RAMAIAH
Institute of Technology

M. S. Ramaiah Institute of Technology

(Autonomous Institute, Affiliated to VTU)

Department of Computer Science and Engineering

Data Structures CS 32

Vandana S Sardar
Mamatha Jadhav V

Syllabus

Unit I

Basic Concepts: Pointers and Dynamic Memory Allocation, Algorithm Specification, Data Abstraction. Arrays and Structures: Arrays, Dynamically Allocated Arrays, Structures and Unions, Polynomials, Sparse Matrices, Representation of Multidimensional Arrays, Strings.

Unit II

Stacks and Queues: Stacks, Stacks Using Dynamic Arrays, Queues, Circular Queues Using Dynamic Arrays, Evaluation of Expressions, Multiple Stacks and Queues.

Syllabus

Unit III

Linked Lists: Singly Linked lists and Chains, Representing Chains in C, Linked Stacks and Queues, Polynomials, Additional List operations, Sparse Matrices, Doubly Linked Lists.

Unit IV

Trees: Introduction, Binary Trees, Binary Tree Traversals, Additional Binary Tree Operations, Threaded Binary Trees, Heaps, Binary Search Trees, Selection Trees, Forests, Representation of Disjoint Sets, Counting Binary Trees.

Syllabus

Unit V

Graphs: The Graph Abstract Data Type, Elementary Graph Operations. Priority Queues: Single- and Double-Ended Priority Queues, Leftist Trees. Efficient Binary Search Trees: AVL Trees.

Text Book:

1. Horowitz, Sahni, Anderson-Freed: Fundamentals of Data Structures in C, 2nd Edition, Universities Press, 2008.

Revision of C Concepts: E Balagurusamy

- Introduction to C
- Data Types
- C Constants
- Operators
- Control Structures
- Loops

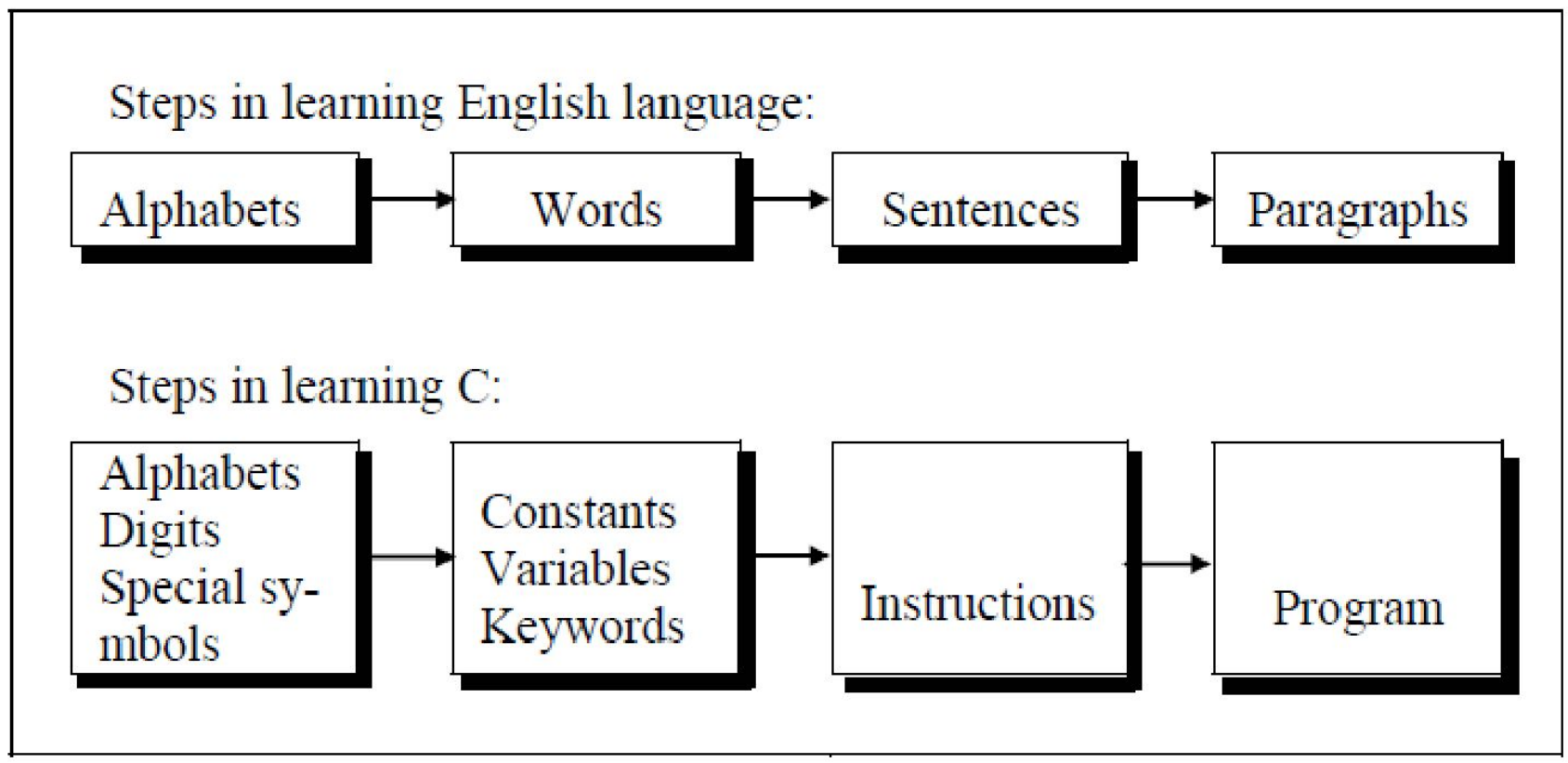
Revision of C Concepts: E Balagurusamy

- Arrays
- Strings
- Functions
- Structures
- Pointers

Introduction to C

- C is a programming language developed at AT & T's Bell Laboratories of USA in 1972.
- C compiler is written in C.
- Major parts of popular operating systems like Windows, UNIX, Linux is still written in C.

