Учебный Центр компьютерного обучения «СПЕЦИАЛИСТ» при МГТУ им. Н.Э.Баумана

Язык программирования C

В.Г.Тетерин – Microsoft Solution Developer (Visual C++)
teterin@specialist.ru

Центр компьютерного обучения «СПЕЦИАЛИСТ» при МГТУ им. Н.Э.Баумана®

МОДУЛЬ 8

КЛАССЫ ПАМЯТИ

Модуль 8. Классы памяти

- Время жизни и область видимости объекта
- Декларации на внутреннем и внешнем уровнях
- Описатели – **`auto`**, **`register`**, **`static`**, **`extern`**
- Динамическое распределение памяти
- Определяемые типы typedef

Время жизни и область видимости объекта

- **Определение** идентификаторов в программе имеет следующий синтаксис:

описатель класса памяти	квалификаторы и модификаторы	основной тип	идентификатор	...

- С каждым идентификатором связаны два атрибута:
  - класс памяти
  - тип.
- Класс памяти определяет:
  - местонахождение,
  - способ доступа,
  - время существования

  памяти, сопоставленной с некоторым идентификатором,
- Тип определяет:
  - структурную организацию этой памяти,
  - смысл значений, хранящихся в этой памяти.

Время жизни и область видимости объекта (продолжение)

- Существует четыре описываемых класса памяти:
  - внешняя
  - статическая
  - автоматическая
  - память на регистрах (или в кеше).
- **Внешние** и **статические** объекты существуют и сохраняют свои значения на протяжении всего времени выполнения программы
  - и различаются между собой возможностью доступа к ним из различных частей программы.
- **Автоматические** и **регистровые** объекты создаются и существуют только внутри блока, в котором они описаны, и уничтожаются при выходе из этого блока,
  - причем, регистровые объекты хранятся в регистрах процессора, если это возможно, или в кэше
  - автоматические всегда хранятся в стеке программы.

Время жизни и область видимости объекта (продолжение)

- В языке С не обязательно компилировать сразу всю программу:
  - исходный текст программы может храниться в нескольких файлах,
  - заранее откомпилированные программы могут загружаться из библиотек,
  - имеется и возможность "срастить" несколько файлов с исходными текстами в единый файл с помощью директивы препроцессора **`#include`** на этапе компиляции.
- В таких условиях важно обеспечить корректность использования идентификаторов в различных частях программы.
- Можно говорить о двух видах областей действия идентификаторов.
  - Первый вид - *лексическая область действия* (область "видимости")
    - это фрагмент программы, где можно пользоваться идентификатором, не рискуя получить диагностическое сообщение "неописанный идентификатор".
  - Второй вид - *область действия внешних имен*
    - в этом случае речь идет о правилах, определяющих, ссылаемся ли мы на один и тот же объект при употреблении некоторого идентификатора.

Декларации на внутреннем и внешнем уровнях

- При рассмотрении областей действия различают два способа описания идентификаторов:
  - *внутреннее (локальное)* описание, когда идентификатор описывается внутри некоторого блока - части программы между открывающей и закрывающей фигурными скобками (в том числе внутри тела функции);
  - *внешнее (глобальное)* описание, когда идентификатор описан вне всех блоков или функций программы.
- Идентификатор, описанный внутри блока, известен только в этом блоке (*локальный* идентификатор).
- Идентификатор, описанный на самом внешнем уровне, известен от места появления этого описания до конца входного файла, в котором он описан (*глобальный* идентификатор).
- Идентификаторы - *формальные аргументы функции* - известны только внутри этой функции.
- Также только внутри функции известны *метки*, т.е. идентификаторы, на которые может ссылаться оператор перехода.

Декларации на внутреннем и внешнем уровнях (продолжение)

- Во всех случаях, если идентификатор явно описывается *внутри* блока, включая и блок представляющий собой функцию, то действие всех внешних по отношению к данному блоку описаний этого идентификатора приостанавливается до конца блока.
  - Это означает, что при описании в блоке локального объекта, идентификатор которого совпадает с идентификатором внешнего по отношению к блоку объекта, локальное описание "перекрывает" внешнее и конфликта между ними не возникает.
  - В блоке создается, существует и изменяет свои значения *новый локальный объект*, имеющий свои собственные атрибуты, но эти изменения никак не отражаются на внешнем объекте, который в блоке неизвестен и сохраняет свое значение неизменным.
  - По выходе из блока локальный объект уничтожается, и в силу вновь вступает описание внешнего объекта с его атрибутами и *сохраненным* значением.
- Если программа разбита на два или более отдельно транслируемых файлов, то связь между ними может осуществляться только через *внешние* (*глобальные*) идентификаторы.
  - При этом внешний идентификатор, описанный в одном файле, может быть доступен или недоступен из других файлов в зависимости от его *класса памяти*.

