Arrays

Module II

Array: One dimensional array –Two dimensional array – Strings and its operations. User defined Functions – Declarations –Definition- Call by value and call by reference- Types of functions- Recursive functions – Storage Classes-Scope , Visibility and Life time of variables.

Introduction

- Arrays
  - Structures of related data items
  - Static entity - same size throughout program
- A few types
  - C-like, pointer-based arrays

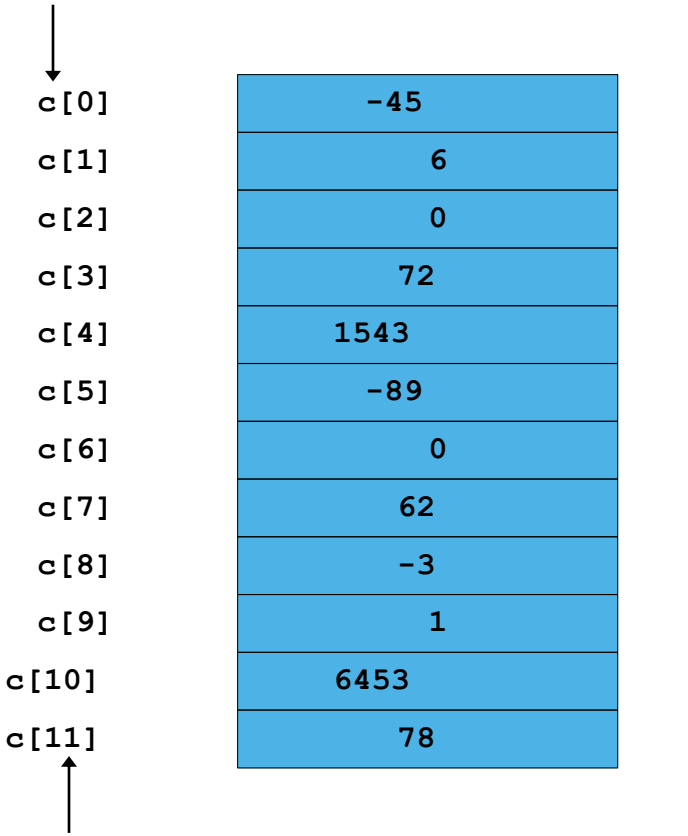
Arrays

- Array
  - Consecutive group of memory locations
  - Same name and type
- To refer to an element, specify
  - Array name and position number
- Format: *arrayname[position number]*
  - First element at position 0
  - **`n`** element array **`c`**:

    `c[ 0 ]`, **`c[ 1 ]`**…**`c[ n - 1 ]`**

Arrays

Name of array (Note that all elements of this array have the same name, **c**)



Position number of the element within array **c**

Declaring Arrays

- Declaring arrays - specify:
  - Name
  - Type of array
  - Number of elements
  - Examples

```
int c[ 10 ];
float hi[ 3284 ];
```

- Declaring multiple arrays of same type
  - Similar format as other variables
  - Example

```
int b[ 100 ], x[ 27 ];
```

Examples Using Arrays

- Initializers

```
int n[ 5 ] = { 1, 2, 3, 4, 5 };
```

  - If not enough initializers, rightmost elements become 0
  - If too many initializers, a syntax error is generated

```
int n[ 5 ] = { 0 }
```

  - Sets all the elements to **0**

- If size omitted, the initializers determine it

```
int n[] = { 1, 2, 3, 4, 5 };
```

  - 5 initializers, therefore **n** is a 5 element array

Examples Using Arrays

- Strings
  - Arrays of characters
  - All strings end with **`null`** (**`'\0'`**)
  - Examples:

```
char string1[] = "hello";
char string1[] = { 'h', 'e', 'l', 'l', 'o',
                   '\0' };
```

  - Subscripting is the same as for a normal array

`String1[ 0 ]` is **`'h'`**

`string1[ 2 ]` is **`'l'`**

Examples Using Arrays

- **https://www.tutorialspoint.com/cprogramming/c_arrays.htm**

- https://www.tutorialspoint.com/cprogramming/c_multi_dimensional_arrays.htm

Passing Arrays to Functions

- Specify the name without any brackets
  - To pass array **`myArray`** declared as

