Курс «Основы программирования»

Федорук Елена Владимировна ст. преподаватель каф РК-6 МГТУ им.Н.Э.Баумана

Лекция №8

Обзор классов памяти

Класс памяти - характеристика переменной, которая определяет способ ее хранения.

Язык Си реализует следующие классы памяти:

- автоматический
- регистровый
- внешний
- статический

Характеристики классов памяти:

- *область действия* — часть программы, в которой определено имя;
- *время жизни* — промежуток времени, в течение которого содержимое переменной остается правильным;
- размещение связанных с идентификатором значений в памяти.

В табл. 1 обобщаются характеристики классов памяти языка Си. Правильное использование классов памяти улучшает структуру, модульность, повышает удобочитаемость и скорость работы программы.

Таблица 1

Класс	Область действия	Время жизни	Место хранения	Использование для массивов, структур	Значение по умолчанию
Автоматический	Блок	Исполнение блока	Стек	Да	Неопределено
Регистровый *	Блок	Исполнение блока	Машинный регистр	Нет	Неопределено
Внешний **	От объявления до конца файла	Исполнение программы	Область данных	Да	0
Статический внешний ***	От объявления до конца файла	Исполнение программы	Область данных	Да	0
Статический внутренний	Блок	Исполнение программы	Область данных	Да	0

Примечание 1

* — повышение быстродействия
** — доступность из других файлов
*** — недоступность из других файлов

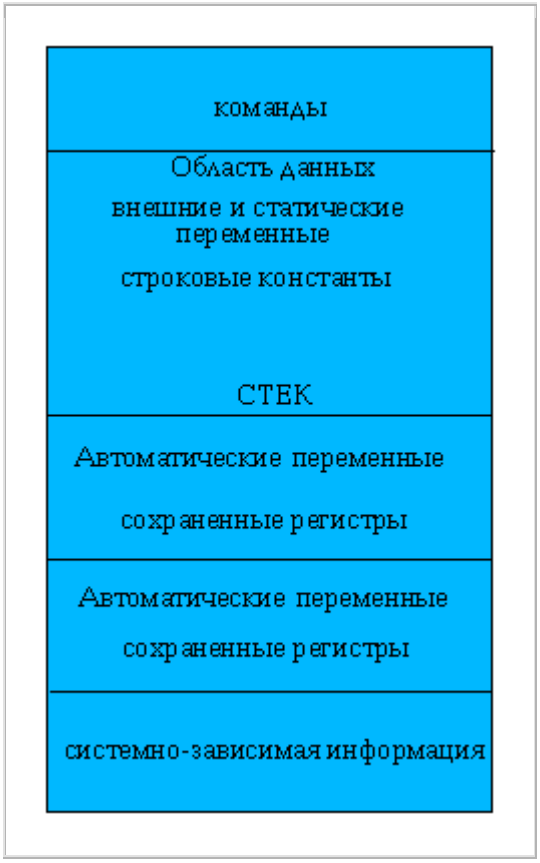

Рис. 1. Размещение данных различных классов памяти

На рис. 1 показано, как типичный компилятор Си размещает команды и данные компилируемой программы на языке Си. Детали зависят от системы. Например, в некоторых системах стек растет от больших адресов к меньшим, в других справедливо обратное. Команды могут размещаться после данных или наоборот. Стек и области данных могут динамически расширяться посредством системных вызовов.

Раздел команд содержит машинный код, определяющий логику программы. На некоторых машинах этот раздел защищен от записи, так как программы не могут сами себя изменять.

Объектам, размещенным в области данных, назначается определенное расположение в этой области, остающееся неизменным во время выполнения этой программы (при использовании виртуальных адресов они имеют один и тот же относительный адрес). В этой области размещаются внешние и статические переменные. Если они не инициализируются в программе, то они устанавливаются в нуль перед выполнением программы и "остаются живыми" в течение времени выполнения программы. Строковые константы, например, заключенные в кавычки строки в вызове функции **`printf()`**, также запоминаются в этой области.

Стек является динамической областью данных. Память в стеке многократно используется повторно при вызовах функций и выходе из них. На стеке размещаются автоматические переменные и параметры функции.

Автоматический класс памяти

Автоматическая переменная — переменная, которая объявляется внутри блока.

Областью действия автоматической переменной является блок, в котором она объявлена. Автоматическая переменная создается динамически (автоматически) при входе в блок и прекращает существование при выходе из него.

Область действия и время жизни параметра функции такие же, как и у автоматической переменной.

В программе может быть несколько совпадающих идентификаторов. Тем не менее, две переменные, объявленные в одном и том же блоке, не могут иметь одинаковых имен. Если переменная во внутреннем блоке имеет то же имя, что и объявленная во внешнем блоке переменная, операторы внутри блока ссылаются на переменную, объявленную во внутреннем блоке.

Пример 1

```
int main()
{
void print_cube(int);
int x;                    /* известно только в main() */

x = getchar();
if ( x == 'c')
  {
   float x;               /* известно в этом блоке */
   x = .75;               /* ссылается на float x */
  }
print_cube(100);
}
void print_cube(int x)       /* известно только в print_cube()*/
{
   printf ("%d\n", x*x*x);
}
```

Примечание 1

На практике лучше избегать таких ситуаций, так как они могут быть причиной ошибок.

Внешний класс памяти

Внешняя переменная — переменная, которая определяется извне функции. Она доступна всем функциям, определенным далее в файле. В отличие от автоматической переменной, внешняя по умолчанию имеет нулевое значение. Элементы внешнего массива инициализируются нулевыми значениями.

