Класи пам'яті та простір імен. Поняття функції

Лектор:
к.т.н., доцент кафедри прикладної математики
Рижа Ірина Андріївна

Про що ця лекція???

- Введемо поняття функції.
- Опишемо особливості оголошення та означення функцій.
- Розглянемо поняття областей видимості змінних та класи пам'яті.

Поняття функції

Функція

— це об’єднання групи операторів-інструкцій із наданням їм спільного імені, як правило унікального.

- До такого об’єднання можна звернутися (викликати) з інших частин програми.
- Використання функції дозволяє скоротити код програми.
- Незважаючи на те, скільки разів (один чи декілька) у програмі виконується певна функція, її код зберігається в одній області пам’яті.

Головна функція

Мова С є ***процедурно-орієнтованою***, тому кожна програма на С обов’язково має одну функцію, яка називається `main()`.

- Функція з таким ім’ям у проекті (програмі) має бути ***єдиною***.
- Ця функція викликається системою і не потребує попереднього оголошення.

Поняття функції

Функції середовища

— це функції, що надаються користувачеві середовищем програмування.

- Для їх використання за допомогою відповідної директиви `#include` під'єднується відповідний заголовковий файл, який містить оголошення необхідних функцій.

Функції користувача

— поіменована, логічно самостійна сукупність оголошень і операторів, призначених для розв'язання визначеної задачі (модуля).

- Таку функцію потрібно оголосити (і можливо описати) перед першим використанням.

Поняття функції

З використанням функцій у мові С(С++) пов'язані три поняття:

1) означення функції — опис дій, які виконуються функцією;
2) оголошення функції — задання форми звертання до функції;
3) виклик функції — звертання до функції.

Синтаксис мови С (С++) передбачає два способи задання функції користувача:

- означення перед її першим викликом;
- попереднє оголошення, а тоді означення.

Опис функції

Синтаксис означення функції

```
тип_даних ім’я_функції (список_параметрів|void)
{
    тіло_функції;
}
```

`тип_даних` — тип результату, який повертає функція.

- Будь-який тип мови С, в тому числі `void`, а також тип користувача.

`ім’я_функції` — це ідентифікатор, що використовується для оголошення та виклику функції.

- Задається за правилами створення ідентифікаторів у мовах програмування.
- Можна використовувати лише символи латинських літер, арабських цифр і підкреслення.
- Не можна починати ім’я з цифри.
- Великі та малі латинські літери розрізняються.

Опис функції

Круглі дужки `()` — обов'язковий атрибут функції, без якого не можна ні означити, ні використати функцію.

- Рекомендується одразу набирати пару дужок `()`, а тоді в середині в них записувати параметри, якщо такі передбачено.

`список_параметрів` — необов'язковий атрибут задання функції, який описує аргументи, з якими вона може працювати функція.

- Якщо `список_параметрів` подано, то ці параметри називають ***формальними*.**

`тіло_функції` — послідовність описових і виконавчих інструкцій, які визначають дію функції.

- У тілі функції записується послідовність простих чи складених операторів для виконання поставленого перед функцією завдання.
- Можуть бути оголошені необхідні додаткові змінні чи поіменовані константи.

Опис функції

Зауваження

*Функція **НЕ може** бути означеною в тілі іншої функції, в тому числі головної функції* `main()`.

Звертання до функції

Виклик функції

— простий оператор, який може бути самостійною інструкцією.

```
ім’я_функції([список_аргументів])
```

- Якщо функція не описана з типом результату `void`, то звертання до функції може входити у вираз або бути складовою частиною іншого оператора чи інструкції.
- Список аргументів може містити змінні, сталі, посилання, вказівники, вирази.
- Типи у списку аргументів при звертанні до функції повинні відповідати типам у списку параметрів при оголошенні функції та її означенні.
- Функції можуть не містити параметрів, тоді такі функції або використовують глобальні змінні, або змінні, що оголошені в тілі функції.

Аргументи функції

Аргумент — одиниця даних, яка передається з програми у функцію.

