

Класи зберігання



Лекція №5

Дисципліна «Програмування»

2-й семестр



Аспекти зберігання даних

Для зберігання даних у пам'яті мова **C** пропонує **п'ять різних моделей**, або **класів зберігання**.

Апаратний аспект – будь-яке збережене значення знаходиться в фізичній пам'яті.

Для опису ділянки пам'яті застосовується термін **об'єкт**. Об'єкт може зберігати одне або декілька значень.

В певний момент об'єкт може поки не містити збережене значення, але він буде мати правильний розмір для розміщення потрібного значення.

Програмний аспект – програмі потрібен будь-який спосіб доступу до об'єкту.

Вираз, який означає об'єкт, називається **l-значенням**.

```
int entity = 3;
```

```
int *pt = &entity;
```

```
int ranks[10];
```

```
const char *pc = "Це рядковий літерал!";
```



Область видимості

Область видимості описує ділянку або ділянки програми, де можна звертатися до ідентифікатора. Змінна в **C** має одну з наступних областей видимості:

- в межах блоку;
- в межах функції;
- в межах прототипу функції;
- в межах файлу.

Змінні **cleo** і **patrick** мають **область видимості в межах блоку**, що простягається до закривальної фігурної дужки:

```
double blocky(double cleo)
{
    double patrick = 0.0;
    ...
    return patrick;
}
```



Область видимості

Змінні, що оголошені **у внутрішньому блоці**, отримують область видимості, яка обмежена тільки цим блоком:

```
double blocky(double cleo)
{
    double patrick = 0.0;
    int i;

    for(i = 0; i < 10; i++)
    {
        double q = cleo * i;    // початок області видимості для q
        ...
        patrick *= q;
    }                          // кінець області видимості для q
    ...
    return patrick;
}
```



Область видимості

За традицією змінні з областю видимості в межах блоку повинні оголошуватися на початку блоку.

В стандарті **C99** ця вимога була послаблена, і змінні дозволено оголошувати в будь-якому місці блоку.

Одна з нових можливостей пов'язана з оголошенням всередині керуючого розділу циклу **for**.

```
for(int i = 0; i < 10; i++)  
    printf("Можливість C99: i = %d", i);
```

Як частина цієї нової можливості, стандарт **C99** розширив концепцію блоку шляхом включення до неї коду, що керується циклами **for**, **while**, **do...while** або оператором **if**, навіть якщо фігурні дужки при цьому не використовуються.



Область видимості

Область видимості в межах функції застосовуються тільки до міток, що використовуються з операторами **goto**.

Це означає, що коли мітка вперше з'являється у внутрішньому блоці функції, її область видимості простирається на всю функцію.

Якщо б можна було використовувати одну й ту ж саму мітку всередині двох окремих блоків, виникла б плутанина, тому область видимості в межах функції для міток запобігає виникненню такої ситуації.

Область видимості в межах прототипу функції застосовується до імен змінних, що використовуються в прототипах функцій:

```
int mighty(int mouse, double large);
```



Область видимості

Область видимості в межах прототипу функції поширюється від місця визначення змінної до кінця оголошення прототипу.

Це означає, що при обробці аргументу прототипу функції компілятор цікавить тільки тип аргументу.

Якщо вказані імена, то зазвичай вони не грають жодної ролі та не обов'язково повинні збігатися з іменами, які застосовуються у визначенні функції.

Імена грають невелику роль у випадку параметрів, що мають типи масивів змінної довжини:

```
void use_a_VLA(int n, int m, int mas[n][m]);
```

При використанні імен в дужках треба пам'ятати, що це повинні бути імена, які були оголошені раніше в прототипі.



Область видимості

Змінна, визначення якої знаходиться за рамками будь-якої функції, має **область видимості в межах файлу**. Така змінна буде видимою від місця її визначення і до кінця файлу.

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

```
int units = 0;    // змінна з областю видимості в межах файлу
```

```
void critic(void);
```

```
int main(void)
```

```
{  
    ...  
}
```

```
void critic(void)
```

```
{  
    ...  
}
```

Змінна **units** має область видимості в межах файлу та може використовуватися і в функції **main()**, і в функції **critic()**.