Преимущество внешних переменных в том, что, в отличие от автоматических переменных, к ним можно обратиться из более чем одной функции. При использовании внешних переменных программа работает быстрее, чем при передаче в функцию длинного списка параметров. В дополнение к этому, внешние переменные обеспечивают способ разделения данных между функциями, не вызывающими друг друга.

Тем не менее, не все переменные могут быть сделаны внешними. Для улучшения структуры программы и повышения ее читаемости лучше сделать переменную автоматической, если она используется только внутри функции или блока. Кроме того, поскольку внешние переменные доступны всем функциям, требуется предпринимать усилия, чтобы избежать конфликтующих изменений значений этих переменных.

Подобно другим идентификаторам, имена внешних переменных могут содержать буквы, цифры и символ подчеркивания, но не могут начинаться с цифры. Количество значащих символов в имени внешней переменной, зависит от системы.

Примечание 1

Некоторые компиляторы не делают различий между строчными и прописными буквами во внешних идентификаторах. Рекомендуется использовать строчные буквы.

Пример 1

```
int status;          /*Внешние переменные, доступные*/
char systname[20];   /*всем функциям, расположенным ниже*/

void fillbuf(void), emptybuf(void);
int main()
{
 fillbuf();
 ...
 emptybuf();
 ...
}
char buf[1024];       /*Внешняя переменная, доступная*/
void fillbuf(void)    /*функциям fillbuf и emptybuf*/
{
 ...
}
```

```
void emptybuf(void)
{
 ...
}
```

Множественные исходные файлы

Программы на языке Си часто располагаются в нескольких исходных файлах. Одним из примеров является операционная система UNIX, исходные файлы которой функционально структурированы. Существуют файлы, содержащие программы драйверов устройств, управления процессами, файловой системой и т.д. Использование нескольких исходных файлов ускоряет редактирование, компиляцию и отладку. Оно облегчает структуризацию программы и организацию коллективной работы.

Для внешних переменных существует отличие между определением и объявлением.

Внешнее определение переменной (например, **`int status;`**) выделяет область памяти под переменную.

Внешнее объявление переменной (например, **`external int status;`**) памяти не выделяет, оно информирует компилятор о том, что переменная определена где-то еще. Это предотвращает выдачу сообщения об ошибке "неопределенный идентификатор" при использование переменной в файле, в котором она не определена.

Для функции объявление класса **`external`** не требуется, так как имя функция всегда определено как внешний идентификатор.

Пример 2

```
/* Файл one.c */

int status;                 /*определения внешних*/
char sysname[20];           /*переменных*/
void fillbuf(void);

main ()
{
   fillbuf();
   . . .
   if (...)
     emptybuf();
}

/* Файл two.c */

external int status;            /*объявления внешних*/
external char sysname[20];      /*переменных*/
char buf[1024];                 /*определение внешней переменной*/
```

```
void emptybuf(void);
void fillbuf(void)
{
   ...
}
void emptybuf(void)
{
  fillbuf();
}
```

Для создания выполнимого файла программы надо выполнить совместную компиляцию этих двух файлов, используя, например, следующую команду операционной системы UNIX:

```
$ cc one.c two.c
```

Статический класс памяти

Статическая переменная — переменная, значение которой сохраняется после выхода из блока до повторного входа в него. При объявлении статической переменной используется ключевое слово **static**. Статической может быть объявлена как внешняя, так и внутренняя переменная.

Статические переменные во многом похожи на внешние. Они инициализируются нулем и существуют на протяжении жизни программы (в отличие от автоматических переменных, существующих лишь во время жизни блока).

Внешняя статическая переменная объявляется вне функции, но известна лишь в том файле, в котором она объявлена (даже при использовании ключевого слова **external**). Если внешняя переменная используется лишь в том файле, в котором она объявлена, ее следует сделать статической. Это повышает удобочитаемость и улучшает структуру программы. Кроме того, это позволяет избежать конфликтов с другими файлами, в которых могут содержаться те же имена внешних переменных, используемых в других целях.

Внутренняя статическая переменная объявляется внутри функции. В отличии от автоматической переменной, она существует на протяжении жизни всей программы.

Статические переменные инициализируются лишь однажды, перед выполнением программы, и сохраняют свои значения от одного вызова функции к другому.

Пример 1

```
static void errmsg (void);
static int count;                    /*внешняя статическая переменная*/
int adjust_total (int amount)
{
```

```
  static int total = 1024;          /*внутренняя статическая переменная*/
  int x = 0;


  total -= amount;
  ...
}
/* статическая функция является приватной в ее исходном файле */
static void errmsg (void)
{
...
}
```

Регистровый класс памяти

Регистровая переменная — автоматическая переменная, располагаемая, по возможности, на машинном регистре.

Для объявления регистровой переменной используется ключевое слово **register**. В этом случае переменная запоминается на машинном регистре, а не на стеке. Количество регистров, доступных для такого использования, является машинно-зависимым. Если потребность в регистровой памяти превышает число доступных аппаратных регистров, избыточные регистровые переменные трактуются как автоматические. В этом случае ключевое слово **register**попросту игнорируется.

Регистровые переменные рекомендуется использовать для повышения быстродействия программы. Лучшими кандидатами на регистровые переменные являются переменные, к которым осуществляется частый доступ, такие как счетчики циклов или индексов массивов.

Автоматические переменные и формальные параметры функции могут быть объявлены с классом памяти **register**. Внешние и статические переменныене могут быть регистровыми. Допустимые типы данных также зависят от системы, обычно это следующие типы данных: **char**, **int** и указатели. Массив не может запоминаться в регистре. Регистровые переменные имеют такие же область действия и время жизни, что и автоматические переменные. Операция **&**(операция получения адреса) не может быть применима к регистровой переменной.

Пример 1

```
f (register int count, register int num)
  {
   register int i;
   . . .
  }
```