    ```
    int myArray[ 24 ];
    ```

    to function **`myFunction`**, a function call would resemble

    ```
    myFunction( myArray, 24 );
    ```
  - Array size is usually passed to function
- Arrays passed call-by-reference
  - Value of name of array is address of the first element
  - Function knows where the array is stored
    - Modifies original memory locations
- Individual array elements passed by call-by-value
  - pass subscripted name (i.e., **`myArray[ 3 ]`**) to function

Passing Arrays to Functions

- Function prototype:

```
void modifyArray( int b[], int arraySize );
```

  - Parameter names optional in prototype
    - **`int b[]`** could be simply **`int []`**
    - **`int arraysize`** could be simply **`int`**

    - Pass arrays to a function in C (programiz.com)

Sorting Arrays

- Sorting data
  - Important computing application
  - Virtually every organization must sort some data
    - Massive amounts must be sorted
- Bubble sort (sinking sort)
  - Several passes through the array
  - Successive pairs of elements are compared
    - If increasing order (or identical), no change
    - If decreasing order, elements exchanged
  - Repeat these steps for every element

Sorting Arrays

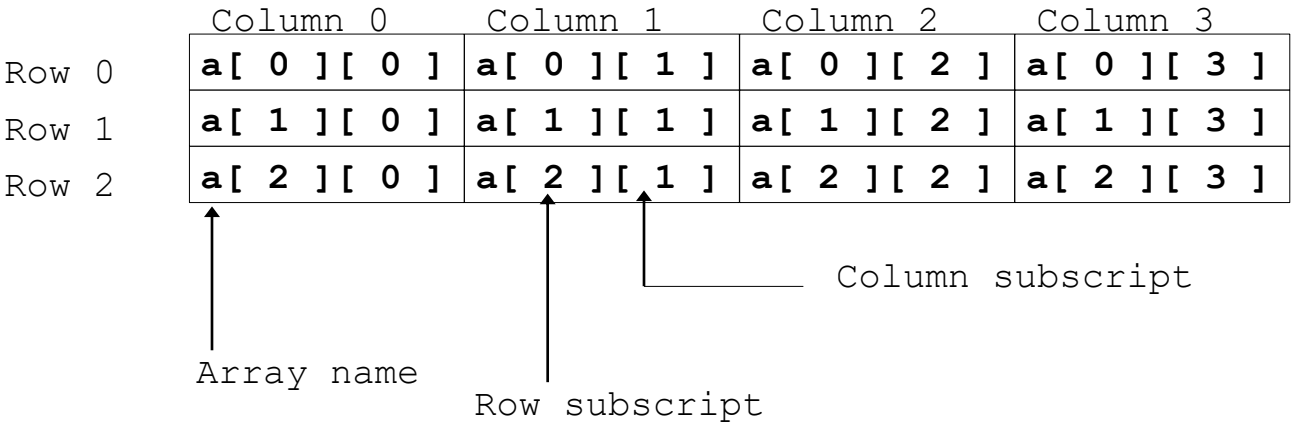
- Example:
  - Original: 3 4 2 6 7
  - Pass 1: 3 2 4 6 7
  - Pass 2: 2 3 4 6 7
  - Small elements "bubble" to the top

Computing Mean, Median and Mode Using Arrays

- Mean
  - Average
- Median
  - Number in middle of sorted list
  - 1, 2, 3, 4, 5 (3 is median)
- Mode
  - Number that occurs most often
  - 1, 1, 1, 2, 3, 3, 4, 5 (1 is mode)

Multiple-Subscripted Arrays

- Multiple subscripts - tables with rows, columns
  - Like matrices: specify row, then column.



