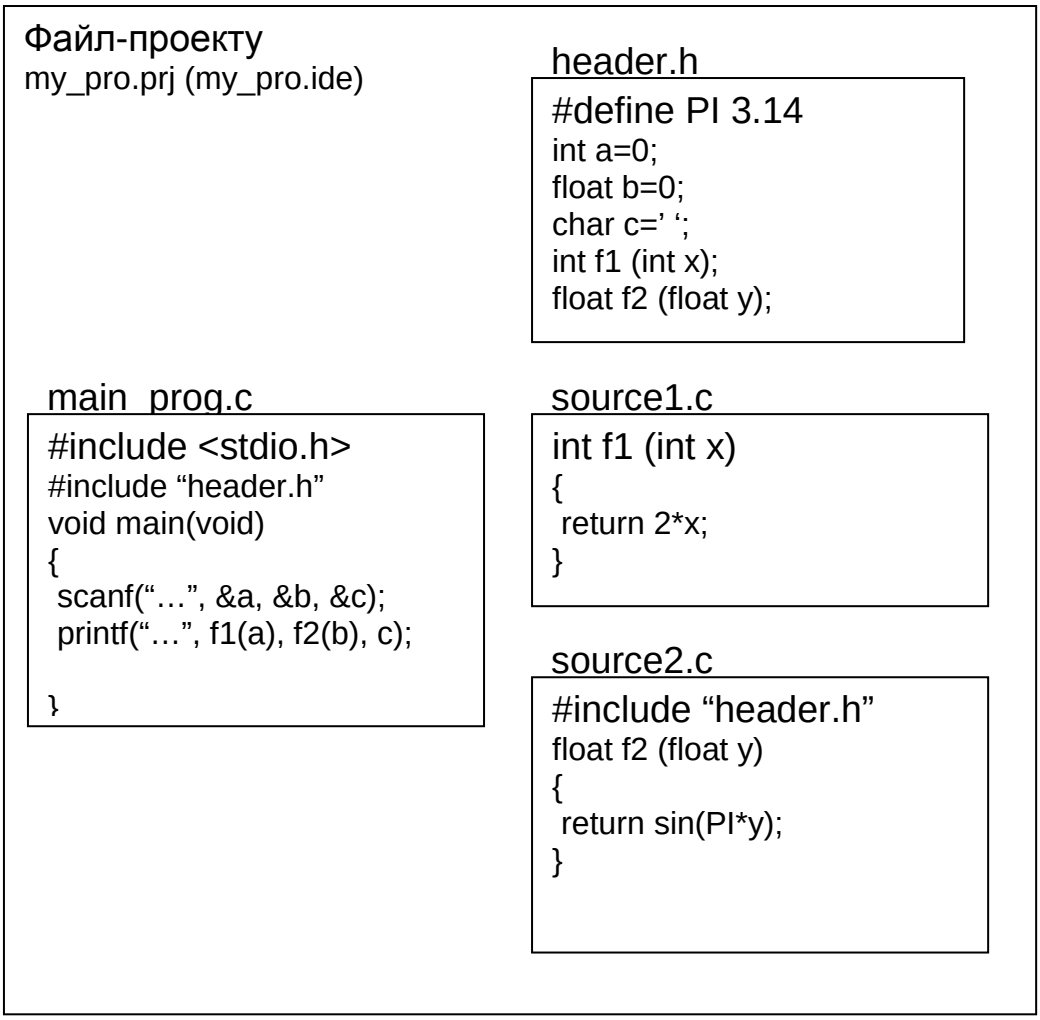


Рис. 2. Загальна структура СІ-програми [2]

Окрім файлів з текстами програм та файлів-заголовків на рисунку поданий ще один файл з розширенням .prj (якщо програма розробляється в середовищі TurboC) або .ide (якщо програма розробляється в середовищі BorlandC++). Цей файл називається *файлом проекту* та об'єднує всі частини Сі-програми в єдине ціле.

Контрольні запитання

1. З чого складається алфавіт мови Сі?
2. Що таке ескейп-послідовність? Як вона записується?
3. Які ескейп-послідовності існують у мові Сі?
4. Для чого і як у програмі використовуються коментарі?
5. Що таке ключові слова мови програмування? Які ключові слова мови Сі Ви знаєте?
6. Яку структуру має Сі-програма, з яких частин (блоків) вона може складатися?
7. Призначення розділу препроцесування.

8. Які команди препроцесування ви знаєте? Наведіть приклади.
9. Що таке бібліотека? Як бібліотека підключається до Сі-програми? Коли під час підключення назва бібліотеки вказується в < > , а коли в “” ?
10. Що означає опис змінних на зовнішньому та внутрішньому рівні?
11. Як описується функція в Сі-програмі?
12. Яка функція Сі-програми вважається головною, в чому її особливості?
13. Чи може програма мовою Сі складатися з декількох файлів?
14. Призначення файлу-заголовка (h-файлу).
15. Призначення файлу-проекту.

Лекція 2. Типи даних мови Сі

Дані, що в процесі роботи програми можуть змінювати своє значення називаються *змінними*. Усі дані зберігаються в пам’яті, тому на початку роботи програми для їх збереження виділяється певна кількість байт пам’яті. Обсяг пам’яті, який виділяється для збереження кожного значення визначає і діапазон, в якому може змінюватися це значення. Тому всі змінні в програмі повинні бути описані певним типом даних до початку їх використання. У табл. 2 наведено перелік простих типів даних мови Сі (треба зауважити, що розмір та діапазон у різних версіях мови може відрізнятися).

Таблиця 2

Прості типи даних мови Сі

Тип даних	***Розмір (байт)***	***Діапазон***
char	1	-128 ...128
signed char	1	-128…128
unsigned char	1	0…255
short int	2	-32768…32767
unsigned int	2	0…65535
int	2	-32768…32767
long	4	-2147483648…2147483647
unsigned long	4	0…4294967295
float	4	$\pm 3.4 \cdot 10^{-38}$ … $\pm 3.4 \cdot 10^{38}$
double	8	$\pm 1.7 \cdot 10^{-308}$ … $\pm 1.7 \cdot 10^{308}$
long double	10	$\pm 3.4 \cdot 10^{-4932}$… $\pm 3.4 \cdot 10^{4932}$

Діапазон представлення змінних цілочисельних типів визначається константами, що зберігаються в бібліотеці <limits.h>. Значення цих констант залежить від компілятора, з яким ми працюємо. Так, наприклад,

для компілятора TurboC діапазон представлення цілих –32768 ... 32 767, а в BorlandC++ цілі обробляються з діапазоном long (рис. 3).

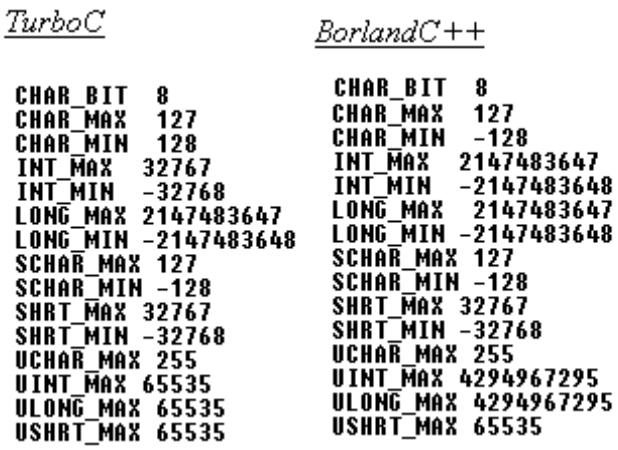


Рис. 3. Константи бібліотеки <limits.h>

Опис змінних

Для того щоб описати змінну певним типом необхідно спочатку вказати тип даних, потім ім'я змінної (або перелік змінних), що описуються цим типом. У разі необхідності під час опису змінної можна присвоїти їй початкове значення (ініціалізувати). Опис змінної разом з її ініціалізацією називається визначенням змінної:

<тип змінної> <ім'я змінної> = <початкове значення>;

```
int a;          /* опис змінної */
long k=6789;    /* визначення змінної */
float b, c, d=2.7;   /* опис змінних b, c, визначення змінної d */
```

Якщо одним типом описується декілька змінних, вони вказуються через кому, список змінних одного типу закінчується крапкою з комою (;).

Примітка!!! Компілятор мови Сі розрізняє літери верхнього та нижнього регістру. Тому, коли обирають ідентифікатори та під час набору програми, треба пам'ятати, що імена *a* та *A* різні.

Як було сказано вище опис змінних у Сі-програмі може здійснюватися на зовнішньому (поза функціями) або на внутрішньому рівнях (всередині функцій):

```
   int x;        /* на зовнішньому рівні */
main()
{
   int y;        /* на внутрішньому рівні */
}
```

Опис констант

Термін константа, у загальному випадку, відноситься до значення, яке не може бути змінено. У мові Сі константи можуть бути цілими, дійсними, символьними та рядковими (табл. 3).

Таблиця 3

Константи мови Сі

Константа	Формат	Приклади
Ціла	Десятковий – послідовність цифр, що не починається з нуля	23, 89, 1000
	Вісімковий – починається з нуля, за яким послідовність вісімкових цифр	023, 076, 01
	Шістнадцятковий – починається з 0x або 0X, за якими послідовність шістнадцяткових цифр	0x1, 0xAB, 0Xf
Дійсна	Десятковий з фіксованою крапкою – [цифри].[цифри]	1.23, 0.4, 12.76
	Десятковий з плаваючою крапкою (експоненціальний) – *[цифри] E/e [+/-] [цифри]*	1e-2, 5.45E+3, 7.98e1
Символьна	Один символ, що взятий в одинарні лапки (’)	’a’, ‘f’, ‘*’
Рядкова	Послідовність символів, що взяті в подвійні лапки (“”) – “символ символ символ …”	“abc”, “e2-e4” “Привіт!!!”

Опис нетипізованих констант у блоці препроцесування

Константи можуть бути описані в блоці препроцесування за допомогою директиви #define. Загальний опис має вигляд:

#define < ім’я константи > < значення >

```
#define A  10
#define PI  3.14
#define PLUS  ‘+’
#define STR “Рядок символів”
```

Треба звернути увагу, що імена констант прийнято набирати з прописних літер (літер верхнього регістру), але це не є обв’язковим. Для кожної константи задається окремий рядок #define.