Introduction to C



Introduction to C

Character Set

Alphabets	A, B, ….., Y, Z a, b, ……, y, z
Digits	0, 1, 2, 3, 4, 5, 6, 7, 8, 9
Special symbols	~ ‘ ! @ # % ^ & * ( ) _ - + = | \ { } [ ] : ; " ' < > , . ? /

Introduction to C

- Identifiers: User defined word used to name of entities like variables, arrays, functions, structures.

- Keywords: Reserved words

  Examples: int, short, signed, unsigned, default, volatile, float, long, double, break, continue, typedef, static, do, for, union, return, while, do, extern, register, enum, case, goto, struct, char, auto, const etc.

Data Types

- Basic built-in data types: int, float, double, char
- Enumeration data type: enum
- Derived data type: pointer, array, structure, union
- Void data type: void

Data Types

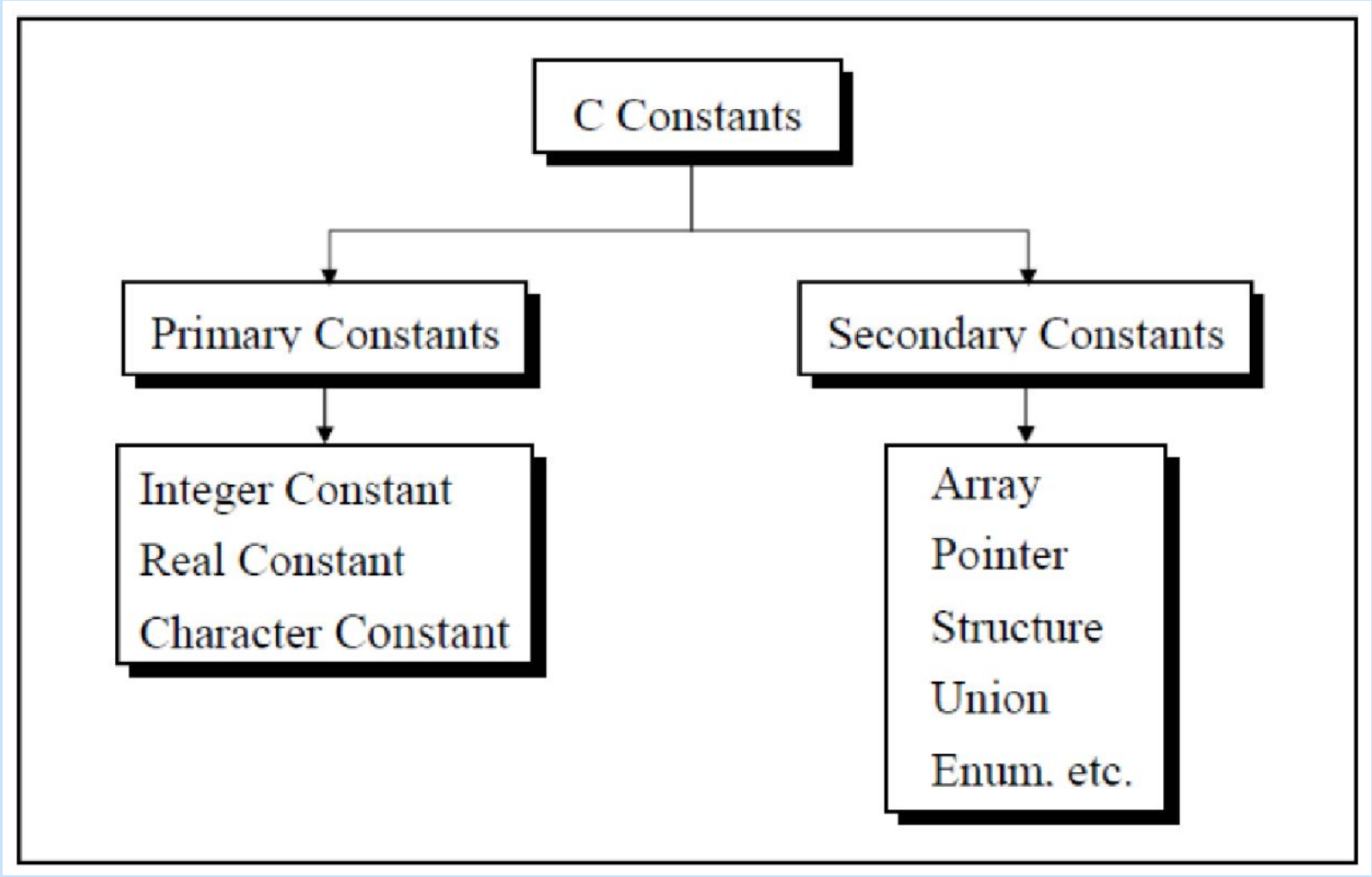
- There are two types of type qualifier in C:

- Size qualifier: short, long
- Sign qualifier: signed, unsigned

Data Types

Basic data type	Data type with type qualifier	Size (byte)	Range
char	char or signed char	1	-128 to 127
	Unsigned char	1	0 to 255
int	int or signed int	2	-32768 to 32767
	unsigned int	2	0 to 65535
	short int or signed short int	1	-128 to 127
	unsigned short int	1	0 to 255
	long int or signed long int	4	-2147483648 to 2147483647
	unsigned long int	4	0 to 4294967295
float	float	4	-3.4E-38 to 3.4E+38
double	double	8	1.7E-308 to 1.7E+308
	Long double	10	3.4E-4932 to 1.1E+4932

C Constants


C Constants
Primary Constants
Secondary Constants
Integer Constant
Real Constant
Character Constant
Array
Pointer
Structure
Union
Enum. etc.