- Initialize

```
int b[ 2 ][ 2 ] = { {1, 2}, {3, 4}};
```

1	2
3	4

  - Initializers grouped by row in braces

```
int b[ 2 ][ 2 ] = { { 1 }, { 3, 4 } };
```

→

1	0
3	4

{1,7, 6,9

2, 8,5,11

6, 5, 3,4}

a[0][3] = 9

a[1][2] = 5

a[2][3]= 4

Strings

- Strings A special kind of array is an array of characters ending in the null character called string arrays .
- A string is declared as an array of characters char s[1 0] char p[30]
- When declaring a string don't forget to leave a space for the null character which is also known as the string terminator character

Strings

- C offers four main operations on strings

  strcpy - copy one string into another

- strcat - append one string onto the right side of the other

- strcmp — compare alphabetic order of two strings

- strlen — return the length of a string

String Functions

- strcpy - strcpy(destinationstring, sourcestring)
- Copies sourcestring into destinationstring
- For example strcpy(str, "hello world"); assigns "hello world" to the string str

Example

- Example with strcpy
- #include main() char x[] = "Example with strcpy", char y[25];
- printf("The string in array x is %s “, strcpy(y,x)) ;
- printf("The string in array y is %s”, x);

StrCAT

- strcat strcat(destinationstring, sourcestring) appends sourcestring to right hand side of destinationstring
- For example if str had value "a big strcat(str, "hello world"); appends "hello world" to the string "a big " to get
- a big hello world

Example with strcat

```
#include <stdio.h>
#include <string.h>
    main()
        {
        char x[] = “Example with strcat”;
       char y[]= “which stands for string concatenation”;
           printf(“The string in array x is %s \n “, x);
           strcat(x,y);
            printf(“The string in array x is %s \n “, x);

           }
```

strcmp

- strcmp(stringa, stringb)

- Compares stringa and stringb alphabetically
- Returns a negative value if stringa precedes stringb alphabetically
- Returns a positive value if stringb precedes stringa alphabetically
- Returns 0 if they are equal
- Note lowercase characters are greater than Uppercase

Example with strcmp

```
#include <stdio.h>
#include <string.h>
   main()
      {
      char x[] = “cat”;
       char y[]= “cat”;
        char z[]= “dog”;
        if (strcmp(x,y) == 0)
          printf(“The string in array x  %s is equal to
   that in %s \n “, x,y);
```

continued

```
if (strcmp(x,z) != 0)
        {printf("The string in array x  %s is not equal to that in z %s \n ",
         x,z);
   if (strcmp(x,z) < 0)
      printf("The string in array x  %s precedes that in z %s \n ", x,z);
   else
      printf("The string in array z  %s precedes that in x %s \n ", z,x);
        }
else
       printf( "they are equal");
          }
```

strlen

- strlen(str) returns length of string excluding null character
- strlen(“tttt”) = 4 not 5 since \0 not counted

Example with strlen

```
#include <stdio.h>
#include <string.h>
    main()
        {
       int i, count;
       char x[] = “tommy tucket took a tiny ticket ”;
       count = 0;
       for (i = 0; i < strlen(x);i++)
          {
            if (x[i] == ‘t’) count++;
           }
      printf(“The number of t’s in   %s is %d \n “, x,count);

       }
```

Vowels Example with strlen

```
#include <stdio.h>
#include <string.h>
    main()
       {
       int i, count;
       char x[] = “tommy tucket took a tiny ticket ”;
       count = 0;
       for (i = 0; i < strlen(x);i++)
          {
            if ((x[i] == ‘a’)||(x[i]==‘e’)||(x[i]==‘I’)||(x[i]==‘o’)||(x[i]==‘u’))
    count++;
          }
      printf(“The number of vowels’s in   %s is %d \n “, x,count);

      }
```

No of Words Example with strlen

```
#include <stdio.h>
#include <string.h>
    main()
       {
       int i, count;
       char x[] = “tommy tucket took a tiny ticket ”;
       count = 0;
       for (i = 0; i < strlen(x);i++)
          {
            if ((x[i] == ‘ ‘) count++;
          }
      printf(“The number of words’s in   %s is %d \n “, x,count+1);

      }
```