Оскільки змінні з областю видимості в межах файлу можуть використовуватися в більш ніж одній функції, вони ще називаються **глобальними змінними**.



Зв'язування

Змінна в **с** має одне з наступних зв'язувань:

- зовнішнє зв'язування;
- внутрішнє зв'язування;
- відсутність зв'язування.

Змінні з областю видимості в межах блоку, функції або прототипу функції не мають зв'язування.

Змінна з областю видимості в межах файлу може мати або внутрішнє, або зовнішнє зв'язування.

Змінна з зовнішнім зв'язуванням може застосовуватися у будь-якому місці багатофайлової програми, а **змінна з внутрішнім зв'язуванням** – де завгодно в одиниці трансляції.



Тривалість зберігання

Область видимості та **зв'язування** описують видимість ідентифікаторів.

Тривалість зберігання характеризує постійність об'єктів, що є доступними за допомогою цих ідентифікаторів.

Об'єкт в **с** має одну з наступних тривалостей зберігання:

- статичну;
- потокову;
- автоматичну;
- виділену.

Якщо об'єкт має **статичну тривалість зберігання**, він існує протягом часу виконання програми.

Змінні з областю видимості в межах файлу мають статичну тривалість зберігання.



Тривалість зберігання

Потокова тривалість зберігання має місце при паралельному програмуванні, коли виконання програми може бути поділено на декілька потоків.

Об'єкт з потоковою тривалістю зберігання існує з моменту його оголошення і до завершення потоку.

Такий об'єкт створюється, коли оголошення, яке в іншому випадку призвело б до створення об'єкту з областю видимості в межах файлу, модифіковане за допомогою ключового слова **_Thread_local**.

Коли змінна оголошена з таким специфікатором, кожен потік отримує власну закриту копію цієї змінної.



Тривалість зберігання

Змінні з областю видимості в межах блоку зазвичай мають **автоматичну тривалість зберігання**.

Пам'ять для цих змінних виділяється, коли потік керування входить до блоку, де вони визначені, та звільняється, коли потік керування покидає цей блок.

Пам'ять, яка використовується для автоматичних змінних, є робочим простором або тимчасовою пам'яттю, яка може застосовуватися багаторазово.

Після завершення виклику функції пам'ять, яку функція використовувала для своїх змінних, може бути задіяна під час виклику наступної функції.

Масиви змінної довжини існують від місця свого оголошення і до кінця блоку, а не від початку блоку і до свого кінця.



Тривалість зберігання

Змінна може мати область видимості в межах блоку, але **статичну тривалість зберігання**. Щоб її створити, треба оголосити її всередині блоку та додати ключове слово **static**:

```
void more(int number)
{
    int index;
    static int ct = 0;
    ...
    return 0;
}
```

Тут змінна **ct** зберігається в статичній пам'яті. Вона існує з моменту завантаження програми в пам'ять і аж до завершення виконання програми. Але область видимості **ct** обмежена блоком функції **more()**. Тільки під час виконання цієї функції програма може використовувати змінну **ct** для доступу до об'єкту, який вона позначає.



Класи зберігання

Клас зберігання	Тривалість зберігання	Область видимості	Зв'язування	Оголошення
Автоматичний	Автоматична	В межах блоку	Ні	В блоці
Регістровий	Автоматична	В межах блоку	Ні	В блоці з зазначенням ключового слова register
Статичний з зовнішнім зв'язуванням	Статична	В межах файлу	Зовнішнє	За рамками усіх функцій
Статичний з внутрішнім зв'язуванням	Статична	В межах файлу	Внутрішнє	За рамками усіх функцій з зазначенням ключового слова static
Статичний без зв'язування	Статична	В межах файлу	Ні	В блоці з зазначенням ключового слова static



Автоматичні змінні

Змінна, що належить до автоматичного класу зберігання, має автоматичну тривалість зберігання, область видимості в межах блоку і не має зв'язування.