Опис типізованих констант

Типізовані константи описуються так само як і змінні, але з використанням ключового слова const перед типом.

```
const int n=10;
const double r=0.001;
```

Різниця між двома способами опису констант полягає в тому, що константи з директиви #define (особливо числові константи) не є константами в нашому розумінні цього терміну. Як вже було сказано всі директиви блоку препроцесування призначені для модифікації тексту самої програми. Отже, до початку компіляції програми в тексті всі ідентифікатори, що описані в #define, будуть змінені на їх значення.

До препроцесування (“исходник”)	Після препроцесування (перероблений прогр. код)
#define A 12 main() { int x; x=A*4; }	main() { int x; x=12*4; }

Контрольні запитання[1]

1. Що таке ідентифікатор? Що таке стандартні визначені ідентифікатори?
2. Правила формування ідентифікаторів у програмі мовою Сі. Наведіть приклади вірних та помилкових ідентифікаторів.
3. Цілочисельні типиданих мови Сі. Який розмір в байтах має кожен тип, який діапазон значень?
4. Які з цілочисельних типів є знаковим цілими, які беззнаковими? Як в мові Сі явно задати знакове чи беззнакове ціле? Як це впливає на діапазон припустимих значень?
5. Дійсні типи даних мови Сі. Що таке мантиса, порядок та точність дійсного числа?
6. Символьний тип даних мови Сі. Як символи подаються в пам’яті комп’ютера? Що таке таблиця ASCII-кодів символів?
7. Чи однаковий діапазон значень для типів даних в різних середовищах програмування мови Сі?
8. Для чого використовується бібліотека <limits.h>?
9. Правила опису змінних у програмі мовою Сі. Наведіть приклади опису змінних різних типів.

[1] Під час підготовки відповідей на деякі запитання до цієї теми та надалі необхідно ознайомитися з матеріалами відповідних тем першої частини конспекту лекцій [5] та підручника [3].

10. Що таке оголошення, визначення та ініціалізація змінної? Наведіть приклади.
11. Як здійснюється опис змінних на зовнішньому та внутрішньому рівнях?
12. Наведіть два способи опису констант у програмі мовою Сі.
13. Наведіть приклади опису символьних констант.
14. Наведіть приклади опису чисельних констант.
15. Чому опис констант у блоці препроцесування називається макропідстановкою?

Лекція 3. Вирази та операції мови Сі

Мова Сі надає програмісту різноманітний набір операцій, за допомогою яких виконуються маніпуляції з даними. Коли вираз складається більше ніж з однієї операції, порядок їх виконання визначається згідно з пріоритетом (табл. 4). Оператори, що записані в одному рядку мають однаковий пріоритет і виконуються у виразі згідно з вказаним порядком (зліва направо чи справа наліво). Рядки табл. 4 впорядковані за зменшенням пріоритету операцій.

Таблиця 4

Пріоритет операцій мови Сі

Операції	Порядок виконання
( ), [ ], ->, .	зліва направо
!, ++, - -, +, -, *, &, (тип), sizeof	справа наліво
*, /, %	зліва направо
+, -	зліва направо
<, <=, >, >=	зліва направо
= =, ! =	зліва направо
&&	зліва направо
||	зліва направо
?:	справа наліво
=, +=, - =, *=, /=, %=,	справа наліво

Розглянемо докладніше наведені операції:

- ( ) – дужки або виклик функції (див. тему “Функції”);
- [ ] – індексування елементів масиву (див. тему “Масиви даних”);
- . (крапка) – розкриття структури (див. тему “Структури даних”);
- ! – логічне “ні”;
- ++ – інкремент (збільшення на 1), унарна операція;

- - – декремент (зменшення на 1), унарна операція;
- +, -, – унарний “плюс” та унарний “мінус”;
- *, & – розкриття посилання та отримання адреси (див. тему “Динамічний розподіл пам’яті”);
- (тип) – явне приведення типу;
- sizeof (< аргумент >) – визначення розміру вказаного аргументу в байтах.
- *, /, % – арифметичні бінарні операції добутку, ділення та визначення остачі від ділення;

Операція ділення мови Сі має дві реалізації.

1. Ділення операндів, один з яких (або обидва) дійсного типу. Виконується як звичайне ділення. Результат буде дійсного типу.
2. Ділення операндів, коли ***обидва цілочисельні***. Виконується цілочисельне ділення за правилами операції div Паскаль. ***Результат цілочисельний***.

- +, - – арифметичні бінарні операції додавання та віднімання;
- <, <=, >, >= – операції порівняння.
- = =, ! = – операції порівняння “рівно”, “нерівно”. Результатом порівняння на рівність є 0, якщо порівнювані величини рівні або якесь інше цілочисельне значення в протилежному випадку;
- || – логічне “або”;
- ?: – операція умови. Загальний синтаксис цієї операції такий:

(*< умова >****)?*** *< вираз 1>* ***:*** *< вираз 2>*

- Якщо умова істинна, то виконується < вираз 1>, інакше – < вираз 2 >.
- =, +=, - =, *=, /=, %= – операції присвоєння. Перша операція – просте присвоєння. Це бінарна операція, що виконується справа наліво.

Складене присвоєння

У випадках, коли праворуч від операції присвоєння стоїть арифметичний вираз, який виглядає таким чином:

< змінна1 > = <змінна1 > < операція > < вираз >,

можна це записати так:

< змінна1 > < операція > = < вираз >.

Наприклад, x=x+5 або x+=5, y=y*10 або y*=10, v=v/(a+b) або v/=(a+b).

Постфіксна та префіксна форма інкрементної та декрементної операцій

Порядок виконання операцій ++ та - - залежить від того, вказуються вони до операнда або після. Наприклад, змінна x=5, тоді після виконання

n=x++;

n буде дорівнювати 5, а x буде 6. Натомість від цього в записі:

n=++x;

n та x будуть дорівнювати 6.

Явне та неявне перетворення типів мови Сі

Неявне перетворення типів

1. При присвоєнні значення змінної одного типу змінній іншого типу.

В операціях присвоєння тип значення, яке присвоюється, приводиться до типу змінної, що отримує це значення. Перетворення здійснюється, навіть якщо воно призводить до втрати інформації. Наприклад:

/* З втратою значення. Діапазон short від –128 до 127 */ int a=3200; short b; b = a;	/* Без втрати значення. Діапазон int -32768…32767, що більше за short */ int a; short b=120; a = b;

Втрата інформації виникає, коли тип з більшим діапазоном значень (або більшою точністю для дійсних) приводиться до типу з меншим діапазоном.

2. Під час обчислення математичних виразів, коли операнди різних типів.

float a,b;

int x;

b=a+x;

Відповідно до пріоритету операцій спочатку виконується додавання a та x, але постає питання, як додати значення різних типів, адже внутрішнє подання цілочисельних та дійсних типів суттєво розрізняється. Це питання вирішується приведенням операндів різних типів до одного без втрати інформації. Отже, значення x буде приведене до типу float, а вже потім буде виконуватися дія.

3. Обробка покажчиків (дивися відповідний розділ).

Покажчики на значення одного типу можуть бути приведені до покажчиків на значення іншого типу. Результат може бути невизначеним завдяки відмінностям у потребах до вирівнювання об'єктів різних типів і в розмірі пам'яті, що займають різні типи. Це не стосується покажчика на тип void. Покажчик на тип void може бути приведено до будь-якого типу.

Неприпустимо перетворення типів структура та об'єднання.

Явне перетворення типів

(< ім'я типу >) < операнд >

Операнд – це вираз, значення якого повинно бути приведено до вказаного типу. Перетворення здійснюється так само, як якщо б він присвоювався змінній типу < ім'я типу >.

Контрольні запитання

1. Що таке операція та операнд? Визначення унарної та бінарної операції.
2. Які арифметичні операції мови Сі?
3. У чому особливості операції ділення мови Сі? Як під час ділення двох цілих можна все ж таки отримати точний результат (результат дійсного типу)?
4. Які операції порівняння мови Сі? Якого типу результат виконання операцій порівняння?
5. Які логічні операції мови Сі? Наведіть таблиці істинності для логічних операцій.
6. Як в мові Сі об'єднати декілька операцій порівняння в один логічний вираз?
7. Правила запису математичних та логічних виразів у Сі - програмі. Як визначається послідовність виконання операцій у виразі? Як можна змінити в разі необхідності цю послідовність?
8. Інкрементна та декрементна операції мови Сі. На що може вплинути постфіксна і префіксна форми запису цих операцій?
9. Які правила використання умовної операції (? :)?
10. Як записується просте та складене присвоєння? В яких випадках застосовується складене присвоєння?
11. Що таке сумісність типів даних за присвоєнням? Наведіть приклади типів даних, що сумісні за присвоєнням.
12. Що таке перетворення типів? Коли відбувається неявне перетворення типів?
13. Як здійснити явне перетворення типів?

14. Математичні функції бібліотеки math.h.
15. Функції перевірки класу літер бібліотеки <ctype.h>: isalnum, isalpha, iscntrl, isdigit, islower, isupper та перетворення літер tolower, toupper.

Лекція 4. Введення/виведення даних бібліотеки < stdio.h >

Форматне введення / виведення

Виведення даних printf ()

Для вивиедення даних у програмах мовою Сі використовується функція ***printf(список параметрів)***. Параметри цієї функція розділені на дві частини: перша – форматний рядок, який описує дані, що виводяться на екран; друга частина – перелік змінних, значення яких виводиться. Загальний опис функції такий:

printf (**“** форматний рядок **“** , arg1, arg2, ...);

де arg1, arg2, ... – список змінних для виведення.

Список змінних для виведення може бути відсутній, тоді на екран виводиться тільки те, що задається в форматному рядку. За використання функції printf **наявність форматного рядка обов’язкова.**

Правила формування форматного рядка для printf ()

Форматний рядок може містити:

1. Зичайні літери, які копіюються на екран:
   printf(“Це повідомлення буде виведене на екран!!!”);
2. Ескейп-послідовності для управління курсором під час виведення даних:
   printf (“\n Виведення на екран з нового рядка !!!”);
3. Опис типу змінної, що виводиться:

%<прапорець><розмір поля виведення><літера – специфікатор типу >

<літера – специфікатор типу >

Для опису типу використовують літери, що наведені в табл. 5.

