Storage Class in C

Variable in C Programming

- **Variables** are the names you give to computer memory locations which are used to store values in a computer program.
- Every variable in C programming has two properties: **type** and **storage** class.
- **Type** refers to the data type(int, char, float, double, etc.)of a variable. And, **Storage class** determines the scope, visibility and lifetime of a variable.

If you are defining a variable :

Storage class _ data type _ variable name

auto int a;

Storage Class

- The **storage class** determines the part of the memory where the variable would be stored. Here we will discuss two storage location in computer : CPU Registers and Memory
- The storage class also determines the **initial value** of the variable.
- It used to define the **scope** and **lifetime** of variable.

CPU Register And Memory

A value stored in a CPU register can always be accessed faster than the one that is stored in memory.

Types of Storage Classes

There are **four types** of storage classes in C:

i. **Automatic** storage class

ii. **Register** storage class

iii. **Static** storage class

iv. **External** storage class

Automatic Storage Class

- Automatic variables are allocated memory automatically at runtime.
- The visibility of the automatic variables is limited to the block in which they are defined.
- The scope of the automatic variables is limited to the block in which they are defined.
- The automatic variables are initialized to garbage by default.
- The memory assigned to automatic variables gets freed upon exiting from the block.
- The keyword used for defining automatic variables is **auto**.
- Every local variable is automatic in C by default.

Automatic Storage Class

Keywords : auto

Storage : memory

Default initial value : garbage value*

Scope : local to the block in which the variable is defined

Life : till the control remains within the block in which the variable is defined.

auto int a=10;

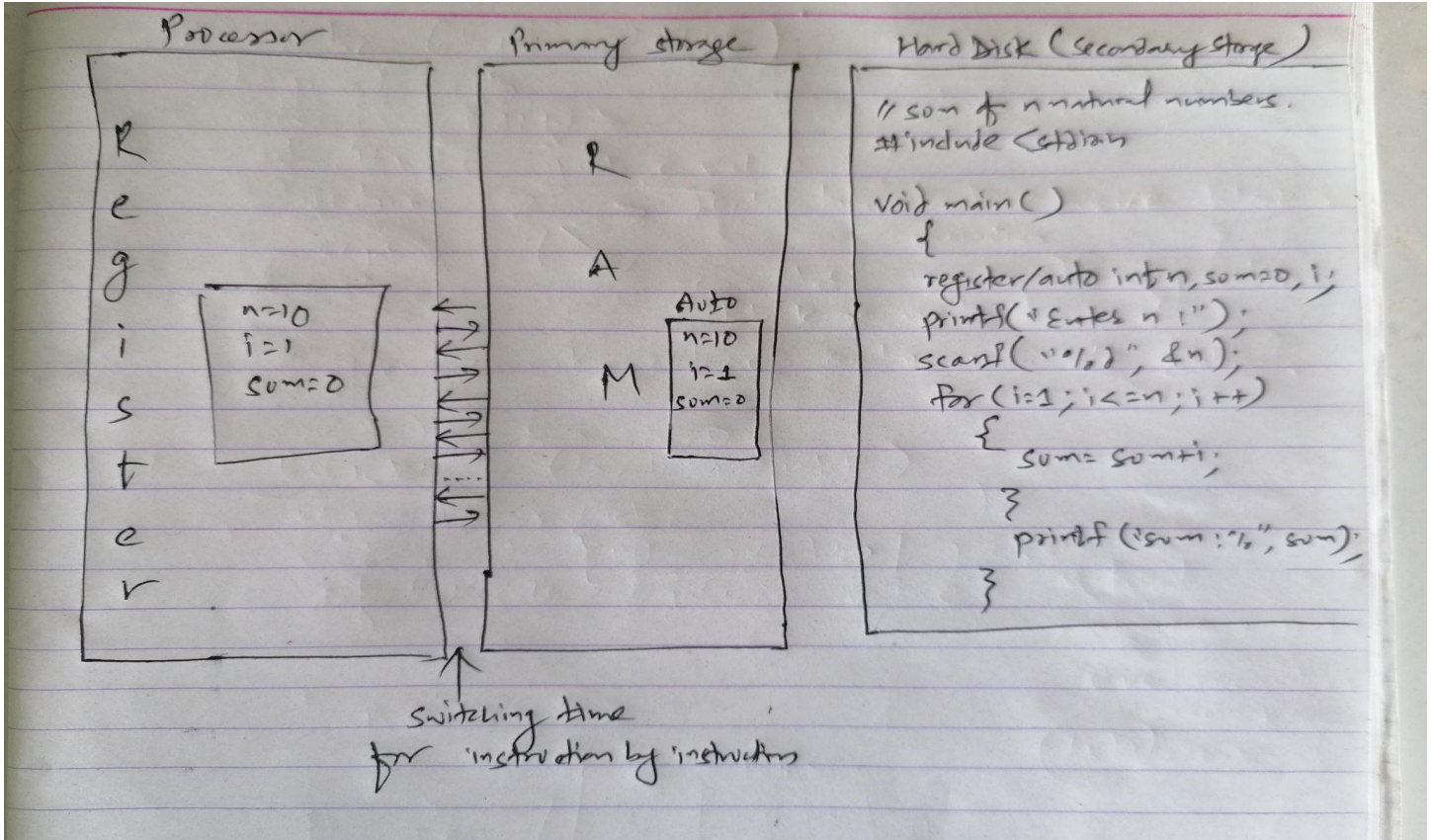
*garbage value: *C does not initialize most variables to a given value (such as zero) automatically. Thus when a variable is assigned a memory location by the compiler, the default value of that variable is whatever (garbage) value happens to already be in that memory location.*