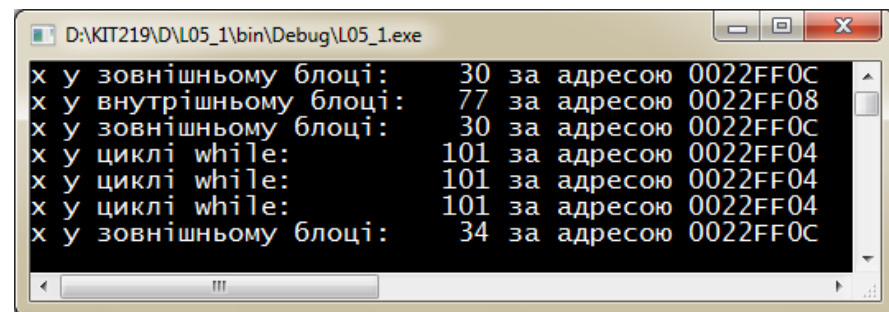
За замовчуванням будь-яка змінна, яка оголошена в блоці або в заголовку функції, належить до автоматичного класу зберігання. Однак ви можете сформулювати свої наміри, явно вказавши ключове слово **auto**:

```
int main(void)
{
    auto int plox;
    ...
}
```

Ключове слово **auto** називається **специфікатором класу зберігання**. Не застосовуйте **auto** в якості специфікатора класу зберігання, щоб домогтися більшої сумісності між **C** і **C++**.

Автоматичні змінні

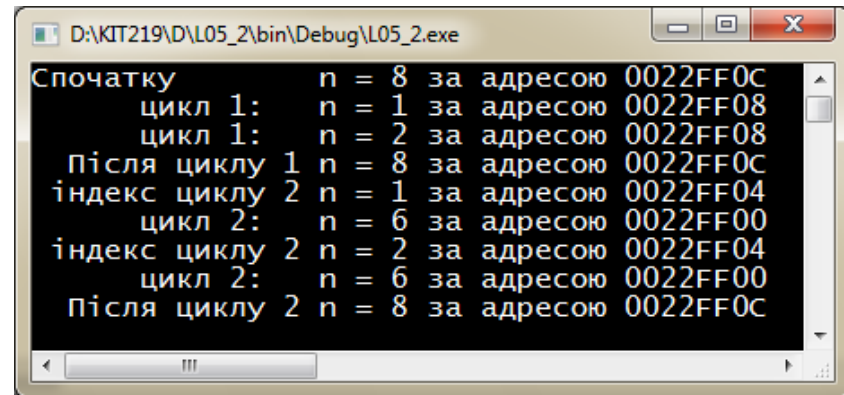
```
#include <stdio.h>
#include <windows.h>
int main(void)
{
    int x = 30; // перша змінна x
    SetConsoleOutputCP(1251);
    printf("x у зовнішньому блоці: %4d за адресою %p\n", x, &x);
    {
        int x = 77; // друга змінна x, що приховує першу x
        printf("x у внутрішньому блоці: %4d за адресою %p\n", x, &x);
    }
    printf("x у зовнішньому блоці: %4d за адресою %p\n", x, &x);
    while(x++ < 33) // перша змінна x
    {
        int x = 100; // третя змінна x, що зв'язує першу x
        x++;
        printf("x у циклі while: %4d за адресою %p\n", x, &x);
    }
    printf("x у зовнішньому блоці: %4d за адресою %p\n", x, &x);
    return 0;
}
```



```
D:\KIT219\D\L05_1\bin\Debug\L05_1.exe
x у зовнішньому блоці:    30 за адресою 0022FF0C
x у внутрішньому блоці:   77 за адресою 0022FF08
x у зовнішньому блоці:    30 за адресою 0022FF0C
x у циклі while:         101 за адресою 0022FF04
x у циклі while:         101 за адресою 0022FF04
x у зовнішньому блоці:    34 за адресою 0022FF0C
```


Блоки без фігурних дужок

```
#include <stdio.h>
#include <windows.h>
int main(void)
{
    int n = 8;
    SetConsoleOutputCP(1251);
    printf("Спочатку          n = %d за адресою %p\n", n, &n);
    for(int n = 1; n < 3; n++)
        printf("        цикл 1:    n = %d за адресою %p\n", n, &n);
    printf("    Після циклу 1 n = %d за адресою %p\n", n, &n);
    for(int n = 1; n < 3; n++)
    {
        printf("індекс циклу 2 n = %d за адресою %p\n", n, &n);
        int n = 6;
        printf("        цикл 2:    n = %d за адресою %p\n", n, &n);
        n++;
    }
    printf("    Після циклу 2 n = %d за адресою %p\n", n, &n);
    return 0;
}
```