Таблиця 5

Опис типів в форматному рядку printf ()

Тип змінної	***Літери***	***Опис***
char	d	виводиться код символу
	c	виводиться символ
short	d	виводиться значення в діапазоні short
	hd	
int	d	виводиться знакове десяткове ціле
	i	
long	li	виводиться знакове довге ціле
float	f	виводиться дійсне з фіксованою крапкою зі знаком, після крапки виводиться 6 знаків або нулі. Наприклад: 2.123456 1.230000
	g, G	виводиться дійсне з фіксованою крапкою зі знаком, після крапки хвостові нулі відсікаються. Наприклад: 2.123456 1.23
	e, E	виводиться дійсне в форматі з плаваючою крапкою
double	lf	виводиться дійсне значення типу double. Перетворення здійснюється аналогічно до типу float.
	lg	
	le	
unsigned	u	виводяться беззнакові цілі
unsigned char	u	
unsigned short	u	
unsigned int	u	
unsigned long	lu	
long int	li	виводиться довге ціле
long double	lf	виводиться довге дійсне в форматі з фіксованою крапкою і хвостовими нулями

<*розмір поля виведення*> – необов'язкова частина. Для цілих типів задається одне число, що визначає ширину поля виведення. Для дійсних типів задається 2 числа: загальна ширина поля (враховуючи крапку) та кількість знаків у дробовій частині.

<*прапорець*> – необов'язкова частина. Може бути таким:

-	значення виводиться з лівого краю поля виведення (якщо не вказаний – до правого);
+	числове значення буде виводитися завжди зі знаком;
прогалина	якщо перша літера не знак, то числу передує прогалина;
0	число доповнюється нулями до всієї ширини поля виведення;
#	для f, g, e при виведенні завжди виводиться крапка; для g хвостові нулі не відкидаються

Наприклад, задаємо такі змінні:

int a=15;

float b=34.67;

Для того, щоб вивести їх значення на екран спочатку сформуємо форматний рядок: для цілого ***a*** формат описується %d; для дійсного ***b*** – %f. Отже, форматний рядок буде "%d%f". Виведення на екран запишеться так:

printf("%d%f", a, b);

Якщо додати розміри поля для виведення, отримаємо: printf ("%5d%7.3f", a, b).

Можна вивести на екран ще додаткові пояснення до значення змінних:

printf ("\n a= %5d \t b= %7.3f", a, b);

Розберемо склад форматного рядка:

1. перевести курсор на новий рядок;
2. на екран вивести a= ;
3. вивести ціле значення (ширина поля для виведення 5). Після виконання дій 1-2 зі списку після форматного рядка буде зчитане перше значення і підставлене після =;
4. відступити (табуляція);
5. вивести b= ;
6. вивести дійсне значення. Значення дійсної змінної b буде виведене після =.

Введення даних scanf ()

Для введення даних використовується функція scanf (), яка є аналогом printf.

scanf (**"**форматний рядок**"**, **&**arg1, **&**arg2, ...);

Функція scanf зчитує дані, які вводить користувач з клавіатури, інтерпретує їх згідно з форматним рядком та відсилає в аргументи.

Форматний рядок може містити тільки опис типів змінних, що вводяться. Усі інші символи у форматному рядку ігноруються.

Кількість специфікацій типу у форматному рядку повинна дорівнювати кількості змінних.

Для опису типу використовують літери, що наведені у табл. 6.

Під час введення користувачем даних scanf ігнорує всі введені прогалини, табуляції (‘\t’) та натиснення клавіші < ENTER > (‘\n’).

Функція scanf припиняє введення, коли:

- вичерпані специфікації типів у форматному рядку;
- введені дані не відповідають вказаному формату. Наприклад, під час введення числових значень користувач помилково задав букви.

Особливістю роботи функції scanf () є те, що в списку введення для всіх змінних вказуються їх адреси в оперативній пам’яті. Для отримання адреси змінної перед її ідентифікатором ставиться значок **&** : **&**< ім’я змінної >.

Розглянемо два приклади введення змінних:

1. Різні типи змінних	2. Змінні одного типу
int a; float b; scanf (“%d%f”, &a, &b);	int k,w; scanf (“%d%d”, &k, &w);

Таблиця 6

Опис типів у форматному рядку scanf ()

Тип змінної	***Літера***	***Опис***
char	d	вводиться код символу
	c	вводиться символ
short	d	вводиться значення в діапазоні short
	hd	
int	d, i	вводиться знакове десяткове ціле
long	li	виводиться знакове довге ціле
unsigned unsigned char unsigned short unsigned int	u	вводиться беззнакові цілі
unsigned long	lu	
long int	li	вводиться довге ціле
float	f, g, e	значення дійсного типу вводиться з фіксованою або плаваючою крапкою
double	lf	значення дійсного типу вводиться з фіксованою або плаваючою крапкою
long double	lf	вводиться довге дійсне в форматі з фіксованою або плаваючою крапкою

Символьне введення/ виведення

Функції getc() та putc()

char getchar() – повертає наступну літеру з потоку введення або -1 у разі помилки.

putchar(char c) – виводить літеру на екран.

Контрольні запитання

1. Які бібліотеки введення/виведення мови Сі Ви знаєте? Наведіть декілька прикладів функції з цих бібліотек?
2. Функції форматного введення / виведення даних бібліотеки <stdio.h>.
3. Які правила формування форматного рядка для функцій printf () та для scanf ().
4. У чому особливості формування списку введення функції scanf ()?
5. Символьне введення / виведення даних бібліотеки <stdio.h>.

Лекція 5. Основи алгоритмізації обчислювального процесу

Існують 3 види обчислювальних процесів: лінійний, розгалужений та циклічний. Властивості, правила організації та способи зображення в схемі алгоритму кожного з процесів детально розглянуті в [3, 5]. Зараз зосередимося на особливостях реалізації розгалуженого та циклічного процесів у мові програмування Сі.

Розгалужений обчислювальний процес

Розгалужений обчислювальний процес у програмах мовою Сі реалізується за допомогою двох операторів: просте розгалуження – оператор if-else, множинне розгалуження – оператор switch-default.

Оператор розгалуження if-else

Оператор розгалуження if-else використовується тоді, коли залежно від виконання (чи невиконання) певної умови потрібно обрати та виконати відповідні дії (рис. 1, А). Загальний синтаксис оператора такий:

if (*умова* **)** *дія_1* **; else** *дія_2* **;**

умова – логічний або математичний вираз мови Сі. Якщо в результаті обчислення цього виразу отримане значення:

- не дорівнює нулю, вважається, що умова істинна і виконується *дія_1;*
- 0 (нуль), вважається, що умова хибна і виконується *дія_2.*

дія_1, дія_2 – звичайні інструкції мови Сі. Якщо необхідно виконати не одну дію, а декілька, використовується складений оператор (рис. 1, Б).

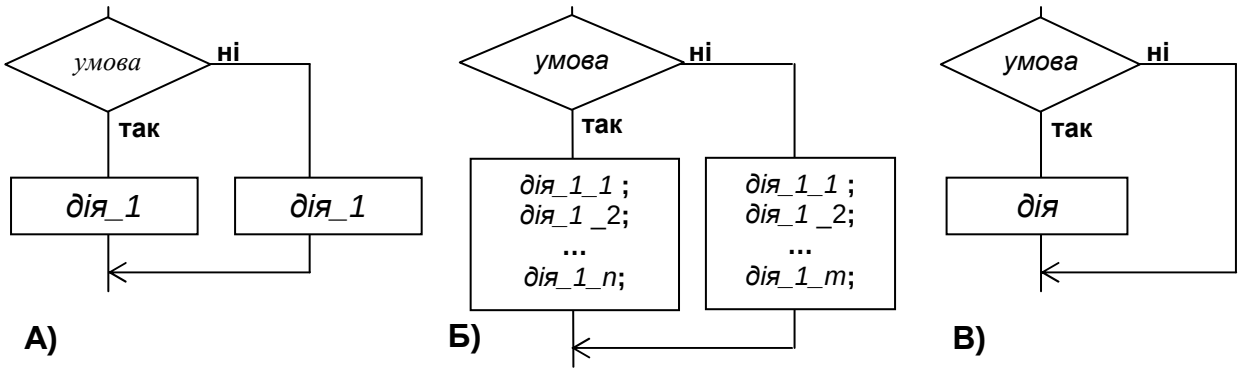


Рис. 3. Схеми алгоритму розгалуженого обчислювального процесу

Складений оператор – група операторів мови програмування Сі, що об'єднується операторними дужками ***{ }*** і надалі розглядаються компілятором як єдине ціле (один блок).

if (*умова* **)**
{
    дія_1_1 **;**
    *дія_1 _2***;**
    …
    дія_1_n **;**
}
else
{
    дія_2 _1**;**
    дія_2 _2**;**
    …
    дія_2 _m**;**
}

У випадку, коли необхідно виконувати дії тільки, якщо умова істинна, застосовується скорочена форма оператору **if** без частини **else** (рис. 1, В):

if (*умова* **)** *дія***;**

де *дія* – проста чи складений оператор мови Сі.

Зверніть увагу на те, що:

- умова після ключового слова **if** береться в дужки **()**;
- перед ключовим словом **else** ставиться **;**.

Оператор розгалуження switch-default

Оператор множинного розгалуження, або оператор вибору, або оператор-перемикач описується в Сі-програмі так:

switch (*вираз* ***)***
{
case *значення_1****:*** *дія_1****; break;***
case *значення_2****:*** *дія_2****; break;***
…
case *значення_m****:*** *дія_m****; break;***
default : *дія_(m+1)* ***;***
}

Схема алгоритму множинного розгалуження подана на рис. 2.

Дія цього оператору така:

1. обчислюється значення виразу, що вказаний у дужках після ключового слова ***switch*** ;
2. отриманий результат порівнюється зі значеннями, що вказані на гілках ***case***. Якщо якесь значення співпаде, виконується відповідна дія;
3. якщо жодне значення не підходить, тоді виконується дія, що вказана після ключового слова ***default***.

Дії можуть задаватися простим або складеним оператором мови Сі.

Рекомендується кожну гілку ***case*** завершувати командою ***break***, що перериває роботу оператора ***switch***. Нажаль, не всі компілятори мови Сі коректно обробляють цей оператор. Було б логічно, щоб після вибору та виконання певної дії робота оператора закінчувалася. Наприклад, якщо під час перевірки результат *виразу* співпав зі *значенням_1*, виконалася *дія_1*, і більше ніякі дії не виконувалися (були б пропущені). На практиці це відбувається не завжди так. Якщо обирається певна гілка, то після її виконання виконуються всі дії, що вказані за нею. Використання команди ***break*** цьому перешкоджає.

Оператор ***switch*** може використовуватися в скороченому вигляді без частини ***default***.