No of Words Example with more than one space between words

```
#include <stdio.h>
#include <string.h>
      main()
         {
        int i,j, count;
         char x[] = “tommy tucket took a tiny ticket ”;
        count = 0;
        for (i = 0; i < strlen(x);i++)
           {
             if ((x[i] == ‘ ‘)
        { count++;
         for(j=i;x[j] != ‘ ‘;j++);
         i = j;
            }
           }
       printf(“The number of words’s in   %s is %d \n “, x,count+1);

        }
```

Input output functions of characters and strings

- getchar() reads a character from the screen in a non-interactive environment
- getche() like getchar() except interactive
- putchar(int ch) outputs a character to screen
- gets(str) gets a string from the keyboard
- puts(str) outputs string to screen

Exercise 1

- Characters are at the heart Of strings
- https://www.programmingsimplified.com/c/source-code/c-program-for-pattern-matching

Exercise 2

Output

```
*

* *

* * *

* * * *

. . . . . .
* * * * * * * * * *
```

Examples

- https://www.programiz.com/c-programming/c-strings#:~:text=In%20C%20programming%2C%20a%20string,at%20the%20end%20by%20default.

Module II - Functions Part II

Introduction-Functions-User defined Functions – Declarations –Definition- Call by value and call by reference- Types of functions- Recursive functions – Storage Classes-Scope , Visibility and Life time of variables.

Functions - Introduction

- **A relation is a <u>function</u> provided there is exactly one output for each input.**

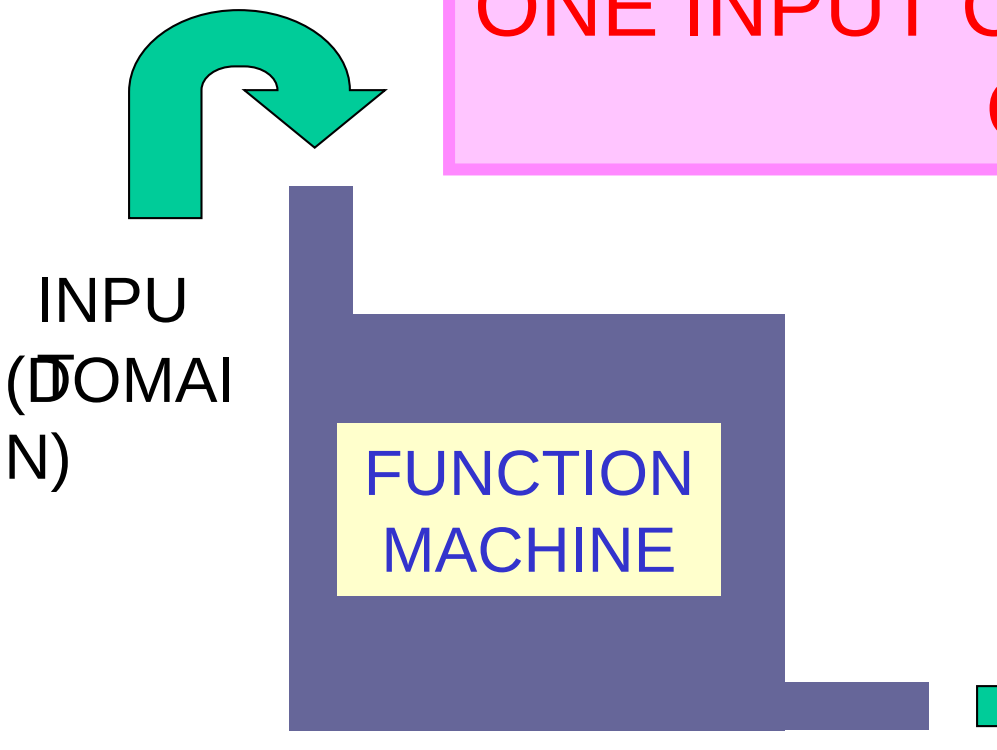
```
main( )
{
message( ) ;
message();
printf ( "\nCry, and you stop the
monotony!" ) ;
}
message( )
{
```

In order for a relationship to be a function…

EVERY INPUT MUST HAVE AN OUTPUT

TWO DIFFERENT INPUTS CAN HAVE THE SAME OUTPUT

ONE INPUT CAN HAVE ONLY ONE OUTPUT



Functions

Functions

A function is a self-contained block of statements that perform a coherent task of some kind. Every C program can be thought of as a collection of these functions.