Operators

Operators	Description	Precedence level	Associativity
()	function call	1	left to right
[]	array subscript		
→	arrow operator		
.	dot operator		
+	unary plus	2	right to left
-	unary minus		
++	increment		
- -	decrement		
!	logical not		
~	1's complement		
*	indirection		
&	address		
(data type)	type cast		
sizeof	size in byte		
*	multiplication	3	left to right
/	division		
%	modulus		

Operators

Operators	Description	Precedence level	Associativity
+	addition	4	left to right
-	subtraction		
<<	left shift	5	left to right
>>	right shift		
<=	less than equal to	6	left to right
>=	greater than equal to		
<	less than		
>	greater than		
==	equal to	7	left to right
!=	not equal to		

Operators

Operators	Description	Precedence level	Associativity
&	bitwise AND	8	left to right
^	bitwise XOR	9	left to right
|	bitwise OR	10	left to right
&&	logical AND	11	
||	logical OR	12	
?:	conditional operator	13	
=, *=, /=, %= &=, ^=, <<= >>=	assignment operator	14	right to left
,	comma operator	15	

Control Structures

- Simple if statement
- if..else statement
- Nested if else statement
- else if ladder
- switch statement

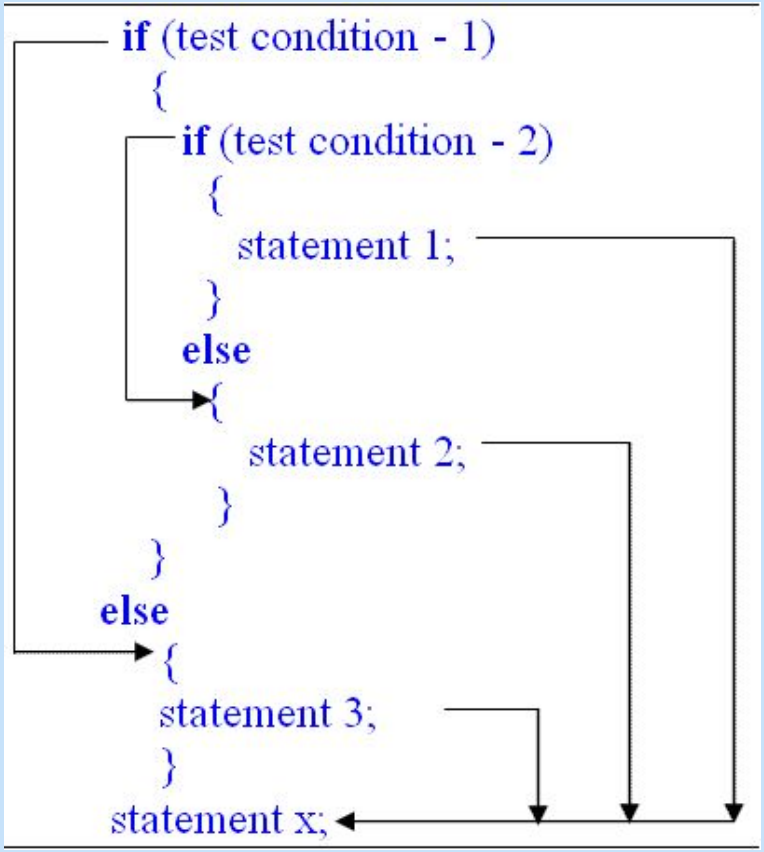
- Control Structures : Simple if and if else

```
if(condition)
    statement;
```

```
if(condition)
{
    statement 1;
    statement2;
    statement3;
}
```

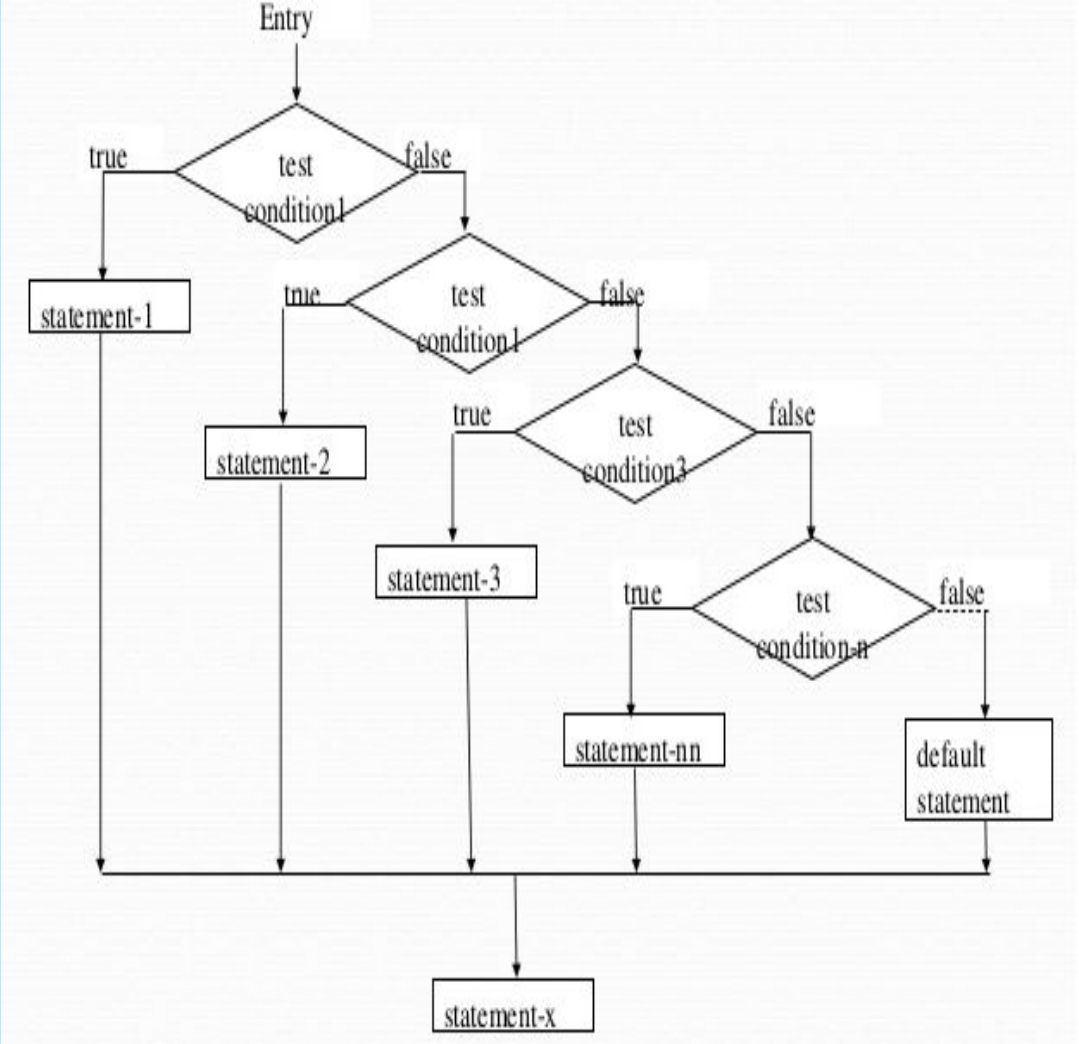
```
#include<stdio.h>
int main()
{
    int num=19;
    if(num<10)
    {
        printf("The value is less than 10");
    }
    else
    {
        printf("The value is greater than 10");
    }
    return 0;
}
```