- Аргументи дозволяють функції оперувати різними значеннями або виконувати різні дії залежно від значень, що передаються у функцію.

Список формальних параметрів — список аргументів в означенні функції.

- Містить перелік типів аргументів і їх ідентифікаторів:

```
(тип1 ід_зм1, тип2 ід_зм2, тип3 ід_зм3 ...)
```

Список фактичних параметрів — список аргументів при звертанні до функції.

- Коли викликається функція, додатково виділяється пам'ять під її формальні параметри, і кожен формальний параметр ініціалізується відповідним йому значенням фактичного параметра.
- Семантика передачі параметрів ідентична семантиці ініціалізації (тип фактичного параметра зіставляється з типом формального параметра, і виконуються всі стандартні й визначені користувачем перетворення типів).

Аргументи функції

Змінні зі списку формальних параметрів є ***локальними***, тобто видимими лише у тілі функції.

- Під такі змінні в області оперативної пам'яті (призначеної для зберігання коду функції) виділяються комірки для зберігання ***копій*** значень переданих фактичних параметрів.
- Такі параметри можуть змінюватися в тілі функції, але при завершенні роботи функції, ці зміни не торкнуться фактично переданих значень.

Особливості передачі масиву у функцію

- При передачі ***масиву*** передається копія адреси його розташування, тому за цією адресою всі зміни, виконані над елементами масиву, зберігаються при поверненні масиву у точку виклику.
- При передачі одновимірного масиву розмірність вказувати не треба, а при передачі двовимірного чи тривимірного можна не вказувати лише крайню ліву розмірність.

Повернення результату виконання функції

Функції, що не повертають результату

Якщо послідовність виконавчих інструкцій не передбачає повернення жодного значення через ім'я функції або треба повернути два й більше окремих результуючих значення, а також масив, тоді для типу результату функції використовують ключове слово `void`.

Повернення результату виконання функції

Функції, що повертають результат

Функція, що не описана з типом результату `void`, повинна повертати значення за допомогою оператора `return` у одній з двох форм:

```
return вираз;   або   return константа;
```

- Тип значення виразу або константи, що вказаний у `return`, перевіряється на узгодженість з типом, що повертається функцією.
- Функція може містити більше, ніж один оператор `return`.
- Функція виконується до першого з наявних операторів повернення результату.
- Усі інструкції після досягнутого оператора повернення не виконуються.
- За допомогою оператора `return` можна повернути лише одне значення.

Повернення результату виконання функції

Наприклад

```
bool vusokosnuj_rik(unsigned n) //визначає, чи є рік високосним,
     // тобто рік ділиться на 4 за винятком років, які кратні 100
     // (такі роки є високосними, якщо вони ще й діляться на 400)
{
     return (n % 4 || (!(n % 100) && n % 400)) ? false : true;
}

int syma(int a[], int n) //обчислює суму елементів одновимірного масиву
{
     int s = a[0];
     for (int i = 1; i < n; s += a[i++]);
     return s;
}
```

Приклад 1

Написати функцію, яка заповнює цілочисельний масив числами із заданого діапазону. Обчислити суму елементів цього масиву.

Приклад 2

Впорядкувати квадратну матрицю за спаданням сум елементів у її рядках.

Прототип функції

У мові С немає вимоги, щоб означення функції обов'язково передувало її викликові.

Прототип функції

— це попереднє оголошення функції — певний узагальнений опис функції.

Синтаксис

```
тип_даних ім’я_функції ([список параметрів]);
```

- Попереднє оголошення функції інформує компілятор про те, що далі у лістінгу програми буде записано код функції до чи після `main()`.
- `тип_даних`, `ім’я_функції`, `список_параметрів` є подібними до конструкцій при означенні функції.
- У списку формальних параметрів прототипу функції потрібно прописувати типи параметрів, проте імена вказувати необов'язково.