Hiring a person for Job, sometimes its easy and sometimes its difficult.

Analogy- Motor bike repair

The monthly service of motor bike is done regularly , so it is done every month.

Let us now look at a simple C function that operates in much the same way as the mechanic. Actually, we will be looking at two things—a function that calls or

Example Function

```
main( )
{
printf ( "\nI am in main" ) ;
italy( ) ;
brazil( ) ;
argentina( ) ;
}
italy( )
{
printf ( "\nI am in italy" ) ;
}
brazil( )
{
printf ( "\nI am in brazil" ) ;
}
argentina( )
{
printf ( "\nI am in argentina" ) ;
}
```

Inferences from the Program

- Any C program contains at least one function.
- − If a program contains only one function, it must be **main( )**.
- − If a C program contains more than one function, then one (and only one) of these functions must be **main( )**, because program execution always begins with **main( )**.
- − There is no limit on the number of functions that might be present in a C program.
- − Each function in a program is called in the sequence specified by the function calls in **main( )**.

Sending and Receiving Values between Functions

```
calsum ( int, int, int ) ;
main( )
{
int a, b, c, sum=0 ;
printf ( "\nEnter any three numbers " ) ;
scanf ( "%d %d %d", &a, &b, &c ) ;
sum = calsum (a,b,c) ;
printf ( "\nSum = %d", sum ) ;
}
calsum ( int x, int y, int z )
{
int d ;
d = x + y + z ;
return ( d ) ;
}
```

User defined Functions

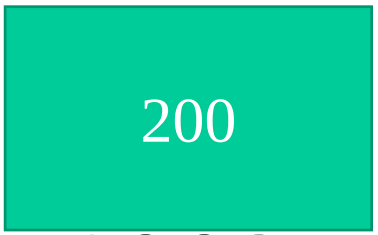
A function is a block of code that performs a specific task.

C allows you to define functions according to your need. These functions are known as user-defined functions. For example:

Suppose, you need to create a circle and color it depending upon the radius and color. You can create two

1002

1004



X Ad -1002

num Ad-1006

Change(&x)

* - value @ address

& - gives address

Function Prototype

A function prototype is simply the declaration of a function that specifies function's name, parameters and return type.

It doesn't contain function body.

A function prototype gives information to the compiler that the function may later be used in the program.

returnType functionName(type1 argument1, type2 argument2, ...);

Function Prototype

Example : Function 1 Demo - https://www.onlinegdb.com/

name of the function is addNumbers()

return type of the function is int

two arguments of type int are passed to the function

The function prototype is not needed if the user-defined function is defined before the main() function.

Pointers –Call by reference

- The **call by reference** method of passing arguments to a function copies the address of an argument into the formal parameter.
- Inside the function, the address is used to access the actual argument used in the call. It means the changes made to the parameter affect the passed argument.
- To pass a value by reference, argument pointers are passed to the functions just like any other value. swap(), exchanges the values of the two integer variables pointed to, by their arguments.

