Unit #: 2

Unit Name: Counting, Sorting and Searching

Topic: Storage classes

Course objectives:

The objective(s) of this course is to make students

CObj1: Acquire knowledge on how to solve relevant and logical problems using computing machine

CObj2: Map algorithmic solutions to relevant features of C programming language constructs

CObj3: Gain knowledge about C constructs and it's associated eco-system

CObj4: Appreciate and gain knowledge about the issues with C Standards and it's respective behaviors

CObj5: Get insights about testing and debugging C Programs

Course outcomes:

At the end of the course, the student will be able to

CO1: Understand and apply algorithmic solutions to counting problems using appropriate C Constructs

CO2: Understand, analyse and apply text processing and string manipulation methods using C Arrays, Pointers and functions

CO3: Understand prioritized scheduling and implement the same using C structures

CO4: Understand and apply sorting techniques using advanced C constructs

CO5: Understand and evaluate portable programming techniques using preprocessor directives and conditional compilation of C Programs

Team – PSWC,

Jan - May, 2022

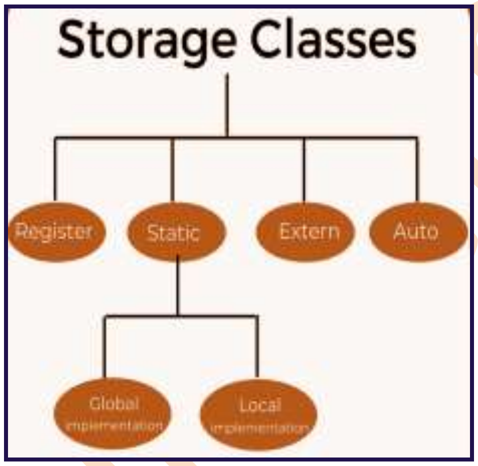
Dept. of CSE,

PES University

Storage classes

Introduction

Storage Classes are used to describe the features of a variable/function. These features basically include the **scope(visibility) and life-time which help us to trace the existence of a particular variable during the runtime of a program**. The following storage classes are most often used in C programming. Storage classes in C are used to determine the **lifetime, visibility, memory location, and initial value of a variable.**



Automatic Variables

A variable declared inside a function without any storage class specification is by default an automatic variable. They are created when a function is called and are destroyed automatically when the function execution is completed. Automatic variables can also be called local variables because they **are local to a function.** By default, they are assigned to undefined values**.**

Coding Example_1:

```
#include<stdio.h>
 int main()
{       int i=90;       // by default auto because defined inside a function
        auto float j=67.5;
        printf("%d %f\n",i,j);
}
int f1()
{
            int a;  // by default auto because declared inside a function f1()
                    // Not accessible outside this function.
}
```

External variables

The extern keyword is used before a variable **to inform the compiler that the variable is declared somewhere else**.The extern declaration does not allocate storage for variables. All functions are of type extern. **The default initial value of external integral type is 0 otherwise null. We can only initialize the extern variable globally, i.e., we cannot initialize the external variable within any block or method.**

Coding Example_2:

```
int main()
{
          extern int i;     // Information saying declaration exist somewhere else and make it available during linking
                // if u comment this line, it throws an error: i undeclared
          printf("%d\n",i);
}
int i=23;
```

Note:

An external variable can be declared many times but can be initialized at only once.

The extern modifier is most commonly used when there are two or more files sharing the same global variables or functions

Coding Example_3: Two files share the same global variable

Sample.c

```
#include <stdio.h>
 int count ;

extern void write_extern();
int main() {
        count = 5;
        write_extern();
        return 0;
}
```

Sample1.c

```
#include <stdio.h>
 extern int count;
 void write_extern(void)
{
   printf("count is %d\n", count);
}
```

Here, extern is being used to declare count in the second file, where as it has its definition in the first file, main.c.

```
// please check the execution steps below
gcc Sample.c Sample1.c –c
gcc Sample.o Sample1.o
```

Output - count is 5

Static variables

A static variable tells the compiler to persist the variable until the end of program. Instead of creating and destroying a variable every time when it comes into and goes out of scope, static is initialized only once and remains into existence till the end of program. A static variable can either be local or global depending upon the place of declaration.

Scope of local static variable remains inside the function in which it is defined but the life time of local static variable is throughout that program file. Global static variables remain restricted to scope of file in each they are declared and life time is also restricted to that file only. **All static variables are assigned 0 (zero) as default value.**

Coding Example_4: Demo for Life time of Static variable

```
 #include <stdio.h>
 void display();
 int main()
 {
        display();
        display();
 }
 void display()
 {
        static int a;
        a+=10;
        printf("The value of a is %d\n",a);
}
```

The ouput of the above program is:

```
The value of a is 10
The value of a is 20
```

Coding Example_5: Understand the differene between local and global static variable.

```
#include<stdio.h>
void f1();
static int j=20;            // global static variable: cannot be used outside this program file
int k=23;                   //global variable: can be used anywhere by linking with this file.
                            //By default   has extern with it.
int main( )
{
        f1();           // 0  20
        f1();           // 0  21
        f1();           // 0  22
        return 0;
}
void f1()
{
        int i=0;
        printf("%d %d\n",i,j);
        i++;
        j++;
}
```

Register variables

Registers are faster than memory to access, so the variables which are most frequently used in a C program can be put in registers using register keyword. The keyword register hints to compiler that a given variable can be put in a register. It's compiler's choice to put it in a register or not.

Register is an integral part of the processor. It is a very fast memory of computer mainly used to execute the programs and other main operation quite efficiently. They are used to quickly store, accept, transfer, and operate on data based on the instructions that will be immediately used by the CPU. Main operations are fetch, decode and execute. Numerous fast multiple ported memory cells are the atomic part of any register. Generally, compilers themselves do optimizations and put the variables in register. If a free register is not available, these are then stored in the memory only. If & operator is used with a register variable then compiler may give an error or warning (depending upon the compiler used), because when a variable is a register, it may be stored in a register instead of memory and accessing address of a register is invalid. **The access time of the register variables is faster than the automatic variables.**

Mostly used cpu registers are Accumulator, Flag Register, Address Register (AR), Data Register (DR), Program Counter (PC), Instruction Register (IR), Stack Control Register (SCR), Memory Buffer Register (MBR) and Index register (IR)

Coding Example_6:

```
#include<stdio.h>
int main()
{
        register int i = 10;
```

```
	int* a = &i;  // error
	printf("%d", *a);
	getchar();
	return 0;
}
```

Coding Example_7: We can store pointers into the register, i.e., a register can store the address of a variable.

```
#include<stdio.h>
int main()
{
	int i = 10;
	register int* a = &i;    // fine
	printf("%d", *a);
	getchar();
	return 0;
}
```

Note: Static variables cannot be stored into the register since we cannot use more than one storage specifier for the same variable

Global variables

The variables declared outside any function are called global variables. They are not limited to any function. **Any function can access and modify global variables**. Global variables are **automatically initialized to 0 at the time of declaration.**

Coding Example_8:

```
#include<stdio.h>
int i = 100;
int j;
int main()
{
        printf("%d\n",i);                       // 100
        i=90;
        printf("%d\n",i);                      // 90
        f1();
        printf("%d\n",i);                      // 40
        printf("%d\n",j);                      // 0 by default, global variable is 0 if just declared
        return 0;
}
int f1()
{
                i = 40;       // Any function can modify the global variable.
}
```

Think about it!

What if we have int i = 200; in a function f1() ?

Happy Thinking and Coding!!