Прототип функції

Наприклад,

```
void creat_mtr(int[][size1], int);
void print_mtr(int[][size1], int);
int syma(int[], int);
void sort_mtr(int[][size1], int);
```

- При використанні прототипів відпадає потреба у чіткому дотриманні порядку означення функцій.
- Єдине, чого слід дотримуватися, — це не задавати одну функцію у тілі іншої.
- Означення функції має такий самий вигляд як і без прототипного оголошення.

Приклад 3

Розглянемо приклад використання функцій створення, виведення, додавання і множення квадратних матриць.

Приклад 4

Уточніть корінь многочлена

$$P_n(x) = a_0x^n + a_1x^{n-1} + a_2x^{n-2} + \cdots + a_{n-1}x + a_n$$

за процедурою Ньютона

$$x_{i+1} = x_i - \frac{P_n(x_i)}{P_n{}'(x_i)}$$

Процес завершити, коли $|x_i - x_{i+1}| < \varepsilon$ (x_0 та ε вводяться з клавіатури).

Приклад 4

Нехай коефіцієнти многочлена $P_n(x)$ задані в одновимірному масиві A розмірності $n+1$. Тоді значення многочлена в точці

$$P_n(x) = a_0x^n + a_1x^{n-1} + a_2x^{n-2} + \cdots + a_{n-1}x + a_n$$

можна обчислити за *схемою Горнера*

$$P_n(x) = \big(\ldots\big((a_0x + a_1)x + a_2\big)x + \cdots + a_{n-1}\big)x + a_n.$$

Знайдемо похідну многочлена $P_n(x)$:

$$P_n{}'(x) = na_0x^{n-1} + (n-1)a_1x^{n-2} + (n-2)a_2x^{n-3} + \cdots + a_{n-1} =$$

$$= b_0x^{n-1} + b_1x^{n-2} + b_2x^{n-3} + \cdots + b_{n-1},$$

де $b_k = (n-k)a_k, k = \overline{0, n-1}$.

Значення похідної $P_n{}'(x)$ в точці x також можна обчислити за *схемою Горнера*

$$P_n{}'(x) = \big(\ldots\big((b_0x + b_1)x + b_2\big)x + \cdots\big)x + b_{n-1}.$$