Ініціалізація автоматичних змінних

Автоматичні змінні не ініціалізуються до тих пір, поки ви не зробите це явно.

```
int main(void)
{
    int repid;
    int tents = 5;
    ...
}
```

Змінна **tents** ініціалізується значенням **5**, але **repid** отримує значення, яке раніше знаходилось в області пам'яті, що виділяється під цю змінну. Неможна розраховувати, на те, що цим значенням буде **0**.

```
int main(void)
{
    int ruth = 1;
    int rance = 8 * ruth;
    ...
}
```

Ви можете ініціалізувати автоматичну змінну неконстантним виразом за умови, що усі задіяні в ньому змінні були визначені раніше.

```
// використовується змінна ruth,
// яка була визначена раніше
```



Регістрові змінні

Регістрові змінні мають область видимості в межах блоку, не мають зв'язування та мають автоматичну тривалість зберігання.

Змінна оголошується з використанням специфікатора класу зберігання **register**:

```
int main(void)
{
    register int quick;
    ...
}
```

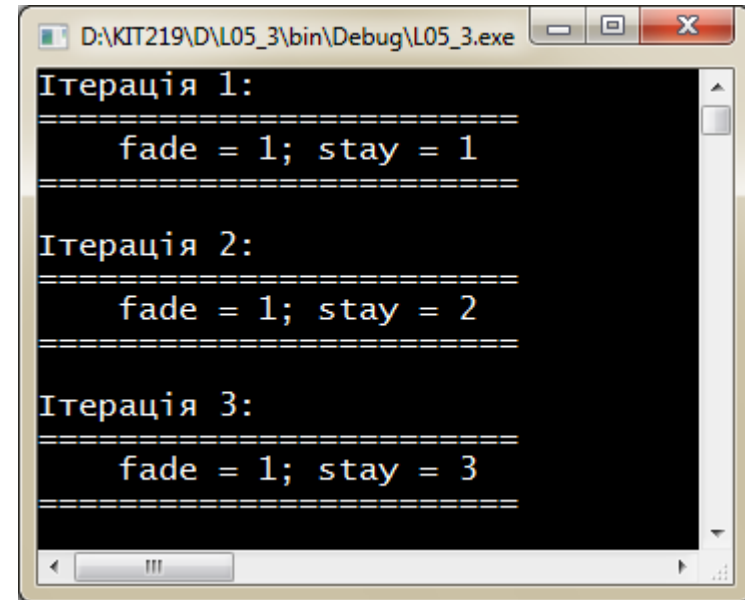
Ви маєте змогу зробити запит, щоб формальні параметри були регістровими змінними. Для цього просто скористайтесь ключовим словом **register** в заголовку функції:

```
void macho(register int n)
```

Статичні змінні з областю видимості в межах блоку (без зв'язування)

```
#include <stdio.h>
#include <windows.h>
void trystat(void);
int main(void)
{
    int count;
    SetConsoleOutputCP(1251);
    for(count = 1; count <= 3; count++)
    {
        printf("Ітерація %d:\n", count);
        trystat();
    }
    return 0;
}

void trystat(void)
{
    int fade = 1;
    static int stay = 1;
    printf("    fade = %d; stay = %d\n", fade++, stay++);
}
```



```
D:\KIT219\D\L05_3\bin\Debug\L05_3.exe
Ітерація 1:
=====
    fade = 1; stay = 1
=====
Ітерація 2:
=====
    fade = 1; stay = 2
=====
Ітерація 3:
=====
    fade = 1; stay = 3
=====
```



Статичні змінні з зовнішнім зв'язуванням

Статична змінна з зовнішнім зв'язуванням має область видимості в межах файлу, зовнішнє зв'язування і статичну тривалість зберігання.