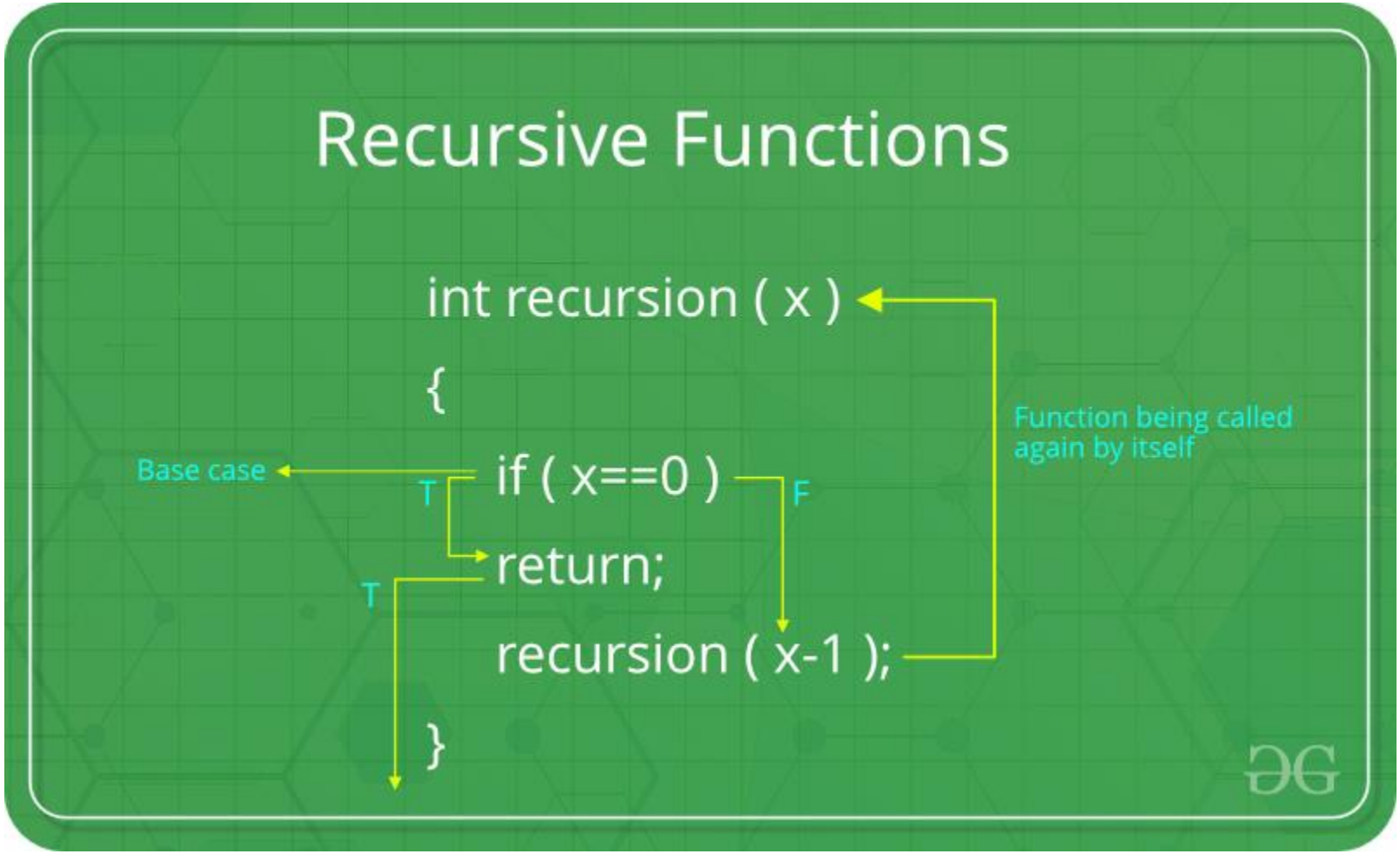
Function Declaration and Definition

- For example, if the my_function() function, discussed in the previous section, requires two integer parameters, the declaration could be expressed as follows: **return_type my_function**(int x, y); where int x, y indicates that the function requires two parameters, both of which are integers.

Call by reference - Example

```
#include <stdio.h>
 int main () {   /* local variable definition */
 int a = 100;   int b = 200;
printf("Before swap, value of a : %d\n", a );
printf("Before swap, value of b : %d\n", b );
  /* calling a function to swap the values */
 swap(&a, &b);
 printf("After swap, value of a : %d\n", a );
 printf("After swap, value of b : %d\n", b );
 return 0;}
void swap(int *x, int *y)
{
 int temp;
temp = *x;
*x = *y;
  /* put y into x */   *y = temp; /* put temp into y */
  return;
}
```

Recursive Function



Recursive Function - Examples

In programming terms, a recursive function can be defined as a routine that calls itself directly or indirectly.