Control Structures: Nested if else




```
if ( age < 18 )
{
 printf("You are Minor.\n");
 printf("Not Eligible to Work");
}
else
{

 if (age >= 18 && age <= 60 )
 {
  printf("You are Eligible to Work \n");
  printf("Please fill in your details and apply\n");
 }
 else
 {
  printf("You are too old to work as per the Government rules\n");
  printf("Please Collect your pension! \n");
 }
}
```

Control Structures: else if ladder




```c
#include<stdio.h>
int main()
{
    int marks=83;
    if(marks>75){
        printf("First class");
    }
    else if(marks>65){
        printf("Second class");
    }
    else if(marks>55){
        printf("Third class");
    }
    else{
        printf("Fourth class");
    }
    return 0;
}
```

Control Structures

- Switch statement

```
void main()
{
int roll = 3 ;
switch ( roll )
      {
      case 1 :
            printf("I am Pankaj");
            break;
      case 2 :
            printf("I am Nikhil");
            break;
      case 3 :
            printf("I am John");
            break;
      default :
            printf("No student found");
            break;
      }
}
```

Loops

- while loop
- do while loop
- for loop

Loops

- while loop

```
while(condition)
{
Statement 1;
Statement 2;
}
```

```
i=1;
while(i<=5)
{
printf("%d\t", i);
i++;
}
```

Loops

- do-while loop

```
do
{
statement;

}while(condition);
```

```
i=1;
do {
printf("%d\t", i);
i++;
} while(i<=5);
```

Loops

- for loop

```
for(exp1;exp2;exp3)
{
Statement;
}
```

```
for (i=1;i<=5;i++)
    printf("%d\t", i);
```

```
for (i=1;i<=5;i++)
    for(j=1;j<=5;j++)
    printf("\n%d\t%d", i,j);
```

Example: C Programs

- Write a C program to check whether number is even or odd.
- Write a C program to print the grade obtained by a student using else if ladder/switch case.
- Write a C program to find the factorial of a number.
- Write a C program to print the table of a number.

Arrays

- Array is the collection of similar data types or collection of similar entity stored in contiguous memory location.
- Array of characters is a string.
- Each data item of an array is called an element and located in separated memory location.
- Each of elements of an array share a variable but each element having different index no. known as subscript.

Arrays: Declaration and Initialization

DECLARATION OF AN ARRAY :

Its syntax is :

Data type array name [size];

int arr[100];

int mark[100];

int a[5]={10,20,30,100,5}

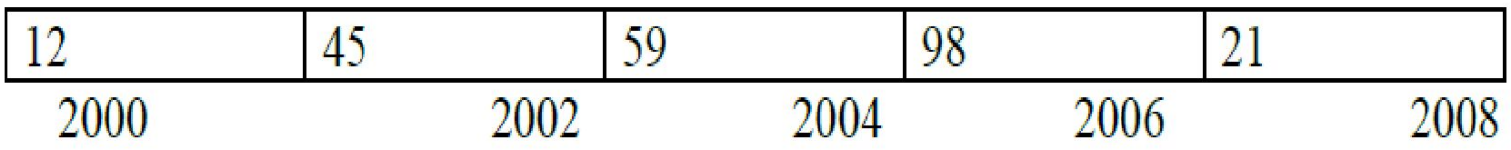
Data type array name [size] = {value1, value2, value3…}

Example:

in ar[5]={20,60,90, 100,120}

Arrays: Memory Layout and storing elements

a[0]=12, a[1]=45, a[2]=59, a[3]=98, a[4]=21




```
for (i=0; i<=9; i++)
{
printf ("enter the %d element \n", i+1);
scanf ("%d", &arr[i]);
}
```

Arrays: 2D, Matrix

- Two Dimensional arrays: Known as matrix.

Its syntax is

Data-type array name[row][column];

Example:-

int a[2][3];

Total no of elements=row*column is 2*3 =6

It means the matrix consist of 2 rows and 3 columns

For example:-

20	2	7
8	3	15

Arrays: 2D Example: Display a matrix

For displaying value:-

```
for(i=0;i<4;i++)
{
for(j=0;j<5;j++)
        {
                printf("%d",a[i][j]);
        }
}
```

Strings

- A string is a simple array with char as a data type.

- String is terminated with a special character '\0'.

The general syntax for declaring a variable as a string is as follows,

char string_variable_name [array_size];

```
char first_name[15]; //declaration of a string variable
char last_name[15];
```

The C compiler automatically adds a NULL character '\0' to the character array created.