Example: Automatic storage class

```
1.  // Example 1
2.  #include <stdio.h>
3.  int main()
4.  {
5.  int a; //auto
6.  char b;
7.  float c;
8.  printf("%d %c %f",a,b,c); // printing initial
    default value of automatic variables a, b,
    and c.
9.  return 0;
10. }
```

```
1.  // Example 2
2.  #include <stdio.h>
3.  int main()
4.  {
5.  int a = 10,i;
6.  printf("%d ",++a);
7.  {
8.  int a = 20;
9.  for (i=0;i<3;i++)
10. {
11. printf("%d ",a); // 20 will be printed 3 times since it
    is the local value of a
12. }
13. }
14. printf("%d ",a); // 11 will be printed since the scope
    of a = 20 is ended.
15. }
```

Automatic Vs Register Storage Class



Register Storage Class

- The variables defined as the register is allocated the memory into the CPU registers depending upon the size of the memory remaining in the CPU.
- We can not dereference the register variables, i.e., we can not use **& operator** for the register variable.
- The access time of the register variables is faster than the automatic variables.
- The initial default value of the register local variables is 0.
- The register keyword is used for the variable which should be stored in the CPU register. However, it is compiler's choice whether or not; the variables can be stored in the register.
- We can store pointers into the register, i.e., a register can store the address of a variable.
- Static variables can not be stored into the register since we can not use more than one storage specifier for the same variable.

Register Storage Class

Keywords : register

Storage : CPU Register

Default initial value : garbage value

Scope : local to the block in which the variable is defined

Life : till the control remains within the block in which the variable is defined.

Register Storage Class

- If the microprocessor has 16-bit registers then they cannot hold a float value or a double value which requires 4 bytes(32-bit) and 8 bytes(64-bit)
- If you want to use the register storage class(16-bit microprocessor) with float and double variable then you won’t get any error messages. Your compiler would treat the variables as auto storage class.

Register Storage Class Examples

```
1.  #include <stdio.h>
2.  int main()
3.  {
4.  register int a; // variable a is allocated memory in the
    CPU register. The initial default value of a is 0.
5.  printf("%d",a);
6.  }
```

```
1.  #include <stdio.h>
2.  int main()
3.  {
4.  register int a = 0;
5.  printf("%u", &a); // This will give a
    compile time error since we can
    not access the address of a
    register variable.
6.  }
```

Static Storage Class

- The variables defined as static specifier can hold their value between the multiple function calls.
- Static local variables are visible only to the function or the block in which they are defined.
- A same static variable can be declared many times but can be assigned at only one time.
- Default initial value of the static integral variable is 0 otherwise null.
- The visibility of the static global variable is limited to the file in which it has declared.
- The keyword used to define static variable is static. E.g. static int a;

Static Storage Class

Keyword : static

Storage : memory

Default initial value : zero

Scope : local to the block in which the variable is defined

Life : value of the variable persists between different function calls.

Example: Static Storage Class

```
1.  #include<stdio.h>
2.  static char c;
3.  static int i;
4.  static float f;
5.  static char s[100];
6.  void main ()
7.  {
8.  printf("%d %d %f %s",c,i,f); // the initial default value
    of c, i, and f will be printed.
9.  }
```

```
1.  #include<stdio.h>
2.  void sum()
3.  {
4.  static int a = 10;
5.  static int b = 24;
6.  printf("%d %d \n",a,b);
7.  a++;
8.  b++;
9.  }
10. void main()
11. {
12. int i;
13. for(i = 0; i< 3; i++)
14. {
15. sum(); // The static variables holds
    their value between multiple function
    calls.
16. }
17. }
```

Extern Storage Class

- The external storage class (or global variable)is used to tell the compiler that the variable defined as extern is declared with an external linkage elsewhere in the program.
- The variables declared as extern are not allocated any memory. It is only declaration and intended to specify that the variable is declared elsewhere in the program.
- The default initial value of external integral type is 0 otherwise null.
- We can only initialize the extern variable globally, i.e., we can not initialize the external variable within any block or method.
- An external variable can be declared many times but can be initialized at only once.
- If a variable is declared as external then the compiler searches for that variable to be initialized somewhere in the program which may be extern or static. If it is not, then the compiler will show an error.

Extern Storage Class

Keywords : extern

Storage : memory

Default initial value : zero

Scope : global

Life : as long as the program’s execution does not come to an end.

External Storage Class

```
1.  #include <stdio.h>
2.  int main()
3.  {
4.  extern int a;
5.  printf("%d",a);
6.  }
```

```
1.  #include <stdio.h>
2.  int a;
3.  int main()
4.  {
5.  extern int a; // variable a is defined globally,
    the memory will not be allocated to a
6.  printf("%d",a);
7.  }
```

Extern Storage Class: Examples

```
1. #include <stdio.h>
2. int a;
3. int main()
4. {
5. extern int a = 0; // this will show a compiler error
   since we can not use extern and initializer at same
   time
6. printf("%d",a);
7. }
```

```
1. include <stdio.h>
2. int main()
3. {
4. extern int a; // Compiler will search
   here for a variable a defined and
   initialized somewhere in the
   pogram or not.
5. printf("%d",a);
6. }
7. int a = 20;
```

Extern Storage Class Example

```
1.  extern int a;
2.  int a = 10;
3.  #include <stdio.h>
4.  int main()
5.  {
6.  printf("%d",a);
7.  }
8.  int a = 20; // compiler will show an error at this line
```

Summary: Storage Class

Storage Classes	Storage Place	Default Value	Scope	Lifetime
auto	RAM	Garbage Value	Local	Within function
extern	RAM	Zero	Global	Till the end of the main program Maybe declared anywhere in the program
static	RAM	Zero	Local	Till the end of the main program, Retains value between multiple functions call
register	Register	Garbage Value	Local	Within the function