STORAGE CLASS

Introduction

- Variables have data types and storage classes.
- Value in a computer can be stored in
  - Memory or
  - CPU registers
- A storage class determines where to store the value of a variable.

STORAGE CLASS

- The storage class determines the part of the memory where the variable would be stored.
- The storage class also determines the initial value of the variable.
- and it used to define the scope and lifetime of variable.
- There are two storage location in computer :

  CPU Registers and Memory

CPU REGISTER AND MEMORY

- A value stored in a CPU register can always be accessed faster then 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

- 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.

Example of Automatic Storage Class

```
#include<stdio.h>
#include<conio.h>
void main()
{
auto int i=1;
{
auto int i=2;
{
auto int i=3;
printf("\n%d",i);
}
printf("%d",i);
}
printf("%d",i);
getch();
}
```

Output:
3 2 1

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.

Example of Register Storage Class

```
#include<stdio.h>
#include<conio.h>
void main()
{
register int i;
for(i=1;i<=10;i++)
printf(“  %d",i);
getch();
}
```

Output:

1 2 3 4 5 6 7 8 9 10

External Storage Class

- Keywords : extern.
- Storage : memory.
- Default initial value : zero.
- Scope : global.
- Life : as long as the program's execution doesn't come to an end.

Example of External Storage Class

```
#include<stdio.h>
#include<conio.h>
int i =1;
increment();
void main()
{
printf("%d\t",i);
increment();
increment();
getch();
}
increment()
{
i++;
printf("%d\t",i);
}
Output:
1  2       3
```

Static Storage Class

- Keywords : 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.

Dif. b/w auto and static storage class

Automatic

```
#include<stdio.h>
#include<conio.h>
increment();
void main()
{
increment();
increment();
increment();
}
increment()
{
auto int i=1;
printf("%d\t",i);
i++;
getch();
}
```

Output:

1 1 1

Static

```
#include<stdio.h>
#include<conio.h>
increment();
void main()
{
increment();
increment();
increment();
}
increment()
{
static int i=1;
printf("%d\t",i);
i++;
getch();
}
```

Output:

1 2 3

Storage Class	Storage Area	Default initial value	Lifetime	Scope	Keyword
Automatic	Memory	Till control remains in block	Till control remains in block	Local	Auto
Register	CPU register	Garbage value	Till control remains in block	Local	Register
Static	Memory	Zero	Value in between function calls	Local	Static
External	Memory	Garbage value	Throughout program execution	Global	Extern

Scope – Visibility – Life time

Scope

The region of a program in which a variable is available for use.

Visibility

The program's ability to access a variable from the memory.

Lifetime

The lifetime of a variable is the duration of time in which a variable exists in the memory during execution.

Life time of the Variable

File Scope

Any variable declared with file scope can be accessed by any function defined after the declaration (in our example both f and main can access global_a).

If global was declared after the function f but before main it would only be accessible within main.

Block Scope

Block scope is defined by the pairing of the curly braces { and } . A variable declared within a block can only be accessed within that block.

Visibility of Variable

Storage Class	Storage Area	Default initial value	Lifetime	Scope	Keyword
Automatic	Memory	Till control remains in block	Till control remains in block	Local	Auto
Register	CPU register	Garbage value	Till control remains in block	Local	Register
Static	Memory	Zero	Value in between function calls	Local	Static
External	Memory	Garbage value	Throughout program execution	Global	Extern

Explaining the Storage Class

Scope of Variable

File Scope

Any variable declared with file scope can be accessed by any function defined after the declaration.

Block Scope

Block scope is defined by the pairing of the curly braces { and } .

A variable declared within a block can only be accessed within that block.

Global Variables

- Variables can be declared outside functions
- Accessible from everywhere
- Initialized before main() started
- Live until program terminates
- Function static variables can be thought of as global

Link Tree - link

https://linktr.ee/rspdarshinir