Strings: Initialization

```
char first_name[15] = “RAMESH";

char first_name[15] = {‘R',‘A',‘M',‘E',‘S',‘H','\0'};

// NULL character '\0' is required at end in this declaration
```

Strings: Read a string

```
char name[10];

printf("Enter your first name : \n");
scanf("%s", name);
```

```
char name[10];

printf("Enter your name: \n");
 gets(name);
```

Strings: Write a string

```
printf("%s", name);
```

```
char name[15];

gets(name); //reads a string

 puts(name); //displays a string
```

Strings: String Handling Functions

Sr.No.	Function & Purpose
1	**strcpy(s1, s2);** Copies string s2 into string s1.
2	**strcat(s1, s2);** Concatenates string s2 onto the end of string s1.
3	**strlen(s1);** Returns the length of string s1.
4	**strcmp(s1, s2);** Returns 0 if s1 and s2 are the same; less than 0 if s1<s2; greater than 0 if s1>s2.
5	**strchr(s1, ch);** Returns a pointer to the first occurrence of character ch in string s1.
6	**strstr(s1, s2);** Returns a pointer to the first occurrence of string s2 in string s1.

Strings: strlen() and sizeof()

```
char str[] = "MSRIT";

printf("Length of String is %d\n", strlen(str));

 // Length of String is 5


printf("Size of String is %d\n", sizeof(str));

// Length of String is 6
```

Strings: String handling functions: A Program

```
#include <stdio.h>
#include <string.h>
int main ()
{
 char str1[12] = "Hello";
 char str2[12] = "World";
  char str3[12];
 int  len ;
/* copy str1 into str3 */
        strcpy(str3, str1);
        printf("strcpy( str3, str1) :  %s\n", str3 );
/* concatenates str1 and str2 */
         strcat( str1, str2);
        printf("strcat( str1, str2):   %s\n", str1 );
/* total length of str1 after concatenation */
        len = strlen(str1);
         printf("strlen(str1) :  %d\n", len );
 return 0;
}
```

Strings: Other functions

- **strncmp(str1, str2, n) :** it returns 0 if the first n characters of str1 is equal to the first n characters of str2, less than 0 if str1 < str2, and greater than 0 if str1 > str2.

- **strncpy(str1, str2, n) :** This function is used to copy a string from another string. Copies the first n characters of str2 to str1

- **strstr(str1, str2):** it returns a pointer to the first occurrence of str2 in str1, or NULL if str2 not found.

- **strncat(str1, str2, n) :** Appends (concatenates) first n characters of str2 to the end of str1 and returns a pointer to str1.

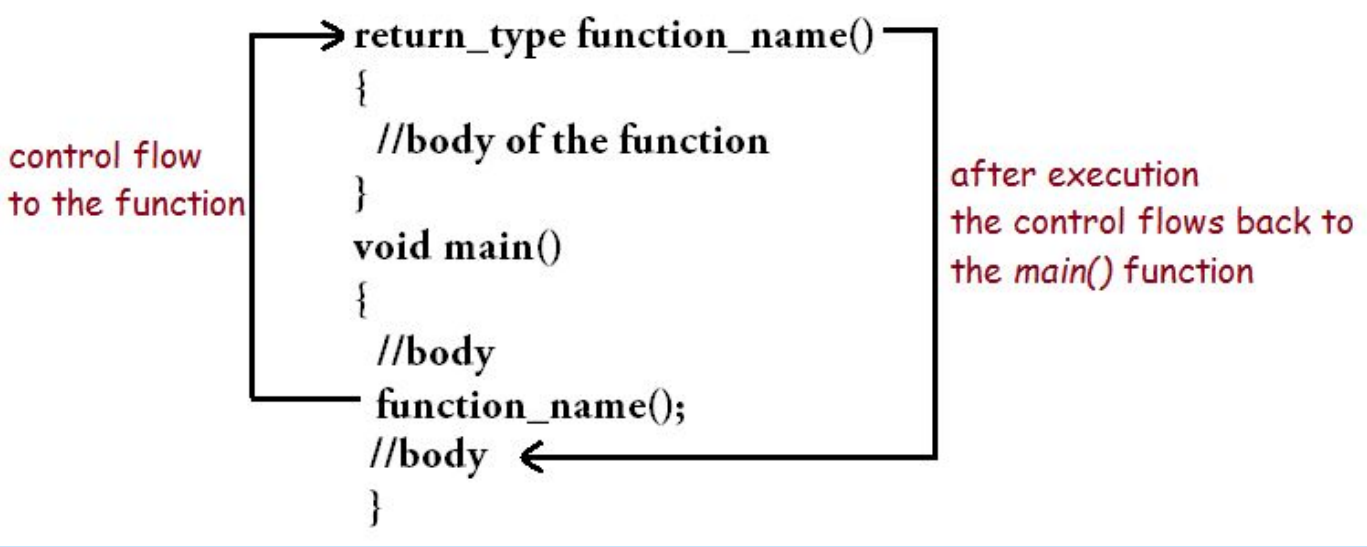
Functions

- 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.
- C functions can be classified into two categories
  - Library functions
  - User-defined functions

Functions: Benefits

- It provides **modularity** to your program's structure.
- In case of large programs with thousands of code lines, **debugging and editing becomes easier.**
- It makes the program **more readable** and **easy to understand**.
- It is used **to avoid rewriting same logic/code** again and again in a program.

Functions: Syntax



C functions aspects	syntax
function definition	Return_type function_name (arguments list) { Body of function; }
function call	function_name (arguments list);
function declaration	return_type function_name (argument list);

Functions: Parameters /Arguments

Formal Parameter: The parameter which is written **at the function definition** is known as a formal parameter.

