Storage classes and Functions

Storage Classes

Storage class specifiers tell compiler the duration and visibility of the variables or objects declared, as well as, where the variables or objects should be stored.

In C++ program we have multiple files. In these files we may have normal variables, array, functions, structures, unions, classes etc. So, variables and objects declared must have the visibility, the lifetime and the storage, when values assigned.

Storage Classes

In C / C++ there are 4 different storage classes available:

`automatic`, `external`, `static` and `register`.

It is similar in C

Storage class	Keyword
Automatic	`auto`
External	`extern`
Static	`static`
Register	`register`

Automatic Variable - auto

Local variables are variables declared within a function or blocks (after the opening brace, { of the block). Local variables are automatic by default. This means that they come to **existence** when the function in which it is declared is invoked and **disappears** when the function ends.

Automatic variables are declared by using the keyword auto. But since the variables declared in functions are automatic by default, this keyword may be dropped in the declaration as you found in many source codes.

Example : auto int x, y, z = 30; auto char firstname;

- Same as:

int x, y, z = 30; char firstname;

External Variable - extern

External variables are variables that are recognized **globally**, rather than locally. In other words, once declared, the variable can be used in any line of codes throughout the rest of the program.

A variable defined outside a function is external. An external variable can also be declared within the function that uses it by using the keyword extern hence it can be accessed by other code in other files

Ex: extern int value1;

extern char name;

extern double value2;

Static Variable - static

In a single file program, static variables are defined within individual functions that they are local to the function in which they are defined. Static variables are local variables that **retain their values** throughout the lifetime of the program. In other words, their same (or the latest) values are still available when the function is re-invoked later.

- Their values can be utilized within the function in the same manner as other variables, but they cannot be accessed from outside of their defined function.

Void show(){ static int s=0;s++;cout<<s;}

Register Variable - register

The above three classes of variables are normally stored in computer memory. Register variables however are stored in the **processor registers**, where they can be accessed and manipulated faster. Register variables, like automatic variables, are local to the function in which they are declared.

Usually, only register variables are assigned the register storage class. If all things equal, a program that makes use of register variables is likely to run faster than an identical program that uses just automatic variables.

Ex: register int x;

Functions

Functions allow you to group commonly used code into a compact unit that can be used repeatedly. You have already encountered one function, main()

It is a special function called at the beginning of the program. All other functions are directly or indirectly called from main().

Suppose you want to write a program to compute the area of three triangles. You could write out the formula three times, or you could create a function to do the work and then use that function three times

Sections of a Function

Name

- Name of the function

Description

- Description of what the function does

Parameters

- Description of each parameter to the function

Returns

- Description of the return value of the function

Classification of function

Library functions

Or

Standard in-build function

Or

Compiler function (like sqrt(), pow(), tan() etc)

User defined functions

Or

Self-contained program

Or

Call by value

Call by reference function (like show(), sum(), compute() etc)

Parts of a function in a Program:

Function declaration (giving info to compiler abt function)

- int show(int);

Function call (calling a function)

- i=show(35);

Function definition (giving body to a function)

- int show(int x)

```
{
cout<<“value of x is “<<x;
x++;
return(x);

}
```

Categories of a function

Function with no argument no return value

Function with argument and no return value

Function with no argument and return value

Function with argument and return value

Actual and Formal Arguments

The arguments declared in the function header/ function declaration is called as **formal arguments.**

And

The arguments passed to the functions while the function is called is known as the **actual arguments,**

Example

Ex. Suppose sum() is a function.

```
 int sum(int x, int y); /*Here x and y are called formal arguments*/
{...}
 void main()
{
int ans;
ans = sum(3,5); /*Here the arguments 3 and 5 are called actual arguments*/
getch();
}
```

Default Argument:

Default Argument instructs the compiler what value to pass for an argument if the programmer deliberately misses the argument when calling a function.

A default argument is a part of function declaration (not definition)

Example

```
#include <iostream>

int showVolume(int length, int width = 1,
int height = 1);

int main()

{

int vola,volb,volc;

vola=showVolume(4, 6, 2);

volb=showVolume(4, 6);

volc=showVolume(4);

return 0;

}
```

```
 int showVolume(int length, int width, int height)

{

int volume;

volume=length*width*height;

cout<<"vol is   "<<volume<<endl;

return volume;

}
```

Recursion:

When a function calls itself, it is called recursion.

It is also called self-calling program.

WAP to find factorial of a number using recursion???

Thanks