Якщо необхідно об’єднати декілька значень на одній гілці case, синтаксис буде такий:

switch (вираз)
{
case *значення_1:* ***case*** *значення_2:* *дія_1****; break;***
…
case *значення_m:* *дія_m****; break;***
default : *дія_(m+1)* ***;***
}

Циклічний обчислювальний процес

Існує три види циклічного обчислювального процесу:

- цикл з передумовою, коли спочатку перевіряється умова виконання тіла циклу, а потім виконується тіло, якщо умова істинна;
- цикл з постумовою, коли виконується тіло циклу, а після цього перевіряється умова для продовження виконання тіла циклу;
- цикл з лічильником, коли кількість ітерацій задається перед початком виконання тіла циклу.

Схеми алгоритмів подані на рис. 4.

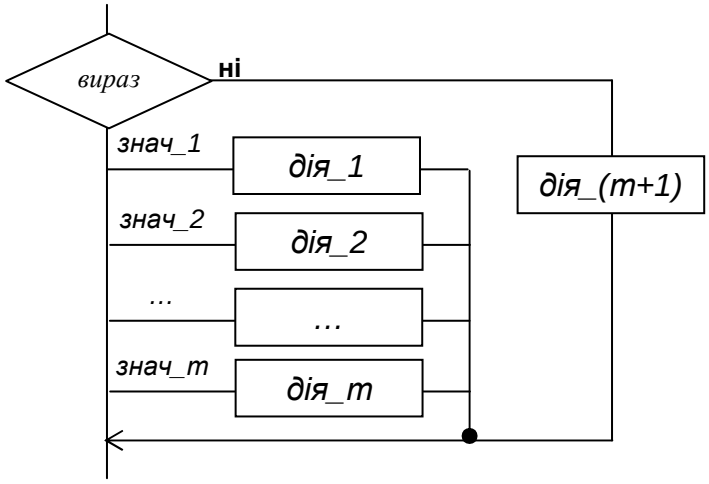


Рис. 4. Схеми алгоритму множинного розгалуження

Оператор циклу з передумовою while

Цикл з передумовою задається в Сі-програмі оператором ***while***:

while (*умова* ***)***

тіло циклу ***;***

де *умова* – логічний або математичний вираз мови Сі. Якщо умова **істинна** (не нуль), тіло циклу виконується;

тіло циклу – простий або складений оператор мови Сі.

Оператор циклу з постумовою do-while

Цикл з постумовою задається в Сі-програмі оператором ***do-while***:

do

{

тіло циклу ***;***

}

while (*умова* ***)***

де *умова* – логічний або математичний вираз мови Сі. Якщо умова **істинна** (не нуль), тіло циклу виконується ще раз;

тіло циклу – простий або складений оператор мови Сі.

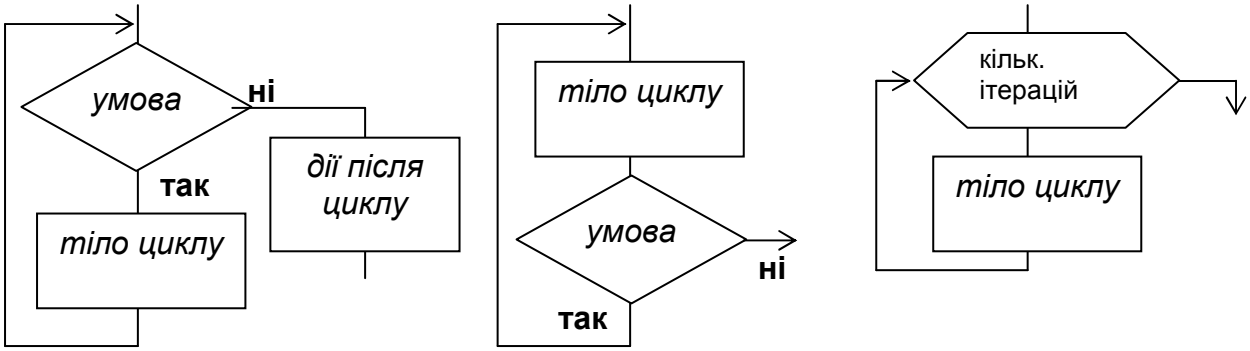


Рис. 5. Різновид циклічних обчислювальних процесів

Оператор циклу з лічильником for

Цикл з лічильником мови Сі описується так:

for (*вираз_1* ***;*** *вираз_2* ***;*** *вираз_3****)***

тіло циклу ***;***

вираз_1 – вираз, який виконується тільки один раз перед першою ітерацією і служить для ініціалізації змінної лічильника;

вираз_2 – логічний вираз, який задає умову продовження виконання тіла циклу;

вираз_3 – вираз мови Сі, який задає правила зміни лічильника після кожної ітерації циклу.

тіло циклу – простий або складений оператор мови Сі.

Наприклад, для обчислення суми натуральних чисел в інтервалі від 1 до 10, можна задати такий цикл:

```
int s=0, i;
for (i=1; i<=10; i++)
        s=s+i; /* або так –    s+=i; */
```

Один з виразів, декілька або всі в заголовку оператора for можуть бути відсутні:

відсутній один вираз	*відсутні два вирази*	*відсутні всі (вічний цикл)*
int s=0, i=1; for (; i<=10; i++) s=s+i;	int s=0, i=1; for (; i<=10;) { s=s+i; i++; }	for (; ;) *тіло циклу ;*

Зверніть увагу на те, що в заголовку ***;*** не пропускається.

Кожен з виразів ***вираз_1 ; вираз_2 ; вираз_3*** в заголовку оператора for може задаватися списком. У кожному списку вирази перелічуються через кому:

int s, i; *for (s=0, i=1; i<=10; s=s+i, i++) ;*	*int s, i;* *for (s=0, i=1; i<=10; s+= i++) ;*

Зверніть увагу на те, що ***;*** після заголовку ***for (…)*** ставиться тільки тоді, коли відсутнє тіло циклу !!!

Оператори break, continue

Оператор *break* використовується найчастіше для того, щоб перервати виконання тіла циклу. Програма продовжить роботу з першого оператора після тіла циклу.

Оператор *continue* використовується для того, щоб перервати виконання ітерації циклу. Програма продовжить обробку циклу з наступної ітерації.

Для того, щоб перервати роботу програми мови Сі використовується функція *exit(код завершення)*, що знаходиться в бібліотеці <*stdlib.h*>. *Код завершення* найчастіше дорівнює 0, тобто *exit(0).*

Контрольні запитання

1. Які існують види обчислювальних процесів? Що таке лінійний, розгалужений та циклічний обчислювальний процес?
2. Дайте визначення алгоритму. Які властивості алгоритму? Які засоби опису алгоритму Ви знаєте.
3. Що таке лінійний обчислювальний процес? Типова схема алгоритму лінійного обчислювального процесу?
4. Що таке розгалужений обчислювальний процес? Як розгалужений обчислювальний процес відображається в схемі алгоритму?
5. Призначення та правила використання оператора if мови Сі.
6. Призначення та правила використання оператора switch мови Сі.
7. Як формується складений (рус. составной) оператор у програмі мовою Сі. В яких випадках він використовуєтья?
8. Що таке циклічний обчислювальний процес? Що таке тіло циклу, умова циклу, параметр циклу, ітерація?

9. Що таке цикл з передумовою, постумовою та з лічильником? Як ці види циклів позначаються в схемі алгоритму?
10. Правила використання операторів циклу з лічильником мови Сі.
11. Правила використання оператору циклу з постумовою мови Сі.
12. Правила використання оператору циклу з передумовою мови Сі.

Лекція 6. Обробка масивів даних

Одновимірні масиви

Загальний вигляд опису одновимірного масиву мови Сі такий:

тип_елементів ім'я_масиву [кількість_елементів] ;

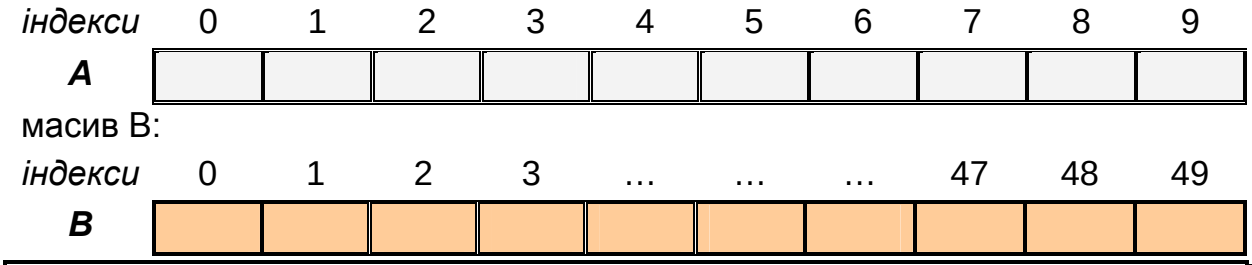
тип_елементів – будь-який базовий тип мови Сі або тип, що описаний програмістом;

ім'я_масиву – правильний ідентифікатор мови;

кількість _елементів – константа, що описана до опису масиву (див. опис масиву А нижче) або задана явно (див. опис масиву В).

```
#define N 10
int A[N];      {цілочисельний масив з 10 елементі}
float B[50];   {масив з 50-ти дійсних чисел}
```

У мові Сі нумерація елементів масиву завжди починається з нуля. Отже, масив А буде виглядати так:



Індекс першого елемента Сі-масиву завжди **дорівнює 0**.
Індекс останнього елемента Сі-масиву **на одиницю менший,** ніж *кількість _елементів*, що задана під час опису масиву.

Виходячи з цього, для введення масиву А можна організувати такий цикл:

```
int i;
for (i=0; i < N; i++)
 {
    printf ("Введіть елемент A[%d] :", i);
    scanf("%d", &A[i]);
 }
```

Звернення до елементу масиву в загальному вигляді здійснюється так:

ім'я_масиву [індекс_елемента]

Під час опису масиву можна провести ініціалізацію його елементів:

int x[5] = {2, 7, 9, 14, 23};

Якщо ініціалізація масиву проводиться явно, то можна не вказувати кількість елементів:

float y[] = {2.1, 4.67, 8.95, 0.89, 23.124, 6.75, 7.89}; {масив з 7 елементів}

Багатовимірні масиви

Загальний вигляд опису багатовимірного масиву (N-мірного) мови Сі такий:

тип_елементів ім'я_масиву [p1] [p2]… [pN] ;

тип_елементів – може бути будь-який базовий тип мови Сі або тип, що описаний програмістом;

ім'я_масиву – правильний ідентифікатор мови;

p1, p2, … , pN – константи, що задають кількість елементів на кожному вимірі.