Actual Parameter: The parameter that is written **at the function call** is known as the actual parameter.

```
int sum(int,int);                    Function Declaration/ Function Prototype
void main()
{     int a=10 ,b=20 ;
      sum(a ,b );//actual parameter          Calling Function   (Function Call)
}
int sum(int x, int y)//formal parameter     Called Function (Function
                                            Definition )
{
//body of the function
}
```

Functions: Categories

- Function with arguments and a return value
- Function with arguments and no return value
- Function with no arguments and no return value
- Function with no arguments and a return value

Functions: Category 1

1. With arguments and with return values	**function declaration:**
	int function (int);
	
	function call:
	z=function (a); ***Function with argument***
	
	function definition:
	int function(int a)
	{
	statements;
	return a; ***Function with return value***
	}

Functions: Category 2

2. With arguments and Without return values	**function declaration:**
	void function (int);
	
	function call:
	function(a); ***Function with argument***
	
	function definition:
	void function(int a)
	{
	statements; ***Function without return value***
	}

Functions: Category 3

3. Without arguments and without return values	**function declaration:**
	void function();
	
	function call:
	function(); ***Function without argument***
	
	function definition:
	void function()
	{
	statements; ***Function without return value***
	}

Functions: Category 4

4. Without arguments and With return values	**function declaration:**
	int function ( );
	
	function call:
	z= function ( ); ***Function without argument***
	function definition:
	int function( )
	{
	statements;
	return a; ***Function with return value***
	}

Structures

- **Structure is a user-defined data type** in C programming language ,**that combines logically related data items of different data types together.**

- **struct** keyword is used to declare the structure in C.

- Variables inside the structure are called **members of the structure**.

Structures: Syntax and Example

Syntax

```
struct struct_name
{
DataType    member1_name;
DataType    member2_name;
DataType    member3_name;
…
};
```

Example

```
struct employee
{
    char    name[50];
     int      age;
     float   salary;
};
```

Structures: Examples

```
struct student
{
    char   name[60];
    int    roll_no;
    float  marks;

} s1,s2,s3….sn;
```

Name of the structure

members of the structure

Declaring Structure variables s1,s2,s3….sn at the time of definition

```
struct student
{
     char   name[60];
     int    roll_no;
    float   marks;
};
```

```
void main()
{
   struct student s1,s2,s3.....sn; // Declaring Structure variables within a main function.


}
```

Structures: Accessing Structure Members

Using Dot(.) operator

Example1 :

```
#include <stdio.h>

 struct student
 {
      char   name[60];
      int    roll_no;
     float   marks;
 } s1;
```

/*Assigning the values to struct member here*/

s1. roll_no = 101;
s1. marks = 25.0

Example 2:

```
#include<stdio.h>

struct Point
{
int x, y;
};

int main()
{
struct Point p1 = {0, 1};

// Accessing members of point p1
p1.x = 20;
printf ("x = %d, y = %d", p1.x, p1.y);

return 0;
}
```

Structures: Examples

```
#include <stdio.h>

 struct tudent
{       char   name[60];
        int      roll_no;
        float    marks;
} s3={ "pavan", 103, 25};

int main()
{
    struct student  s1 = { "ramesh", 101, 23 };
    struct student  s2 = { "suresh", 102, 27 };

printf ("x = %s, y = %d, z = %f\n", s1.name, s1.roll_no, s1.marks);
printf ("x = %s, y = %d, z = %f\n", s2.name, s2.roll_no, s2.marks);

}
```

Structure member (S3) initialization

Structure member (S1 and S2) initialization

Structures: Arrays of Structures

/* Array of Structures in C Initialization */

```
struct Emp
{
int       age;
char    name[10];
int      salary;
} Employees[4] = { {25, "Suresh", 25000}, {24, "Tutorial", 28000}, {22, "Gateway", 35000}, {27, "Mike", 20000} };
```



```
Employees[0]   ={25,   "Suresh",   25000};
Employees[1]  =  {24,  "Tutorial",  28000};
Employees[2]  =  {22,  "Gateway",  35000};
Employees[3] =   {27, "Mike", 20000};
```

Structures: Arrays of Structures

Structure Definition

```
struct student
{
    int     age;
   char   name[10];
   int     marks;
 };
```

```
void main()
{

struct student  S[4];

S[4] = { {25, "Suresh", 25},
         {24, “Ramesh", 28},
         {22, “Anoop", 35},
         {27, “Arun", 20}
       };

}
```

Arrays of Structures

Initialization members

Examples: C Programs

- Write a C program to find the sum of array elements.
- Write a C program to implement a linear search technique.
- Write a C program to find the transpose of the matrix.
- Writ a C program to copy one string into another without using built in functions.
- Write a C program to check whether number is palindrome or not using a function with argument and with return value.
- Write a C program to add to complex numbers using structures.

Pointers

Pointers is a variable ,which stores the address of another variable.

•The purpose of pointer is to save memory space and achieve faster execution time.

- **& symbol** is used to **get the address of the variable**.
- *** symbol is** used to **get the value of the variable** that the pointer is pointing to.

	variable	Address	value
Normal Variable →	**q**	783592244	**50**
Pointer Variable →	***ptr**	1755463112	**783592244**

Normal variable stores the value ,whereas **pointer variable stores the address of the variable.**

Pointers: Declaring a Pointer

A pointer declaration has the following form.

data_type * pointer_variable_name;

- **data_type** is the pointer's base type of C's variable types and indicates the type of the variable that the pointer points to.