Приклад 4

Ідея алгоритму

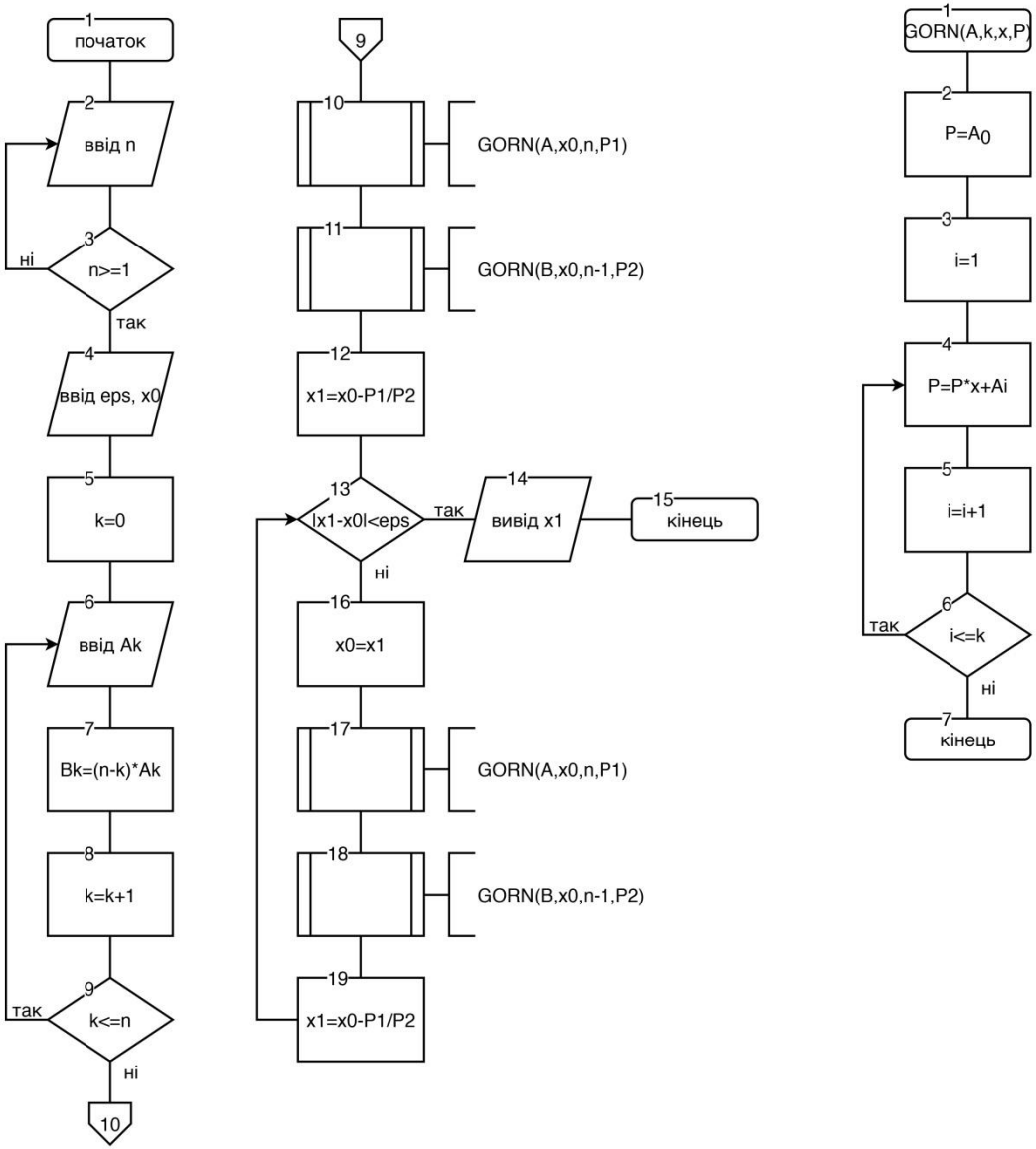
- Сформувати (ввести) одновимірний масив коефіцієнтів многочлена $a_k, k = \overline{0,n}$ та його похідної $b_k = (n-k)a_k, k = \overline{0,n-1}$.
- Обчислити значення многочлена та його похідної в початковій точці x_0 за схемою Горнера.
- Знайти перше наближення кореня многочлена за формулою Ньютона

$$x_1 = x_0 - \frac{P_n(x_0)}{P_n{}'(x_0)}.$$

- Якщо точність ε не досягнуто, тобто не виконується умова $|x_0 - x_1| < \varepsilon$, то обчислити наступне наближення кореня многочлена:

$$x_2 = x_1 - \frac{P_n(x_1)}{P_n{}'(x_1)}.$$

- Цикл послідовного обчислення наближених коренів продовжуємо доти, поки не виконається умова $|x_i - x_{i+1}| < \varepsilon$.


1
початок
2
ввід n
3
n>=1
ні
так
4
ввід eps, x0
5
k=0
6
ввід Ak
7
Bk=(n-k)*Ak
8
k=k+1
9
k<=n
так
ні
10
9
10
GORN(A,x0,n,P1)
11
GORN(B,x0,n-1,P2)
12
x1=x0-P1/P2
13
|x1-x0|<eps
так
14
вивід x1
15
кінець
ні
16
x0=x1
17
GORN(A,x0,n,P1)
18
GORN(B,x0,n-1,P2)
19
x1=x0-P1/P2
1
GORN(A,k,x,P)
2
P=A0
3
i=1
4
P=P*x+Ai
5
i=i+1
6
i<=k
так
ні
7
кінець

Класи пам'яті та простір імен

Кожна змінна (функція), оголошена в програмі, належить до певного ***класу пам'яті***, який встановлює

1. час існування;
2. область видимості;
3. місце зберігання.