Наприклад, опис двовимірного масиву, можна здійснити так:

#define N 3

#define M 5

int A[N][M]; {цілочисельна матриця з 3-х рядків та 5-ти стовпців}

Нумерація елементів на кожній розмірності багатовимірного масиву також починається з нуля.

індекси		0	1	2	3	4
A	0					
	1					

Звернення до елементу багатовимірного масиву в загальному вигляді здійснюється так:

ім'я_масиву [індекс1] [індекс2]… [індексN],

де кількість індексів визначається розмірністю масиву.

Виходячи з цього, для введення матриці A можна організувати такий цикл:

```
int i, j;
for (i=0; i < N; i++)
for (j=0; j < M; j++)
 {
    printf ("Введіть елемент A[%d][%d] :", i, j );
    scanf("%d", &A[i][j]);
 }
```

Явна ініціалізація двовимірного масиву така:

```
int  x[2][3] = { {2, 7}, {9, 14}, {23, 17} };
```

Можна не вказувати кількість елементів для першої розмірності за явної ініціалізації:

```
float  y[][3] = { {0.89, 23.124, 6.75},  {7.89, 0.0, 7,77} };
```

Рядки символів

У мові Сі не введенно спеціального типу для опису рядків символів, тому рядок описується як масив з символьним типом елементів:

char ім'я_рядка [кількість_символів] ;

Вважається, що взагалі кількість символів у рядку не обмежена, але існує спеціальна ознака кінця рядка – символ '\0', тому довжина рядка при його описі повинна бути на 1 більша за бажану довжину.

Під час оголошення можна проводити явну ініціалізацію рядка.

```
char s1[10];
char s2[] = "Папа у Васи силен в математике";
```

Треба зауважити, що під час опису рядка s2 розмір рядка не заданий, під час його ініціалізації в рядок записано 30 символів (враховуючи пробіли між словами). Довжина рядка буде 31 символ, адже наприкінці автоматично буде додано ознаку кінця рядка.

Введення/виведення рядків

Введення/виведення рядків можна здійснити за допомогою функцій форматного введення/виведення scanf, printf, якщо вказати в форматному рядку формат %s:

scanf("%s", *рядок*); /* *рядок* вказується без & попереду !!! */

printf ("%s", *рядок*);

Наприклад:

```
char s1[10];
 printf("\nInput string s1: ");
 scanf("%s", s1);
 printf("\nYour string s1: %s", s1);
```

Але, коли вводиться scanf рядок буде зчитаний до першого пробілу, наприклад, якщо необхідно ввести для рядка s1 таку послідовність "Миру мир", то після роботи scanf в рядок потрапить лише "Миру".

Щоб уникнути цього «побічного ефекту», рядок можна вводити та виводити, використовуючи спеціальні функції рядкового введення/ виведення gets() та puts() з бібліотеки <stdoi.h>. Загальний опис цих функцій такий:

- gets(*рядок*) зчитує рядок, що вводиться, до натиснення клавиші <Enter>. У кінець рядка автоматично додається символ '\0';

- puts(*рядок*) виводить на екран заданий рядок і переводить курсор на новий екранний рядок.

Наприклад:

```
char s2[10];
 puts("\nInput string s2: ");      gets(s2);
 puts("\nYour string s2: ");       puts(s2);
```

Таблиця 7

Спеціальні функції обробки рядків бібліотеки <string.h>

Назва функції	***Опис призначення***
strlen(s)	визначає довжину рядка, не враховуючи ознаку кінця. char s[ ] = “abcdef”; int len; len = strlen(s); // len набуде значення 6.
strcat(s1, s2)	«доклеює» до рядка s1 рядок s2. char s[ ]=”ab”, p[ ]=”cd”; strcat(s, p); // s набуде значення “abcd”
strncat(s1, s2, n)	«доклеює» до рядка s1 перші n символів з рядка s2.
strcpy(s1, s2)	копіює з рядка s2 символи до рядка s1. Ця функція за смислом аналогічна присвоєнню s1=s2, але в мові Сі **присвоювати** рядки один одному **не можна**!
strncpy(s1, s2, n)	копіює з рядка s2 перші n символів до рядка s1.
strchr(s, c)	перевіряє, чи є в рядку заданий символ c, якщо символ є, то повертає частину рядка s, що починається з *першого* входження символу c.
strrchr(s, c)	перевіряє, чи є в рядку заданий символ c, якщо символ є, то повертає частину рядка s, що починається з *останнього* входження символу c.
char s[ ]=”xxaabbaaxx”; char c=’a’;	strchr (s,c); → ”aabbaaxx” strrchr (s, c) → “axx”
strspn(s1, s2)	визначає номер першого символу, який входить в рядок s1, але не входить до рядка s2.
strstr(s1, s2)	визначає чи входить рядок s2 в рядок s1. Якщо входить, то повертає частину рядка s1, що починається з *першого* входження s2.
strtok(s1, s2)	визначає частину рядка s1, яка розташована перед першим однаковим символом в рядках s1 та s2.

Закінчення табл. 7

Назва функції	***Опис призначення***	
char s [ ]=”xxaabbcc”; char p[ ]=”ab”;		strspn(s,p); → 0 strstr(s, p) → “abbcc” strtok(s,p) →”xx”
strnset(s, c, n)	вставляє n разів на початок рядка s символ c.	
strupr(s)	переводить малі букви в рядку в великі (“xx”→”XX”).	
strlwr(s)	переводить великі букви в рядку в малі (“XX”→”xx”).	
strrev(s)	переписує вміст рядка в зворотному порядку (“abc”→”cba”).	
strcmp(s1, s2)	виконує порівняння рядків. Якщо рядок s1 співпадає з s2, то функція повертає 0, якщо рядок s1 менший (в алфавітному порядку) ніж s2, то повертається від’ємне значення, якщо більший – додатне.	
char s[ ] = “abc”; char p[ ] = “abc”; char t[ ] = “def”;		strcmp(s, p); → 0 strcmp(s, t); → від’ємне значення strcmp(t, s); → додатне значення

Контрольні запитання

1. Опис одновимірних масивів у Сі-програмі. Особливості індексації елементів масиву мови Сі.
2. Як здійснюється введення/виведення одновимірних масивів.
3. Які операції припустимі над масивами в мові Сі?
4. Опис двовимірних масивів. Що таке розмір та розмірність масиву?
5. Як здійснюється введення/виведення двовимірних масивів.
6. Як здійснити ініціалізацію елементів одно- та двовимірних масивів під час опису?
7. Як здійснюється опис багатовимірних масивів?
8. Як передати та обробити масиви у функції мови СІ?
9. Як здійснюється опис та ініціалізація рядків символів. Як визначається довжина рядка?
10. Як здійснити введення/виведення рядків символів?
11. Які операції є припустимими над рядками мови СІ?
12. Функції роботи з рядками мови Сі: strcpy, strncpy, strcat, strncat, strcmp, strncmp, strchr, strrchr, strlen, strspn, strcspn.

Лекція 7. Обробка файлів даних

Опис файлової змінної, відкриття та закриття файлу

Файлова змінна описується так:

FILE* *ім'я_файлової_змінної;*

FILE* f;

Для відкриття файлу використовується функція fopen:

ім'я_файлової_змінної = fopen (*ім'я_файлу, режим_доступу*);

ім'я_файлу – повне ім'я файлу з вказанням диску, та списку каталогів або скорочене ім'я. Треба зауважити, що в програмах мовою Сі роздільник між каталогами подвійний слеш, наприклад, "c:\\stud\\laba\\file.txt".

режим_доступу – файл може бути відкритий для читання, запису, дозапису в кінець файлу або в змішаному режимі. *Режим_доступу* задається одним з таких рядків:

- "r" – відкриття вже існуючого файлу для зчитування даних;
- "w" – створення нового файлу для запису даних. Якщо файл з вказаним ім'ям вже існує, то його вміст затирається для запису нових даних;
- "a" – відкриття вже існуючого файлу для дозапису інформації в кінець файла;
- "r+" – відкриття файла для читання і запису;
- "w+" – створення нового файлу з можливістю запису та читання інформації;
- "a+" – відкриття файла для запису даних у кінець файла та читання даних.

Для коректного збереження даних у файлі наприкінці роботи програми файл необхідно закрити:

fclose (*ім'я_файлової_змінної*);

Прграма мовою Сі розглядає всі файли як **текстові**. Але, якщо вказати відкриваючи файл формат "b", то такий файл вважається **бінарним** (він є аналогом нетипізованого файла Паскаль).

Таблиця 8

Стандартні функції роботи з файлами бібліотеки < stdio.h >

Назва функції	***Опис призначення***
feof (*ІФЗ*)	повертає 0, якщо досягнутий кінець файлу (*ІФЗ* – скорочення від *ім'я_файлової_змінної*). Для організації послідовного читання даних з файла можна організувати такий цикл: while (!feof(*ІФЗ*)) { обробка даних … }
ftell (*ІФЗ*)	повертає поточну позицію файлового покажчика або -1 у разі помилки.
rewind (*ІФЗ*)	переводить файловий покажчик на початок файла
fseek (*ІФЗ, зміщення, напрямок*)	переводить файловий покажчик на задане зміщення в заданому напрямку. Напрямок може задаватися такими константами: ▪ SEEK_SET – відносно початку файла; ▪ SEEK_CUR – відносно поточної позиції файлового покажчика; ▪ SEEK_END – відносно кінця файла. Але під час роботи з текстовим файлом можна використовувати тільки константу SEEK_ SET.

Файлове введення/виведення бібліотеки < stdio.h >

Для організації файлового введення/виведення в бібліотеці <*stdio.h*> передбачено десяток функцій. Вибір тієї чи іншої функції залежить від поставлених задач або вподобань програміста. Далі наводяться найпопулярніші з цих функцій.

Форматне файлове введення/виведення

Використання функцій форматного файлового введення виведення майже нічим не відрізняється від простого форматного введення/ виведення, окрім першого параметру – *імені_файлової _змінної*:

fscanf (*ІФЗ*, **"**форматний рядок**"**, **&**arg1, **&**arg2, ...);

fprintf (*ІФЗ*, **"**форматний рядок**"**, arg1, arg2, ...);

Символьне файлове введення/виведення

fgetc(*ІФЗ*) – повертає як результат символ, що зчитаний з файлу.

fputc(char c, *ІФЗ*) – виводить символ у файл.