Описатель auto

- Идентификатор, описанный на локальном уровне, по умолчанию получает класс памяти **auto**.

```
#define N 10


void f2(int b)
{
    int a = b;
    …
    ++a; b = a+2;
}
void f1(void)
{
    int a = 0, arr[N];
    auto int b;
    …
    ++a; b = a+1;
    f2(a); f2(b);
}
```

Описатель extern

- Идентификатор, описанный на глобальном уровне, по умолчанию получает класс памяти **extern**.

```
#define N 10
int b, arr[N];    //определение

void f2(int b)
{
    int a = b;
    …
    ++a; b = a+2;
}
void f1(void)
{
    int a = 0;
    …
    ++a; b = a+1;
    f2(a); f2(b);
}
```

```
#define N 10
extern int b, arr[];    //объявление
extern void f2(int);
void f4(int b)
{
    int a = b;
    …
    ++a; b = a+4;
}
void f3(void)
{
    int a = 0;
    …
    ++a; b = a+3;
    f2(a); f4(b);
}
```

Описатель extern (продолжение)

- Для согласования описаний глобальных идентификаторов используют собственные заголовочные файлы (header файлы).

my_header.h

```
#define N 10
extern int b, arr[];    //объявления
extern void f1(void);
extern void f2(int);
       void f3(void);
       void f4(int);
```

```
#include "my_header.h"
int b, arr[N];    //определение
void f1(void)
{
     …
}
void f2(int b)
{
     …
}
```

```
#include "my_header.h"

void f3(void)
{
     …
}
void f4(int b)
{
     …
}
```

Описатель static

- Идентификатор с классом памяти **static** может быть описан на локальном или глобальном уровне.

```
static int a=1;
void f2(void)
{
   static int b = 0;
   int a = 0;
   ++a; ++b;
}
void f1(void)
{
    …
    f2(); f2();
}
```

```
static int a, b;
static void f4(void)
{
    …
    ++a; b = a+1;
}
void f3()
{
    …
    ++a; b = a+1;
    f4(); f4();
}
```

Описатель register

- Описатель класса памяти **register** может использоваться только на локальном уровне.
  - Он является лишь «просьбой» компилятору оптимизировать скорость доступа к значению указанного идентификатора – разместить его в регистре ЦП (или в кэше).
  - Компилятор без предупреждения может присвоить ему класс памяти **auto** и тем самым разместить его на стеке.

```
void f2(register int a)
{
   …
}
void f1(void)
{
   int a = 10;
   register int b = a+1;
     …
   f2(a); f2(b);
}
```

Динамическое распределение памяти

- Память для *внешних* и *статических* объектов программы выделяется *статически* – перед запуском программы.
- Память для *автоматических* объектов программы распределяется *динамически* в программном стеке:
  - при входе потока управления в блок память выделяется,
  - при выходе из блока – освобождается.
  - Особенность данного способа управления памятью – его *автоматизм*, так как за правильность выделения и своевременность освобождения блоков памяти полностью отвечает компилятор.
- Стандартная библиотека функций языка С предоставляет и альтернативную возможность – набор функций для «ручного» управления свободной памятью системы – так называемой «кучей» (*heap*).
  - Этот способ обладает большой гибкостью, но вся ответственность за правильное управление памятью при этом возлагается на программиста (будет рассмотрен позднее).

Определяемые типы typedef

- Ключевое слово **typedef** предоставляет удобную возможность введения псевдонимов для ранее описанных типов данных, как встроенных, так и определенных программистом.

typedef	существующий тип	идентификатор (псевдоним)	;

  - Например:

```
typedef unsigned char byte;

typedef unsigned short word;

typedef unsigned long dword;

typedef unsigned long ulong;

typedef char string[32];

string client_name_list[100];
```

Итоги

- В этом модуле Вы изучили:
  - Влияние способа декларации идентификаторов на время их жизни и область видимости
  - Описатели класса памяти – **`auto`**, **`register`**, **`static`**, **`extern`** и семантику их применения в программе
  - Понятие статического и динамического распределения памяти
  - Способы создания многофайловых проектов на языке C
  - Применение typedef для упрощения описаний сложных типов

Вопросы?

- В.Г.Тетерин – Microsoft Solution Developer (Visual C++)
- teterin@specialist.ru

ПРИЛОЖЕНИЕ

ЗАДАЧИ

Задачи

1. Используя условия задачи 12 из модуля 6 и задачи 4 из модуля 7, реализуйте в виде многофайлового проекта решение задачи о студентах
   - Выделите в отдельные модули интерфейсные функции и функции, выполняющие обработку данных
   - Реализуйте подходящее меню
2. Аналогично задаче 1 объедините решение задач 7 и 9 из модуля 7