- **The asterisk** (*: the same asterisk used for multiplication) which is indirection operator, declares a pointer.

```
1  int      *ptr1;    /* pointer to an integer */
2  double   *ptr2;    /* pointer to a double */
3  float    *ptr3;    /* pointer to a float */
4  char     *ch1 ;    /* pointer to a character */
5
```

Pointers: Initializing a Pointer

To get the address of a variable, we use the ampersand (&)operator, placed before the name of a variable whose address we need. Pointer initialization is done with the following syntax.

pointer = &variable;

```c
#include <stdio.h>
void main()
{
   int a=10;    //variable declaration
   int *p;      //pointer variable declaration

   p=&a;        //store address of variable a in pointer p

   printf("Address stored in a variable p is:%x\n",p);  //accessing the address
   printf("Value stored in a variable p is:%d\n",*p);   //accessing the value
}
```

Pointers: Accessing the Address of a Variable

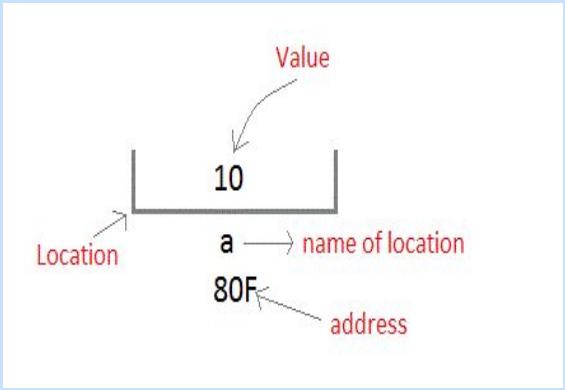
Whenever **a variable is defined in C language, a memory location is assigned for it, in which it's value will be stored**.

We can easily check this memory address, using the & symbol.

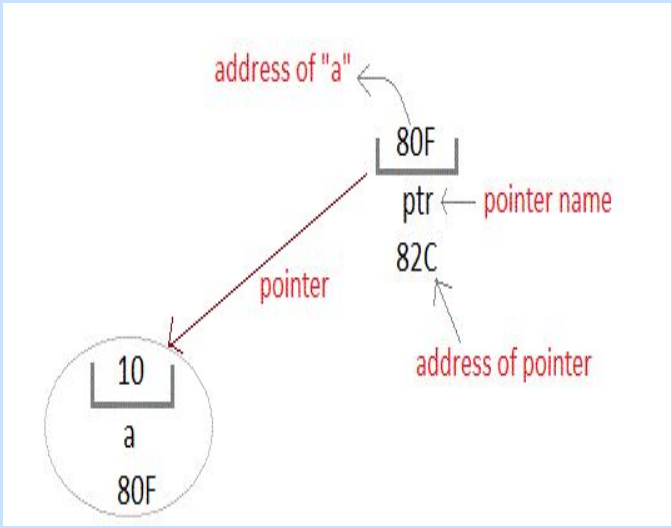
Assume **memory location 80F** for a **variable a.**

We can access the value 10 either by using the variable name a or by using its address 80F.

int a = 10;



A pointer variable is therefore nothing but a variable which holds an address of some other variable. And the value of a pointer variable gets stored in another memory location.



Pointers: Accessing a variable through its pointer

STEPS:

- Declare a normal variable, assign the value
- **Declare a pointer variable with the same type as the normal variable**
- Initialize the pointer variable with the address of normal variable
- Access the value of the variable by using asterisk (*) - it is known as dereference operator

```
1  #include <stdio.h>
2  void main()
3  {
4      //normal variable
5      int num = 100;
6
7      //pointer variable
8      int *ptr;
9
10     //pointer initialization
11     ptr = &num;
12
13     //pritning the value
14     printf("value of num = %d\n", *ptr);
15
16 }
```

Pointers

```c
#include <stdio.h>
void main()
{
   /* q is a normal variable and *ptr is pointer variable */
   int *ptr, q;

   q = 50;

   /* address of q is assigned to ptr */
   ptr = &q;

   /* display q's value using ptr variable */
   printf("%d\n", *ptr);   //50

   printf("%d\n", &ptr);   //1755463112
   printf("%d\n",    q);   //50
   printf("%d\n",   &q);  //783592244

}
```

Pointers

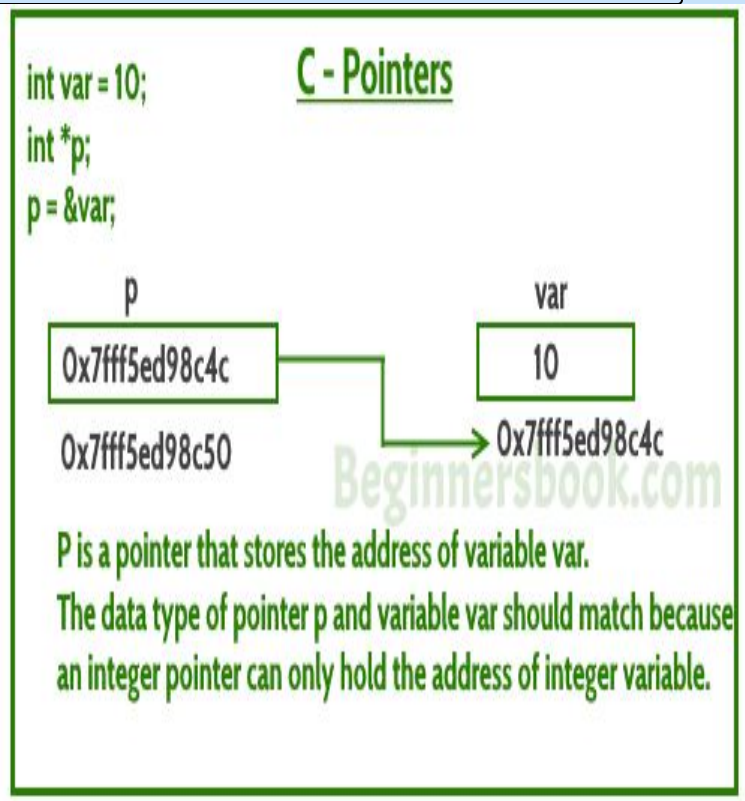
```c
#include <stdio.h>

int main () {

   int  var = 20;   /* actual variable declaration */
   int  *ip;        /* pointer variable declaration */

   ip = &var;  /* store address of var in pointer variable*/

   printf("Address of var variable: %x\n", &var  );

   /* address stored in pointer variable */
   printf("Address stored in ip variable: %x\n", ip );

   /* access the value using the pointer */
   printf("Value of *ip variable: %d\n", *ip );

   return 0;
}
```

Pointers: Benefits(use) of pointers in C

1. Pointers provide **direct access to memory**
2. Pointers provide a **way to return more than one value** to the functions
3. **Reduces the storage space and complexity** of the program
4. **Reduces the execution time** of the program
5. Pointers allows us to **perform dynamic memory allocation and deallocation.**
6. Pointers **helps us to build complex data structures** like linked list, stack, queues, trees, graphs etc.
7. Pointers allows us to **resize the dynamically allocated memory block**.