Час існування (життя) змінної відображає час, упродовж якого вона існує в пам'яті комп'ютера:

- *статичний* — пам'ять під змінну зберігається на весь час виконання програми;
- *автоматичний* — пам'ять виділяється лише на час виконання блоку, в якому оголошена змінна;
- *динамічний* — пам'ять виділяється і звільняється за потребою під час виконання програми.

Область видимості (дії) змінної визначає частини програми, з яких до неї можливий доступ:

- *локальна* — від місця оголошення змінної *до кінця блоку*, в якому вона оголошена.
- *глобальна* — від місця оголошення змінної *до кінця файлу*, в якому вона оголошена.

Класи пам'яті та простір імен

У С використовують 4 класи пам’яті:

1. автоматичний (auto);
2. регістровий (register);
3. зовнішній (extern);
4. статичний (static).

Керувати характеристиками змінних програміст може двома шляхами:

1) зміною місця оголошення змінної у програмі — клас пам'яті встановлюється компілятором за замовчанням;
2) використанням модифікаторів класу пам’яті (`auto`, `register`, `static`, `extern`).

Класи пам'яті та простір імен

Автоматичний клас пам’яті (auto)

Автоматична змінна має локальну область дії і відома тільки всередині блоку свого оголошення.

- Пам’ять автоматично виділяється при вході у блок і звільняється при виході з нього, тобто змінна знищується.
- Змінна зберігається у стековій (stack) пам’яті.
- Якщо специфікатор класу пам'яті не вказаний, то змінна вважається автоматичною за замовчуванням.
- Початкове значення автоматичної змінної непередбачуване, тому її потрібно явно ініціалізовувати.

Класи пам'яті та простір імен

Регістровий клас пам’яті (register)

є підмножиною класу auto.

- Регістрова змінна відрізняється від автоматичної лише пам'яттю, яка виділяється для її збереження.
- Змінна зберігається у регістрі центрального процесора (при наявності вільних регістрів), тому доступ до цієї змінної набагато швидший, ніж до звичайних змінних, які зберігаються в оперативній пам'яті (auto).
- У випадку відсутності вільних регістрів регістрова змінна стає автоматичною — компілятор самостійно приймає рішення про внесення змінної у регістрову чи стекову пам’ять.
- Регістровою змінною може бути лише дане цілого типу або вказівник.

Класи пам'яті та простір імен

Зовнішній клас пам’яті (extern)

дає змогу оголошувати змінні, дія яких поширюється на всі файли, з яких складається програма.

- Якщо змінна оголошена зі специфікатором `extern`, то це вказівка компілятору, що в якомусь із інших файлів проекту є означення цієї змінної (без `extern`) і надання їй певного значення, яке буде доступне в усіх файлах і функціях проекту.
- При цьому пам’ять під змінну не виділяється. Вона виділяється при означенні змінної.
- При оголошенні змінної зі специфікатором `extern` НЕ можна їй надавати початкового значення.
- За замовчуванням усі функції є типу extern.

Класи пам'яті та простір імен

Статичний клас пам'яті (static)

Статична змінна зберігається до кінця виконання програми.

- Початкове значення такої неініціалізованої змінної є нульовим.
- Якщо змінна оголошена поза будь-яким блоком (функцією), то вона є статичною змінною.
- Якщо змінна оголошена у довільному блоці (функції), то за допомогою модифікатора `static` час її життя можна продовжити на весь час виконання програми, проте її видимість не зміниться (однаково залишиться лише в межах блоку оголошення).

Глобальні змінні

Глобальні змінні

— змінні, що оголошені за межами будь-якої функції, в тому числі функції main().

- Початкове значення таких змінних рівне нулю відповідного типу.
- Такі змінні видимі в межах кожної функції того файлу, в якому вони оголошені, якщо ці функції оголошені пізніше від глобальної змінної.
- Пам'ять під глобальні змінні є ***постійною***, виділяється на початку виконання програми та закріплюється за ними до кінця роботи програми.
- За замовчуванням для глобальних змінних встановлено ***статичний клас пам'яті***.