Рядкове файлове введення/ виведення

fgets(*рядок, кількість_символів, ІФЗ*) – зчитує з файлу *рядок* заданої довжини (*кількість_символів*) з файлу.

fputs(*рядок, ІФЗ*) виводить в файл заданий *рядок*.

Контрольні запитання

1. Дайте визначення файлу? Коли доцільно зберігати у файл дані з програми?
2. Для чого потрібна в програмі файлова змінна і як вона описується?
3. Особливості структури текстового файлу. Чи можна в текстовий файл записати чисельні значення?
4. У чому особливості бінарних файлів мови Сі?
5. Призначення та правила використання функцій fopen (). Які режими доступу до файлу існують?
6. Як організувати послідовне читання даних з текстового файлу (або запис у файл послідовності значень)?
7. Для чого використовується функція fclose ()? Чому необхідно дбати про закриття файлу під час завершення програми?
8. Які стандартні функції роботи з файлами бібліотеки <stdio.h>?
9. Функції форматного файлове введення/виведення даних бібліотеки <stdio.h>.
10. Які функції символьного та рядкового файлового введення/виведення?

Лекція 8. Структуровані типи даних

Структури struct

Структури допомагають в організації складних даних. Вони дозволяють групу різних за типом, але пов'язаних за змістом елементів, подати як єдине ціле.

Опис структури здійснюється так:

struct *ім'я_тегу*

{

тип1 поле1;

тип2 поле2;

…

} *опис*_змінних ;

ім'я_тегу – правильний ідентифікатор групи елементів структури;

поле1, поле2 – правильні ідентифікатори елементів структури різних типів;

*опис*_змінних – перелік ідентифікаторів змінних типу структура (цей перелік може бути відсутній).

Імена полів (або членів структури) можуть збігатися з іменами інших змінних. Однакові імена полів можуть збігатися в різних структурах.

Як приклад розглянемо опис структури для збереження та обробки координат точки (дійсного типу), її кольору (цілочисельне значення) та її імені (в якості імені можна використовувати букви латинського алфавіту).

```
struct point
{
float x, y;     int color;      char name;
}  p1, p2;
```

Якщо в іншому місці програми необхідно оголосити ще змінні для точок, для їх опису використовується *ім'я_тегу*:

```
struct point z1, z2, z3;
```

Явна ініціалізація полів структури може проводитися під час її опису (опис з ініціалізацією полів називається визначенням):

```
struct point c1 = { 0.0, 0.0, 2, 'A'};
```

Точка p1 набуває таких властивостей: координати (0.0, 0.0), колір – зелений (2 – код зеленого кольору), ім'я точки – A.

Доступ до окремого елемента структури:

ім'я_зміної . поле

```
p1. x = 10.5;
p1. y = 2.75;
```

Структури можуть бути використані як параметри функції і повертаються як результат:

```
struct poin  add ( struct point a, struct point b )
{
        a. x + =  b. x ;
        a. y + =  b,y;
        return  a;
   }
```

Введення/виведення структур

Введення/виведення структур проводиться по кожному полю окремо:

```
scanf("%f%f%d%c", &p1.x, &p1.y, &p1.color, &p1.name);
printf("x=%f\ty=%f\ncolor=%d\nname=%c\n", p1.x, p1.y, p1.color,
                                                    p1.name);
```

Об’єднання union

Об’єднання – це змінна, яка також як і структура може містити в собі елементи різних типів. Але в кожний момент часу, об’єднання може зберігати значення тільки одного елемента з набору .

Синтаксис опису об’єднання аналогічний до структури:

union *ім’я_тегу*

{

тип1 поле1;

тип2 поле2;

…

} *опис*_змінних ;

Якщо для кожного з полів структури в оперативній пам’яті виділяється окрема ділянка для збереження значення, то для полів об’єднання виділяється одна ділянка максимально потрібного розміру (рис. 6).

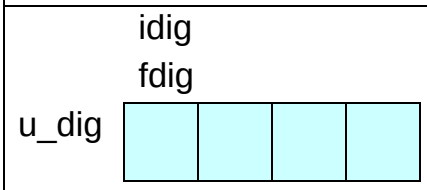
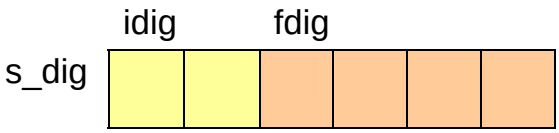
union digits	struct digits
{	{
int idig;	int idig;
float fdig;	float fdig;
} u_dig;	} s_dig;
	

Рис. 6. Порівняння об’єднання та структури

Для збереження даних структури digits буде виділена область в 6 байтів (сума розмірів полів структури), значення кожного поля зберігається окремо одно від іншого.

Для збереження даних об’єднання digits буде виділено 4 байти (максимальний розмір з полів 4 байти). У кожний момент часу в об’єднанні може зберігатися значення тільки одного поля. Присвоєння значення одному полю знищить (зітре) значення іншого поля:

u_dig. idig = 12;

u_dig. fdig = 7.5;

Після такого присвоєння буде збережено тільки значення 7.5.

Перелічення enum

enum *ім'я_тегу* { *список_перелічення* } *опис*_змінних ;

список_перелічення – список констант з зазначенням їх значення:

константа1=значення1, константа2=значення2, …

Константи в списку розділяються комами.

Значення констант – додатні чи від'ємні цілі числа. *Значення констант* можуть бути відсутніми, тоді константам присвоюються значення за замовчанням, розпочинаючи з 0.

enum boolean { false = 0 , true = 1 } bool;

Використовується змінна-перелічення як звичайна змінна:

bool=true; //присвоєння значення

if (bool == false) … //перевірка значення

Контрольні запитання

1. У яких випадках доцільно використовувати тип структура (struct) мови Сі? Як описати структуру?
2. Чи можуть бути структури вкладеними? Як здійснити такий опис (наведіть приклад)?
3. Як здійснюється введення/виведення даних для полів структури?
4. Опис та використання типу об'єднання (union) мови Сі. В чому особливості його використання?
5. Як поля структур та об'єднань зберігаються в пам'яті? Як визначити їх розмір?
6. Як передати структури та об'єднання в якості параметрів у функції мови Сі?
7. Опис та використання типу enum мови Сі.
8. Як здійснюється опис власних типів у програмах мовою Сі за допомогою typedef?

Підпрограми в програмах мови Сі

На відміну від Паскаль-програм в Сі-програмах:

- існують тільки підпрогами-функції;
- вкладених функцій бути не може, тобто функція не може бути описана всередині іншої функції.

Опис функції

Опис функції в Сі-програмах здійснюється на зовнішньому рівні, тобто поза будь-яких інших функції.

Синтаксис опису функції такий:

тип_результату **ім'я_функції** (список_формальних_параметрів)

{
тіло_функції;
return результат;
}

Тип_результату функції – будь-який простий тип даних мови Сі. Якщо тип результату під час опису не визначений, то вважається, що функція повертає результат типу int. Якщо функція не повертає результат в основну програму (тобто є аналогом Паскаль-процедури), необхідно вказати тип void. Функція не може повертати як результат масив, але може повертати покажчик на нього.

Ім'я_функції – правильний ідентифікатор мови Сі.

Список_формальних_параметрів формується за таким правилом:

тип1 параметр1, *тип2* параметр2, …, *типN* парметрN

Якщо в функцію передаються декілька параметрів одного типу, нажаль, їх неможна об'єднати в одну групу. Наприклад, якщо в функцію func дійсного типу, передаються два параметри типу int, то опис заголовку функції буде:

float func(int a, int b)

Якщо функція не має параметрів, то замість списку параметрів вказується тип void.

Необхідно звернути увагу на те, що після заголовку функції перед тілом функції крапка з комою не ставиться. Крапка з комою після заголовку ставиться тільки тоді, коли здійснюється випереджене оголошення функції (див. відповідний розділ).

Тіло_функції може включати опис змінних та оператори (інструкції) програми.

Інструкція ***return*** реалізує механізм повернення результату роботи функції в основну програму. Як результат функції в рядку return може бути вказане безпосередньо значення, наприклад:

return 0;

або вираз:

return sqrt(a+b);

Тип значення або тип результату виразу, що вказується в рядку return повинен бути сумісним з типом функції, що вказаний у заголовку.

Якщо рядок return не вказаний, або не вказаний результат, що повертається, то функція поверне в точку виклику невизначене значення ("сміття").

Виклик функції

Зазвичай виклик функції має бути операндом у виразі або аргументом іншої функції, але в мові Сі це обмеження не є жорстким. Виклик функції може бути оформлений як окремий оператор програми, тоді результат, який повертає функція, ігнорується. Наприклад, визначена така функція:

```
float f(float x, int a)
{
 return x+a;
}
```

Виклик такої функції може бути таким:

```
float y, b=10.5; int s=4;
y=f(b, s);
```

Або таким:

```
float b=10.5; int s=4;
printf("\n Result=%9.3f \n", f(b, s));
```

А може бути таким:

```
float y, b=10.5; int s=4;
f(b, s);
```

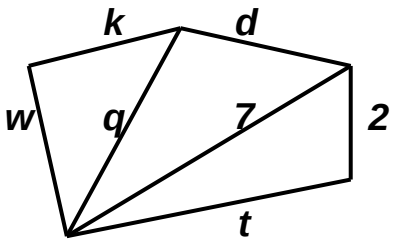
Синтаксис мови не накладає ніяких обмежень на виклик функції, окрім єдиного: список фактичних параметрів повинен збігатися за кількістю та бути сумісним за типами зі списком формальних параметрів. Якщо функція була описана з порожнім (void) списком параметрів, її виклик має синтаксис:

```
ім'я_функції();
```

Порожні дужки упускати не можна!!!

Приклад використання функції

Як приклад буде розглянуто опис функції для визначення площі трикутника, яка використовується для визначення площі фігури, заданої на рисунку.




```
#include <stdio.h>
#include <math.h>
float S_tr (float a, float b, float c)
  {
    int p=(a+b+c)/2;
    return sqrt(p*(p-a)*(p-b)*(p-c));
  }
void main (void)
{
```

```
    float w, k, q, d, t, S_f;
    printf("Введіть довжини сторін фігури w, k, q, d, t:");
    scanf("%f%f%f%f%f", &w, &k, &q, &d, &t);

    S_f:=S_tr(w, k, q)+S_tr(q, d, 7)+S_tr(7, t, 2);
    printf("\nПлоща фігури = %f10.4 ",  S_f);
}
```

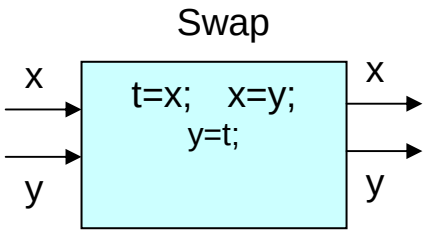
Різновид параметрів функцій

У вищенаведеному прикладі параметри a, b, c функції S_tr передавалися за значенням, тобто під час виклику функції значення фактичних параметрів копіювалися в формальні.