Pointers

```c
#include <stdio.h>
void main()
{
   /* Pointer of integer type, this can hold the
    address of a integer type variable.*/
   int *p;

   int var = 10;

   /* Assigning the address of variable var to the pointer
    p. The p can hold the address of var because var is
    an integer type variable.*/
   p= &var;

   printf("Value of variable var is: %d",     var);
   printf("\nValue of variable var is: %d",   *p);
   printf("\nAddress of variable var is: %p", &var);
   printf("\nAddress of variable var is: %p", p);
   printf("\nAddress of pointer p is: %p",    &p);

}
```




```
Value of variable var is: 10
Value of variable var is: 10
Address of variable var is: 0x7fff5ed98c4c
Address of variable var is: 0x7fff5ed98c4c
Address of pointer p is: 0x7fff5ed98c50
```

Pointers

A limited set of arithmetic operations can be performed on pointers. A pointer may be:

1.Incremented (++)

2.Decremented (—)

3.An integer may be added to a pointer (+ or +=)

4.an integer may be subtracted from a pointer

(– or -=)

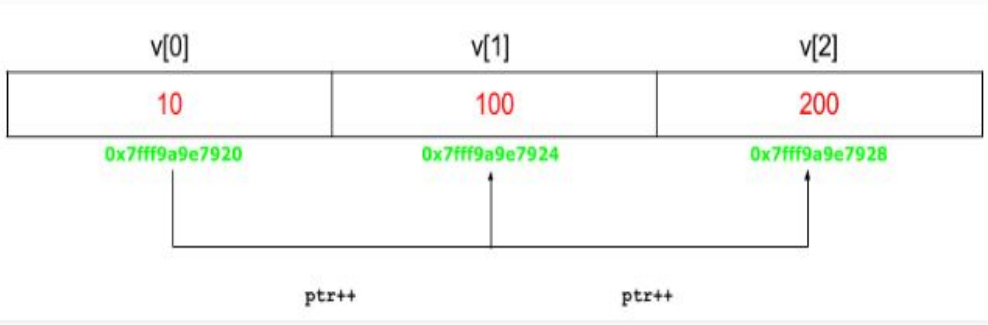
```c
#include <stdio.h>
void main()
{
    int a = 50; // Initializing integer variable

    int *ptr_a; // Declaring pointer variable

    ptr_a = &a; // Initializing pointer variable

    printf( "Before increment a = %d\n", *ptr_a);
    (*ptr_a)++;   // Unary increment operation
    printf( "After increment a = %d\n", *ptr_a);

    printf("Before decrement a = %d\n", *ptr_a);
    (*ptr_a)--; // unary decrement operation
    printf("After decrement a=%d", *ptr_a);

}
```

```
Before increment a = 50
After increment a = 51
Before decrement a = 51
After decrement a=50
```

Pointers

```c
#include <stdio.h>

void main()
{

    int v[3] = {10, 100, 200}; // Declare an array


    int *ptr; // Declare pointer variable


    ptr = v; // Assign the address of v[0] to ptr

    for (int i = 0; i < 3; i++)
    {
        printf("Value of *ptr = %d\n", *ptr);
        printf("Value of ptr = %p\n\n", ptr);
        ptr++; // Increment pointer ptr by 1
    }
}
```

```
Output:Value of *ptr = 10
Value of ptr = 0x7ffcae30c710

Value of *ptr = 100
Value of ptr = 0x7ffcae30c714

Value of *ptr = 200
Value of ptr = 0x7ffcae30c718
```



Pointers and Arrays

```c
#include <stdio.h>
int main()
{
    int a[5]={1,2,3,4,5};   //array initialization
    int *ptr;     //pointer declaration
                /*the ptr points to the first element of the array*/

    ptr=a; /*We can also type simply ptr==&a[0] */

    printf("Printing the array elements using pointer\n");
    for(int i=0;i<5;i++)
    {
            printf("\n%x",*p);  //printing array elements
            p++;    //incrementing to the next element, you can also write p=p+1
    }
    return 0;
}
```

Pointers and Strings

```c
#include <stdio.h>
#include <string.h>
int main()
{

char str[]="Hello MSRIT";
char *p;
int i;

p=str;

printf("Printing all the characters in a string\n");
   for(i=0;i<strlen(str);i++)
   {
      printf("%c\n",*p);
      p++;
   }

}
```

- C-program using pointers to determine the length of a character String.

```c
#include<stdio.h>
#include<string.h>
void main ()
{
    char str [20],*p;
    int l=0;

    printf ("enter a string \n");
    scanf ("%s", str);

    p=str;

    while(*p!='\0')
    {
      l++;
      p++;
    }
    printf ("the length of the given string is %d", l);
}
```

- Pointers: Call by reference

```c
#include <stdio.h>
void swap(int *a, int *b); //Function Declration
int main()
{
    int m = 10, n = 20;
    printf("m = %d\n and n = %d\n", m,n);

    swap(&m, &n);    //Function Call
    printf("After Swapping: m = %d and n = %d\n:",m,n);
}

void swap(int *a, int *b) //Function Definition
{
    int temp;
    temp = *a;
    *a = *b;
    *b = temp;
}
```