Такий клас іноді називають **зовнішнім класом зберігання**, а змінні цього типу – **зовнішніми змінними**.

Зовнішня змінна створюється шляхом розміщення оголошення за рамками усіх функцій.

Відповідно до документації, зовнішня змінна може додатково бути оголошена всередині функції, в якій вона використовується, з застосуванням ключового слова **extern**.

Якщо будь-яка зовнішня змінна визначена в одному файлі початкового коду та використовується в іншому файлі початкового коду, то **оголошення цієї змінної в іншому файлі з ключовим словом extern є обов'язковим**.



Статичні змінні з зовнішнім зв'язуванням

```
int Errupt;           // змінна, що має зовнішнє визначення
double Up[100];       // масив, що має зовнішнє визначення
extern char Coal;     // обов'язкове оголошення, якщо
                     // Coal визначається в іншому файлі

void next(void);

int main(void)
{
    extern int Errupt;   // необов'язкове оголошення
    extern double Up[];  // необов'язкове оголошення
}

void next(void)
{
    ...
}
```



Ініціалізація зовнішніх змінних

Зовнішні змінні можуть ініціалізуватися явно.

На відміну від автоматичних, **зовнішні змінні за замовчуванням ініціалізуються нулем**, якщо ви не ініціалізували їх.

Це правило можна застосовувати також до елементів зовнішньо визначеного масиву.

Для ініціалізації змінних з областю видимості в межах файлу можна використовувати тільки константні вирази, що відрізняється від випадку автоматичних змінних.

```
int x = 10;           // допустимо, 10 - це константа
int y = 3 + 20;       // допустимо, константний вираз
size_t z = sizeof(int); // допустимо, константний вираз

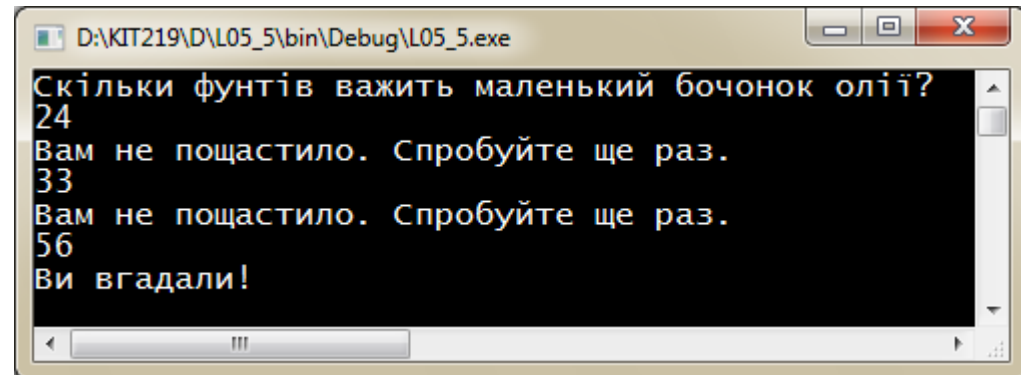
int x2 = 2 * x;       // недопустимо, x - це змінна
```



Використання зовнішньої змінної

```
#include <stdio.h>
#include <windows.h>
int units = 0;           // зовнішня змінна
void critic(void);
int main(void)
{
    SetConsoleOutputCP(1251);
    extern int units;    // необов'язкове повторне оголошення
    printf("Скільки фунтів важить маленький бочонок олії?\n");
    scanf("%d", &units);
    while(units != 56)
        critic();
    printf("Ви вгадали!\n");
    return 0;
}

void critic(void)
{
    printf("Вам не пощастило. Спробуйте ще раз.\n");
    scanf("%d", &units);
}
```





Статичні змінні з внутрішнім зв'язуванням

Змінні з цим класом зберігання мають статичну тривалість зберігання, область видимості в межах файлу і внутрішнє зв'язування.

Така змінна створюється шляхом її визначення поза будь-яких функцій (як і у випадку зовнішньої змінної) з вказуванням специфікатора класу зберігання **static**:

```
static int svil = 1;    // статична змінна,  
                        // внутрішнє зв'язування
```

```
int main(void)  
{  
    ...  
}
```