Як і в мові Паскаль в Сі-прогамах є спеціальний синтаксис для опису параметрів різних категорій. У цьому випадку залежно від компілятора, з яким ви працюєте можна застосовувати дві нотації:

- передача параметрів через покажчики;
- передача параметрів через посилання (тільки для компіляторів C++).

Передача параметрів-змінних



Як приклад розглянемо функцію, яка міняє місцями значення двох змінних x та y.

На вході і на виході цієї функції два параметри x та y, отже вони мають бути описані як параметри-змінні. N

Таблиця 9

Для C та C++	***Тільки для C++***
Опис функції	
void Swap(int *x, int *y)	void Swap(int &x, int &y)
{	{
int t;	int t;
t = *x;	t = x;
*x = *y;	x = y;
*y = t;	y = t;
}	}
Виклик функції	
int a, b;	int a, b;
Swap(&a, &b);	Swap(a, b);

Передача параметрів-масивів

Якщо в функцію потрібно передати масив, він передається тільки як параметр-змінна. Наприклад, розглянемо функцію, яка обчислює суму цілочисельного масиву A.

```
long int summa (int *A, int n)
  {
     int i; long int s=0;
     for (i=0; i<n; i++)
          s+=A[i];
     return s;
  }
```

З опису параметру A навіть і не зрозуміло, що це масив. З одного боку, це значно ускладнює розуміння тексту програми, з іншого – така функція може обробляти будь-який за розміром цілочисельний масив, тому другим параметром можна передати кількість елементів у масиві.

Виклик такої функції з головної програми буде таким:

```
int x[20]; long int s_x;
s_x = summa(x, 20);
```

Як видно з наведеного приклада опис параметрів-масивів, їх використання в тілі Сі-функції та передача як фактичних параметрів за синтаксисом значно відрізняються від простих параметрів-змінних.

Передача як параметрів структурованих типів даних

Структуровані типи даних (структури та об'єднання) бажано передавати в функції також як параметри-змінні. Цей матеріал буде розглянуто після вивчення теми «Покажчики та структуровані типи даних мови Сі».

Передача як параметрів інших функцій

Параметрами однієї функції може бути інша функція. Загальний синтаксис опису параметра-функції (ПФ) такий:

```
тип_результату ім'я_функції
                    (тип_результату_ПФ (*ім'я_ПФ)
                                        (список_типів-параметрів_ПФ) )
```

Виклик параметра-функції здійснюється так:

```
(* ім'я_функції)(список фактичних)
```

Розглянемо приклад. Описати дві функції $y = \cos x + e^x$ та $y = \sin x - e^x$. Вивести на екран таблиці табулювання цих функцій на проміжку [a, b] з кроком dx.

Загальний алгоритм буде таким:

1. Ввести значення a, b, dx.
2. Протабулювати функцію $y = \cos x + e^x$.
3. Протабулювати функцію $y = \sin x - e^x$.

Необхідно зауважити, що табулювання першої та другої функції здійснюється за однаковим алгоритмом, тобто в даній програми маємо два ідентичні (однакові) фрагменти коду, які відрізняються тільки функцією для табулювання. Тому бажано описати алгоритм табулювання функції як підпрограму, передаючи в неї як параметр функцію для табулювання.

Таблиця 10

До опису підпрограми	*Після опису підпрограми*
#include <stdio.h> #include <conio.h> #include <math.h> float y1(float x) { return cos(x)+exp(x); } float y2(float x) { return sin(x)-exp(x); } main () { float a, b, dx, x; clrscr(); printf(“Введіть [a, b] та крок dx:”); scanf(“%f%f%f”, &a, &b, &dx); x=a; while (x<=b) { printf(“%7.3f\t%7.3f”, x, y1(x)); x+=d; } x=a; while (x<=b) { printf(“%7.3f\t%7.3f”, x, y2(x));	#include <stdio.h> #include <conio.h> #include <math.h> float y1(float x) { return cos(x)+exp(x); } float y2(float x) { return sin(x)-exp(x); } void table (float p, float k, float d, float (*y)(float)) { float x=p; while (x<=k) { printf(“%7.3f\t%7.3f”, x, (*y)(x)); x+=d; } } main () { float a, b, dx; clrscr(); printf(“Введіть [a, b] та крок dx:”); scanf(“%f%f%f”, &a, &b, &dx);

Закінчення табл. 10

До опису підпрограми	*Після опису підпрограми*
x+=d; } }	 table(a, b, dx, y1); table(a, b, dx, y2); }

У наведених текстах програм пропущені фрагменти виведення шапки таблиці та роздільників для стовпчиків для того, щоб не обтяжувати приклад «несуттєвими дрібницями».

Розглянемо докладніше опис та виклик функції table. У заголовку функції описано 4 параметри: три парметри-значення початок p та кінець інтервалу k, крок d; четвертий парметр-функція з ім'ям y, типом результату float, та одним власним параметром також типу float:

void table (float p, float k, float d, float (*y)(float))

У тілі функції здійснюється виклик функції y з фактичним параметром x:

printf("%7.3f\t%7.3f", x, (*y)(x));

У головній функції main() двічі викликається функція table для двох фактичних параметрів-функцій y1 та y2:

table(a, b, dx, y1); table(a, b, dx, y2);

Контрольні запитання

1. Визначення підпрограми. Правила опису функцій у програмі мовою Сі. Як функція викликається з головної програми? Чи можуть бути в мові Сі вкладені функції?
2. Яким може бути тип результату функції? Чи може функція повертати декілька результатів? Чи може функція не повертати жодного результату (якщо так, то яким чином описується така функція)?
3. Чи може бути пустим список параметрів? Що таке формальні та фактичні параметри функції? Чи можуть бути вхідні та вихідні параметри у функції (якщо так, то яким чином описується така функція)?
4. Чим відрізняються параметри-змінні та параметри-значення функції, яке їх призначення та доцільність використання?
5. Механізм передачі параметрів-змінних та параметрів-значень у функціях під час виклику з головної програми.
6. Рекурсивний виклик підпрограм. Пряма та непряма рекурсія.
7. Рекурсивне занурення та підйом. Чи може (повинна) бути обмежена кількість рекурсивних викликів?

Локальність та глобальність ідентифікаторів у Сі-програмах

Час життя і область дії змінних

Поняття локальності та глобальності безпосередньо пов'язані з поняттям часу життя та областю дії ідентифікаторів.

Ідентифікатор з *глобальним часом* життя характеризується тим, що **на весь час** виконання програми за ним закріплена комірка оперативної пам'яті і певне значення.

Ідентифікатору з *локальним часом* життя виділяється нова комірка пам'яті **під час кожного входу в блок**, в якому він визначений або оголошений. Коли виконання блоку завершується, пам'ять, виділена під збереження значення ідентифікатора, звільняється і його значення втрачається.

Блок – складений оператор мови Сі, містить оператори та інші блоки.

Слід зауважити, що в мові Сі розрізняються поняття **визначення** та **оголошення** ідентифікатора. Під *оголошенням змінної* розуміється опис типу змінної, під *визначенням змінної* розуміється опис типу змінної та ініціалізація значення змінної. Під *оголошенням функції* розуміється опис заголовку функції, під *визначенням функції* розуміється опис заголовку та тіла функції.

Оголошення і визначення, що знаходяться всередині блоку називається *внутрішніми* або зробленими *на внутрішньому рівні*.

Оголошення і визначення, що знаходяться за межами всіх блоків називається *зовнішнім* або зробленими *на зовнішньому рівні.*

Усі ідентифікатори, що визначені на зовнішньому рівні мають глобальний час життя та область дії, що починається від місця визначення і до кінця тексту програми.

Усі ідентифікатори, що визначені (оголошені) на внутрішньому рівні мають локальний час життя та область дії в межах блоку, в якому вони описані.

Класи пам'яті мови Сі

У мові Сі є потужний механізм, який впливає на час життя та область дії змінних – класи пам'яті:

- extern – зовнішній;
- static – статичний;
- auto – автоматичний;
- register – регістровий.

Загальний синтаксис оголошення змінних такий:

клас_пам'яті тип ідентифікатор;

Вплив класів пам'яті на час життя та область дії змінних залежить від того, описана змінна на зовнішньому або внутрішньому рівні (клас register буде описаний окремо).

Таблиця 11

Клас пам'яті	*Зовнішній рівень*		*Внутрішній рівень*	
	Час життя	*Область дії*	*Час життя*	*Область дії*
extern	глобальний	глобальна	глобальний	глобальна
static	глобальний	локальна в межах файлу	глобальний	локальна в межах блоку
auto	не використовується		локальний	локальна

Опис змінної на зовнішньому рівні

На зовнішньому рівні використовують тільки класи пам'яті static та extern.

Якщо змінна оголошена на зовнішньому рівні з класом пам'яті extern, вважається, що вона визначена в іншій частині програми: нижче по тексту на зовнішньому рівні або в іншому файлі. Тоді під збереження цієї змінної не виділяється додаткова (нова) пам'ять, а коли вона використовується буде знайдено значення зовнішньої змінної.

Наприклад, програма складається з двох файлів c1.c та c2.c.

При компіляції буде помилка невизначений ідентифікатор x.

/* c1.c */	/* c2.c */
int x=0;	int x=4;
main()	
{	
x=x+10;	
}	

Значення x буде 10. Змінна x повторно визначена в файлі c1, для неї буде виділена нова область пам'яті, а значення x з файлу c2 буде недоступним. Якщо оголосити в головній програмі цю змінну з класом пам'яті extern:

/* c1.c */	/* c2. c */
extern int x;	
main()	int x=4;
{	
x=x+10;	
}	

Тоді значення x буде 14. Змінна x в головній програмі c1 описана як зовнішня, для неї **не буде** виділена нова область пам’яті, а буде знайдене значення x з файлу c2.

Для кожної зовнішньої змінної повинно бути одне єдине визначення. Інші файли для того, щоб отримати доступ до неї, повинні включати її опис з класом пам’яті extern.

