Основы алгоритмизации и программирования

Лекция 12

Классы памяти и область действия объектов

Область действия

Все объекты программы на **Си** перед их использованием должны быть **декларированы**. Операционные объекты (в частности переменные) при этом, кроме атрибута «***тип***», имеют необязательный атрибут «***класс памяти***», существенно влияющий на область и время их действия.

Класс памяти программного объекта определяет время ее существования (время жизни) и область видимости (действия) и может принимать одно из значений: ***auto***, ***extern***, ***static*** и ***register***.

Область действия объекта – это часть кода программы, в которой его можно использовать для доступа к связанному с ним участку памяти. В зависимости от области действия переменная может быть локальной (внутренней) или глобальной (внешней).

Класс памяти и область действия объектов по умолчанию зависят от места их размещения в коде программы

Область действия

Имеется три основных участка программы, где можно декларировать переменные

- внутри функции (блока)
- в заголовке функции при определении параметров
- вне функции

Эти переменные соответственно называются **локальными (внутренними) переменными**, **формальными параметрами** и **глобальными (внешними) переменными**.

Область действия локальных данных – от точки декларации до конца функции (блока), в которой произведена их декларация, включая все вложенные блоки.

Областью действия глобальных данных считается файл, в котором они определены, от точки описания до его окончания.

Если класс памяти у переменной **не указан явным образом**, он определяется компилятором исходя из контекста ее декларации.

Время жизни может быть **постоянным** – в течение выполнения программы, и **временным** – в течение выполнения функции (блока) программы.

Автоматические переменные

Переменные, декларированные внутри функций, являются **внутренними** и называются **локальными переменными**. Никакая другая функция не имеет прямого доступа к ним. Такие объекты существуют временно на этапе активности функции.

Каждая локальная переменная существует только в блоке кода, в котором она объявлена, и уничтожается при выходе из него. Эти переменные называются автоматическими и располагаются в стековой области памяти.

При необходимости такая переменная инициализируется каждый раз при выполнении оператора, содержащего ее определение. Освобождение памяти происходит при выходе из функции (блока), в которой декларирована переменная, т.е. время ее жизни – с момента описания до конца блока.

По умолчанию локальные объекты, объявленные в коде функции, имеют атрибут класса памяти ***auto***.

```
void main(void)  {
        auto int max, lin;
            ...
}
```

так поступают, если хотят показать, что определение переменной не нужно искать вне функции.

Автоматические переменные

Регистровая память (атрибут ***register***) – объекты целого типа и символы рекомендуется размещать не в ОП, а в регистрах общего назначения (процессора), а при нехватке регистров – в стековой памяти (размер объекта не должен превышать разрядности регистра), для других типов компилятор может использовать другие способы размещения или просто проигнорировать данную рекомендацию.

Регистровая память позволяет увеличить быстродействие программы, но к размещаемым в ней объектам в языке Си (но не C++) не применима операция получения адреса «&».

Статические и внешние переменные

Объекты, размещаемые в статической памяти, декларируются с атрибутом ***static*** и могут иметь **любой атрибут области действия**. В зависимости от расположения оператора описания статические переменные могут быть **глобальными** и **локальными**. **Время жизни – постоянное**. Инициализируется один раз при первом выполнении оператора, содержащего определение переменной. Глобальные объекты всегда являются статическими. Атрибут ***static***, использованный при описании глобального объекта, предписывает ограничение его области применения только в пределах остатка текущего файла, а значения локальных статических объектов сохраняются до повторного вызова функции, т.е. в языке **Си** ключевое слово ***static*** имеет разный смысл для локальных и глобальных

Переменная, описанная вне функции, является внешней (глобальной) переменной.

Так как внешние переменные доступны всюду, их можно использовать вместо списка аргументов для передачи значений между функциями

Внешние переменные существуют постоянно. Они сохраняют свои значения и после того, как функции, присвоившие им эти значения, завершат свою работу.

Статические и внешние переменные

Внешняя переменная должна быть определена вне всех функций. При этом ей выделяется фактическое место в памяти. Такая переменная должна быть описана в той функции, которая собирается ее использовать. Это можно сделать либо явным описанием ***extern***, либо по контексту.

Описание ***extern*** может быть опущено, если определение внешней переменной находится в том же файле, но до ее использования в некоторой конкретной функции.