Локальні змінні

Локальні змінні

— змінні, що оголошені в тілі будь-якої функції, у списку формальних параметрів функції чи у довільному блоці.

- Такі змінні видимі лише в межах блоку оголошення.
- Час життя локальних змінних — це період виконання блоку, де вони оголошені.
- Початкове значення автоматичних змінних є непередбачуване.
- Пам’ять для них виділяється в області стекової пам’яті.
- За замовчуванням для локальних змінних встановлено ***автоматичний клас пам’яті***, тобто змінні автоматично створюються при вході у блок оголошення і знищуються при виході з нього.

Локальні змінні

Локальна змінна, що оголошена в певній функції, НЕ існує в пам'яті комп'ютера доти, доки не відбудеться звертання до цієї функції.

Наприклад

```
void func()
{
     int cile_dane;
     double diysne_dane;
     //інші оператори
}
```

Обмежений час життя змінної дозволяє економити пам'ять під час виконання програми.

Екранування змінних

- Якщо у блоці оголошено локальну змінну з ідентифікатором, що співпадає з іменем глобальної змінної, тоді від моменту оголошення цієї змінної і до кінця блоку за цим іменем буде доступна локальна змінна, а не глобальна.
- Якщо ж змінна неініціалізована, то, як локальна, вона міститиме «непередбачуване значення», а як глобальна — дорівнюватиме 0.
- Щоб використати значення глобальної змінної зі спорідненим ім'ям використовується оператор розширення контексту `::`

Ситуацію, при якій локальна змінна є «більш видимою» від глобальної, називають ***екрануванням****.*

Екранування змінних

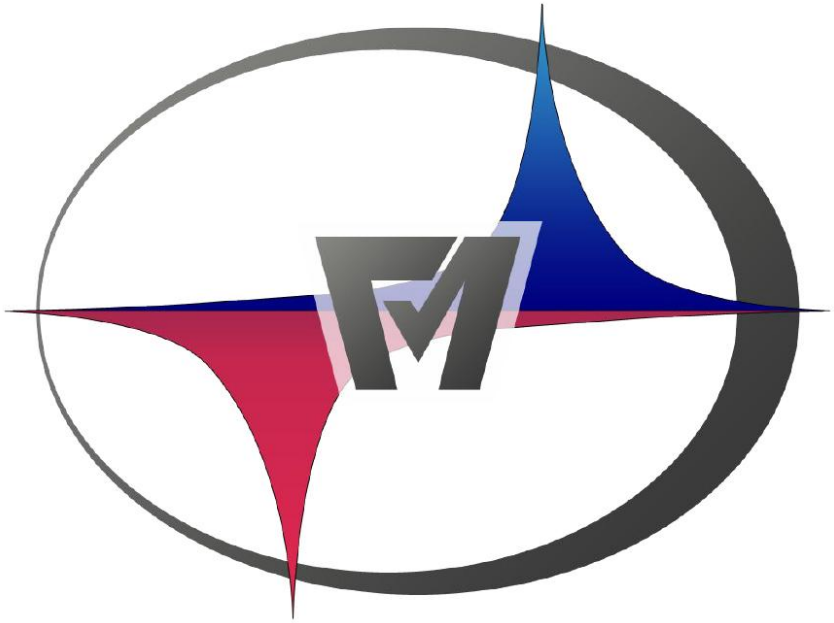
Наприклад

```
#include<iostream>
using namespace std;

int main()
{
     int i, s = 0;                          //змінна i – локальна у функції
     for (int i = 0; i < 10; i++);          //змінна i – локальна у циклі
     cout << "s=" << s << "\ti=" << i << endl; // неініціалізована змінна i
     return 0;
}
```

Приклад 5

Розглянемо приклад створення багатофайлового проекту, який складається з одного заголовкового файлу та двох cpp-файлів.



Кафедра прикладної математики

http://amath.lp.edu.ua