Якщо змінна на зовнішньому рівні об'явлена зі словом static, то її область використання обмежується файлом, в якому вона описана, тобто вона не може використовуватися іншими файлами програми.

Якщо змінні з класом static явно не визначені, вони отримують початкове значення 0.

Якщо не визначена змінна з класом extern, то вважається, що вона отримує початкове значення в іншому місці програми (в іншому файлі).

Опис змінної на внутрішньому рівні

На внутрішньому рівні можна використовувати всі чотири класи пам’яті.

Змінні на внутрішньому рівні мають клас пам’яті auto за замовчанням, тобто його використання ніяк не вплине на локальність часу життя та області дії змінної.

Змінна, що описана на внутрішньому рівні з класом пам’яті static, буде мати глобальний час життя та локальну область дії. Це означає, що за межами блоку, в якому описана ця змінна звертатися до неї не можна, але значення цієї змінної буде зберігатися в пам’яті на весь час роботи головної програми.

Опис змінної з класом пам’яті extern, як і на зовнішньому рівні є посиланням на змінну з тим самим ім'ям, описану в іншому місці програми (іншому файлі).

Клас пам’яті register

З класом пам’яті register можуть бути описані тільки цілочисельні змінні за розміром не більше за int. Ці змінні за можливості будуть зберігатися в регістрах процесора. Якщо всі регістри (а їх кількість обмежена) будуть використані, то значення змінної повернеться в оперативну пам’ять.

Контрольні запитання

1. Структура програми мовою Сі. Що означає і як здійснюється опис змінних на зовнішньому та внутрішньому рівнях?

2. Локальність та глобальність змінних у програмі мовою Сі. Час життя та область дії змінних. Наведіть приклади локальних та глобальних змінних за часом життя та областю дії.
3. Чи може ім’я локальної змінної збігатися з іменем певної глобальної змінної? Якщо так, то на що це може впливати під час роботи програми? Поясніть чому локальна змінна-двійник не впливатиме на значення глобальної змінної (підказка: згадайте сегменти пам’яті, де зберігаються під час роботи лок. та глоб. змінні)?
4. Класи пам’яті мови Сі та їх вплив на час життя та область дії змінних. Опис змінних на зовнішньому рівні з класами пам’яті extern, static.
5. Класи пам’яті мови Сі та їх вплив на час життя і область дії змінних. Опис змінних на внутрішньому рівні з класами пам’яті extern, static.
6. Класи пам’яті automatic, register.

Лекція 11. Динамічний розподіл пам’яті

Змінні покажчики

Покажчик – змінна, що зберігає адресу області (комірки) в оперативній пам’яті. Змінна-покажчик (ЗП) може бути типізованою чи нетипізованою.

Типізований покажчик

```
тип*  ім’я_ЗП;
int*  p;
```

Нетипізований покажчик

```
void* ім’я_ЗП;
void * t;
```

У вищенаведеному прикладі змінна-покажчик p – типізований покажчик. Це означає, що вона може зберігати адресу області пам’яті, в яку записане тільки цілочисельне значення (під час опису змінної задано цілочисельний тип даних).

Змінна-покажчик t – нетипізований покажчик, тобто ця змінна зберігає адресу області пам’яті, в яку записані дані невизначеного типу. Але відомо, що всі дані повинні бути описані певним типом, тому під час роботи з нетипізованим покажчиком необхідно завжди явно приводити тип (див. відповідний розділ нижче).

Розподілення пам'яті

Виділення пам'яті для покажчиків

Після опису зміної-покажчика (ЗП) її значення невизначене, адже під час опису ніяку адресу в цю область не було записано. Отже, необхідно виділити область пам'яті та записати адресу цієї області в ЗП.

Типізований покажчик

ім'я_ЗП = (тип*)malloc(sizeof(тип));

p = (int*)malloc(sizeof(int));

Нетипізований покажчик

ім'я_ЗП = malloc(розмір області);

t = malloc(sizeof(float));

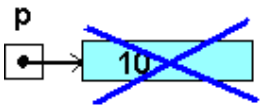
У мові С функція виділення пам'яті однакова для нетипізованих та типізованих покажчиків. У цьому випадку функція malloc орієнтована на роботу з нетипізованими покажчиками, тому під час виділення пам'яті під типізовані покажчики необхідно явно приводити тип результату malloc до типу покажчика.

Параметром malloc є розмір області пам'яті, яку необхідно виділити. У вищенаведеному прикладі виділяється область пам'яті розміром рівним до розміру дійсного значення типу real (float).

Звільнення пам'яті для покажчиків

Після того, як «усі роботи» з використанням покажчика закінчені, ***обов'язково необхідно звільнити*** виділену область пам'яті:

Типізований покажчик
free(ім'я_ЗП);
free(p);
Нетипізований покажчик
free(ім'я_ЗП);
free(t);

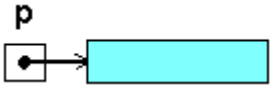


Розкриття посилання

Розкриття посилання забезпечує доступ до вмісту області пам'яті, на яку посилається покажчик. Розглянемо приклад:

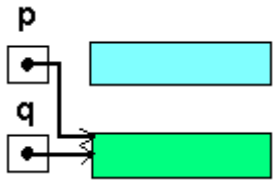
int* p, q;

p =(int*)malloc(sizeof(int));



q =(int*)malloc(sizeof(int));

p = q;



У наведеному прикладі описані два типізовані покажчики p та q. Для них виділені області пам'яті рівного розміру. Після цього відбувається присвоєння p = q, яке призведе до того, що до змінної p буде записана та сама адреса, що зберігається в змінній q (у цьому випадку адресу, що зберігалася в p буде втрачено!).

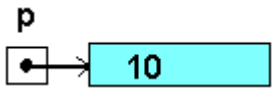
Під час роботи з покажчиками необхідно чітко розрізняти, коли ведеться робота з адресами виділених областей пам'яті, а коли з вмістом виділених областей пам'яті (тобто з даними, на які посилаються покажчики). Щоб отримати доступ до області пам'яті, на яку посилається покажчик, необхідно застосувати операцію розкриття посилання:

*ім'я_ЗП

*p

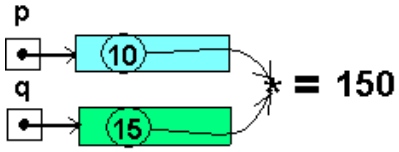
Розглянемо приклад, в якому у виділені області пам'яті для покажчиків p та q записуються певні значення, а їх добуток виводиться на екран.

*p = 10;



*p = 15;

printf("Добуток=%d",
(*p)*(*q));



Приведення типів для нетипізованих покажчиків

Під час роботи з нетипізованими покажчиками, коли відбувається розкриття посилання потрібно явно приводити тип (t – нетипізований покажчик з попереднього розділу):

(тип_ЗП)ім'я_ЗП

(float*)t

У мові Сі під час явного приведення типів для нетипізованого покажчика спочатку нетипізований покажчик приводиться до типізованого покажчика, а потім розкривається посилання.

Покажчики та структуровані типи даних мови Сі

Опис покажчика на структурований тип даних – структуру мови Сі – виконується наступним чином:

```
struct Some {    /* тег структури */
  int A;
  float B
};
struct Some *p;   /* покажчик на структуру */
```

Виділення пам'яті під покажчик (зверніть увагу на приведення типу!):

```
p=(struct Some*)malloc(sizeof (struct Some));
```

Доступ до полів запису (структури) виконується наступним чином:

- спочатку розкривається посилання, унаслідок чого отримується доступ до області пам'яті, де зберігаються дані полів запису (структури);
- здійснюється доступ до конкретного поля структури. Слід зауважити, що в мові Сі доступ до полів структур можна здійснити двома способами: через розкриття посилання та оператор '**.**' або за допомогою оператора '->' (стрілка):

```
*p.A = 10;  або так  p->A = 10;
*p.B = 0.5; або так p->B = 0.5;
```

У випадку роботи з нетипізованими покажчиками доступ до полів здійснюється аналогічно, але з обов'язковим явним приведенням типу покажчика:

```
void *t;
*(struct Some*)t.A = 10;  або (struct Some*)t ->A = 10;
*(struct Some*)t.B = 0.5; або (struct Some*)t->B = 0.5;
```

Контрольні запитання

1. Що таке змінна покажчика?
2. Як описати в програмі типізований та нетипізований покажчик? Чим вони відрізняються за призначенням?
3. Як здійснюється операція розкриття посилання (покажчика) та отримання адреси змінної? Наведіть приклад використання.
4. Для чого і як здійснювати приведення типів під час роботи з нетипізованими покажчиками?
5. Що таке динамічний розподіл пам'яті? Як називається область пам'яті, в якій здійснюється динамічний розподіл?
6. Які функції розподілення пам'яті мови Сі? Яких правил потрібно дотримуватися, коли розробляються програми з динамічним розподілом пам'яті?

7. Як виділити пам’ять для типізованого та нетипізованого покажчика мови Сі?
8. Як звільнити пам’ять, що була виділена динамічно? Чи небхідно дбати про звільнення динамічної пам’яті по завершенні роботи (відповідь обґрунтуйте)?
9. Правила використання покажчиків на масиви та структури мови Сі.

Рекомендована література

1. *Глинський Я.М.* С++ і С++Builder / Я.М. Глинський, В.Є. Анохін, В.А. Ряжська. – Львів: Деол, 2003. – 193 с.
2. *Кернига́н Б.* Язык программирования Си: [пер. с англ.]. – 4-е изд. – М.: Вильямс, 2007. – С. 304.
3. *Ковалюк Т.В.* Алгоритмізація та програмування: підручник. – Львів: Магнолія 2006, 2013. – 400 с.
4. *Шпак З.Я.* Програмування мовою С / З.Я. Шпак. – Львів: Оріяна-Нова, 2006. – 432 с.

Методична література

5. *Красовська Г.В.* Основи програмування та алгоритмічні мови. Мова програмування Паскаль. Основні поняття та управляючі конструкції: конспект лекцій, ч.1 / Г.В. Красовська. – К.: КНУБА, 2003. – 64 с.
6. *Красовська Г.В.* Основи програмування та алгоритмічні мови. Мова програмування Сі [*Електронний ресурс*]: конспект лекцій, ч.2 / Г.В. Красовська. – К.: КНУБА, 2008. – 54 с. – Режим доступу: http://org.knuba.edu.ua/mod/resource/view.php?id=2151 (дата звернення 26.11.2014)