Ключевое слово ***extern*** позволяет функции использовать внешнюю переменную, даже в том случае, если она определяется позже в этом или

Важно различать описание внешней переменной и ее определение. Описание указывает свойство переменной, ее размер, тип и т. д.; определение же вызывает еще и отведение ей участка оперативной памяти. Например, если вне какой-либо функции появляются инструкции

```
int sp;
double val[20];
```

то они определяют внешние переменные *sp* и *val*, вызывают отведение памяти для них и служат в качестве описания для остальной части этого исходного файла. В то же время строчки:

```
extern int sp;
extern double val [ ];
```

описывают в остальной части этого исходного файла переменную *sp* как *int*, а *val* как массив типа *double*, но не создают переменных и не отводят им места в памяти.

Статические и внешние переменные

Любая инициализация внешней переменной проводится только в декларации. В декларации должны указываться размеры массивов, а в описании ***extern*** этого можно не делать.

Например, в основном файле проекта:

```
int sp = 50;
double val [20];
void main() {
        ...
```

а в дополнительном файле этого проекта:

```
extern int sp;
extern double val [ ];
        ...
```

Статические и внешние переменные

В Си есть возможность с помощью директивы компилятору ***#include*** использовать во всей программе только одну копию описаний ***extern*** и присоединять ее к каждому файлу во время его препроцессорной обработки.
Если переменная с таким же идентификатором, как внешняя, декларирована в функции без указания ***extern***, то тем самым она становится внутренней (локальной) для данной функции.

Не стоит злоупотреблять внешними переменными, так как такой стиль программирования приводит к программам, связи данных внутри которых не вполне очевидны. Переменные при этом могут изменяться неожиданным образом. Модификация таких программ вызывает затруднения.

Статические и внешние переменные

Пример, иллюстрирующий использование внешних данных:

Основной файл проекта	Дополнительный файл

…
int x, y;
char str[ ] = “Rezult = ”;
void fun1(void);
void fun2(void);
void fun3(void);
void main(void)
{
fun1();
fun2();
fun3();
}
void fun1(void) {
y = 15;
printf(“\n %s %d\n”, str, y);
}
	…
	extern int x, y;
	extern char str[ ];
	int r = 4;
	
	void fun2(void) {
	x = y / 5 + r;
	printf(“ %s %d\n”, str, x);
	}
	
	void fun3(void) {
	int z= x + y;
	printf(“ %s %d\n”, str, z);
	}

В результате выполнения этого проекта, состоящего из двух различных файлов, будет получено следующее:

Rezult = 15
Rezult = 7
Rezult = 22

Область действия переменных

В языке Си нет ключевого слова, указывающего область действия (видимости) объекта. Область действия определяется местоположением декларации объекта в файле исходного текста программы.

Общая структура исходного текста программы:

<директивы препроцессора>
<описание глобальных объектов>
    <заголовок функции>
    {
        <описание локальных объектов>
        <список инструкций>
    }

Файл исходного текста может включать любое количество определений функций и/или глобальных данных.

Область действия переменных

Общая структура исходного текста программы:

<директивы препроцессора>
<описание глобальных объектов>
<заголовок функции>
{
<описание локальных объектов>
<список инструкций>
}

Файл исходного текста может включать любое количество определений функций и/или глобальных данных.
Параметры функции являются локальными объектами и должны отличаться по идентификаторам от используемых в коде функции глобальных объектов.

Область действия переменных

Существуют следующие области действия: блок, файл, функция, прототип функции, область структурированных типов данных.

Блок. Идентификаторы, описанные внутри блока, являются локальными. Область действия идентификатора начинается в точке определения и заканчивается в конце блока, видимость – в пределах блока и внутренних блоков, время жизни – до выхода из блока. После выхода из блока память освобождается.

Файл. Идентификаторы, описанные вне любого блока, функции, класса или пространства имен, имеют глобальную видимость и постоянное время жизни и могут использоваться с момента их определения.

Функция. Единственными идентификаторами, имеющими такую область действия, являются метки операторов. В одной функции все метки должны различаться, но могут совпадать с метками других функций.

Прототип функции. Идентификаторы, указанные в списке параметров прототипа (декларации) функции, имеют областью действия только прототип функции.

Структурированный тип данных. Элементы структур и объединений являются видимыми лишь в их пределах. Они образуются при создании переменной указанного типа и разрушаются при ее уничтожении.