Класи пам'яті даних

Клас пам'яті, час існування та видимість об'єкта

Кожен об’єкт програми (змінна, функція,…) має свій тип і клас пам’яті. Тип визначає обсяг пам’яті для об’єкта і операції, що можуть виконуватись над об’єктом.

Клас пам'яті задає місце розташування об'єкта в оперативній пам'яті та встановлює для нього час існування, тобто час, протягом якого об‘єкт зберігається в оперативній пам‘яті, і область видимості, яка визначає ту частину програми, де можна використовувати цей об‘єкт. На відміну від типу, клас пам’яті можна явно не вказувати, тоді він встановлюється компілятором за місцем оголошення об'єкта.

За часом існування об‘єкти поділяють на три групи:

- глобальні (або статичні) – існують протягом усього часу виконання програми;
- локальні – існують лише при виконанні блоку, де вони оголошені;
- динамічні – час існування змінюється програмою.

Глобальні дані зберігаються в окремій області оперативної пам’яті – сегменті даних.

Локальні дані – в області, що називається стеком.

Динамічні дані – в області динамічної пам’яті.

За правилами замовчування змінні, описані всередині блоку, та формальні параметри функцій мають локальний час існування. Змінні, описані зовні всіх блоків (між функціями) мають глобальний час існування.

(Блоком вважається тіло функції, а також внутрішня група описів у фігурних дужках)

Функції в С – програмах можна описувати тільки на зовнішньому рівні, тому всі функції мають глобальний час існування.

За областю видимості (або областю дії) об‘єкти ділять на три групи:

- глобальні – видимі в межах усієї програми;
- частково глобальні – видимі в межах одного програмного файла;
- локальні – видимі в блоці, де оголошено даний об ‘єкт.

Область дії глобальних і локальних змінних

Переважно змінні програми мають локальний час існування та локальну видимість. Глобальними оголошують окремі змінні, призначені для спільного використання. Правила:

- змінні зовнішнього рівня (між функціями) є частково глобальними з областю дії від точки оголошення до кінця файла;
- змінні внутрішнього рівня (всередені блоку) і формальні параметри функцій мають область дії від точки оголошення до кінця функції (блоку);

- змінні - параметри прототипу функції видимі в межах цього прототипу;
- якщо ім’я внутрішньої змінної збігається з іменем зовнішнього об’єкта, в межах блоку внутрішня змінна “закриває” зовнішню, а за межами блоку дія зовнішнього об’єкта відновлюється;
- клас пам’яті змінних можна встановити явно за допомогою специфікаторів.

Всі глобальні та статичні змінні автоматично ініціалізуються нульовими значеннями, а для локальних це не виконується.

```
#include …
  int n;                    //частково глобальна змінна

  int load(int n);          //n - локальна для прототипу

  int main(int a)           //a - локальна для функції
  {
        int i = 0;          //i - локальна для блоку
        …
   }
  int load(int a)
 {
        int n = 0;          //n - ”закриває” зовнішню змінну
        …                   //   з такою назвою в межах блоку
   }
```

Специфікатори класів пам'яті

Специфікатори застосовують тільки тоді, коли потрібно змінити стандартний клас пам‘яті об‘єкта, інакше він встановлюється за правилами замовчування.

Синтаксис:

специфікатор_класу_пам‘яті ***тип*** *ім‘я_змінної*

Специфі-катор	Клас пам'яті	Застосування		Час існування	Область дії	Автоматична ініціалізація
		локальні змінні	глобальні змінні			
auto	автоматичний	так	ні	локальний	локальна	відсутня
register	регістровий	так	ні	локальний	локальна	відсутня
static	статичний	так	так	глобальний	локальна / частково-глобальна	ініціалізація нулем
extern	зовнішній	так	так	глобальний	частково-глобальна	ініціалізація заборонена

Специфікатори локальних змінних:

- auto - клас за замовчуванням;
- register - зберігання змінної у регістрі процесора, дає змогу істотно скоротити час звертання до змінних, за відсутності вільних регістрів змінна буде опрацьовуватись як змінна з класом auto; така змінна не має адреси;
- static - статичні змінні існуються протягом усього часу виконання програми, проте областю їх дії залишається той блок, у якому вони оголошені;
- extern - змінна є посиланням на глобальну змінну з тим самим іменем і типом, описану далі в тексті програми або в іншому програмному файлі.

Специфікатори глобальних змінних:

- static - областю дії даної змінної буде частина програми від точки опису до кінця файлу; в інших файлах програми ця змінна недоступна;
- extern - змінні є посиланнями на відповідні зовнішні змінні та роблять ці змінні видимими (оголошеними) в межах поточного файлу.

Багатофайлові програми

Складні великі програми доцільно поділити на декілька програмних файлів, кожен з яких можна програмувати та компілювати автономно. У програмах, що поділені на окремі текстові файли, функції розривати не можна. Текст з файлу вставляється в програму в тому місці, де була записана директива ***#include*** . Дві форми запису

#include < *ім'я_файла* ***>*** файл зберігається в спеціальному каталозі середовища програмування;

#include *“ ім'я_файла“* пошук файлу розпочинається з поточного активного каталога.

Для створення багатофайлових програм створюють заголовні файли ***. h*** (прототипи функцій, шаблони структур, користувацькі типи тощо) і підключають ці файли до кожного програмного файла.

Правила видимості у багатофайлових програмах

- щоб звернутись до глобальної змінної, описаної в іншому файлі, необхідно в поточному файлі оголосити цю змінну ***extern*** ;
- функції є глобальними об’єктами, вони описуються тільки один раз і областю їх дії є вся програма;
- прототипи функцій є частково видимими, вони діють у межах одного програмного файла.

Функція описується один раз у якомусь із файлів, для звертання до неї з інших файлів треба попередньо вказати прототип функції. В кожному файлі програми треба оголошувати прототипи всіх необхідних функцій:

- відповідні заголовні файли бібліотечних функцій ***< *.h >*** повинні бути підключені до кожного файлу;
- заголовний файл, в якому записані прототипи користувацьких функцій підключають до кожного програмного файлу.

В інтегрованому середовищі **Borland C** підтримується робота з багатофайловими програмами: компанування єдиного виконавчого коду програми з окремих файлів завдяки створенню проекту програми. Через пункт меню **Project** вказують ім'я проекту, наповнюють проект іменами складових файлів (можуть бути текстовими ***.c** або бінарними ***.obj**). Проекти зберігаються як файли ***.prj**

Проводиться компіляція текстових файлів проекту, після чого компанувальник формує з отриманого набору об'єктних файлів виконавчий файл (exe – код) програми.

Приклад

Бібліотека для роботи з комплексними числами

```
        /*   complex.h   */
typedef struct
  {    double a,b;     } complex_t;
complex_t mul (complex_t x, complex_t y);
        /*    complex.c   */
#include "complex.h"
complex_t mul (complex_t x, complex_t y)
{
   complex_t t;
   t.a = x.a * y.a - x.b * y.b;
   t.b = x.a * y.b + x.b * y.a;
  return t;
}
```

Використання бібліотеки

```
/*    test_complex.c */
 #include <stdio.h>
 #include "complex.h"
 int main()
 {
    complex_t x = {1,2};
    complex_t y = {3,4};
    complex_t z = mul(x,y);
     printf ("z = (%lf, %lf)\n", z.a, z.b );
     return 0;
 }
```