BET4302

Data structures and Programming

UNIT I

Fundamentals of C Programming

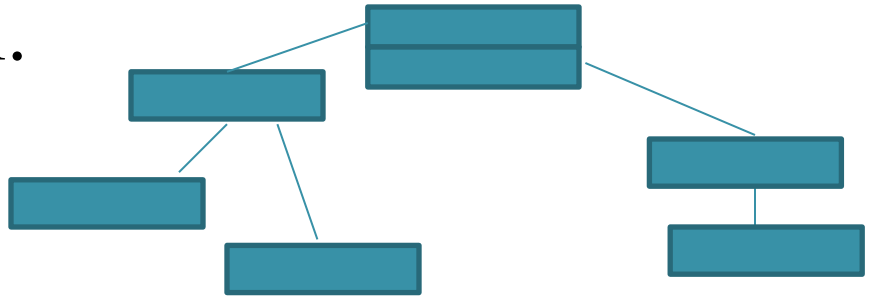
Syllabus

Unit I Introduction to C Programming

Constants, variables and keywords in C, Data Types, Operators, Control structure, Arrays, Pointers and String manipulation, structure, Union, Functions: parameter passing, call by value and call by reference, scope rules.

What is data structure

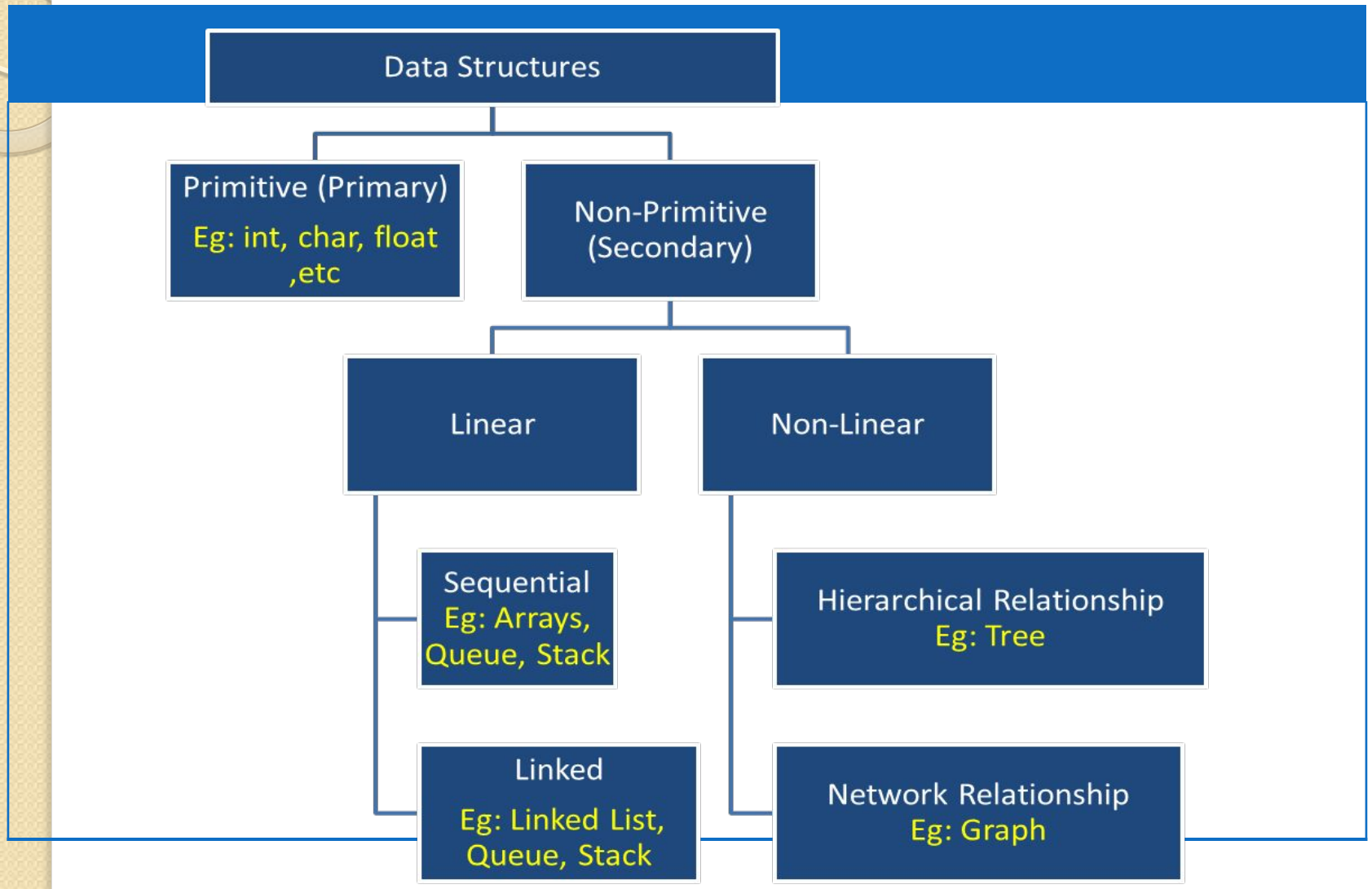
- Data structure is the organization of data through different ways.
- It can be organized through logical or mathematical model.



Linear

Non Linear

What is data structure



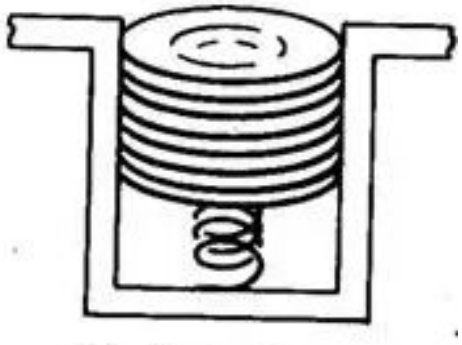
1) Array

1 D

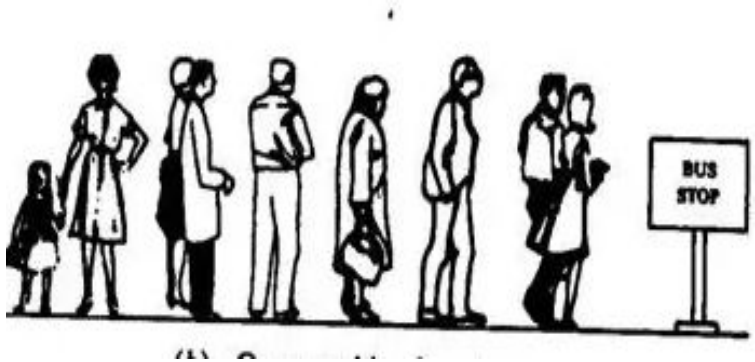
2D

---

2) Stack



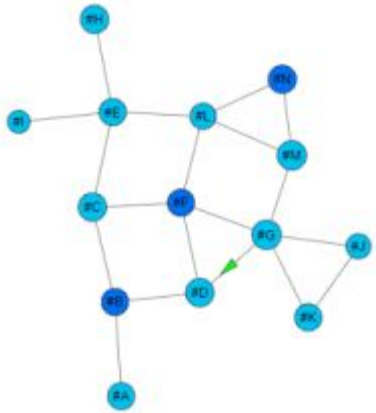
3)Queue



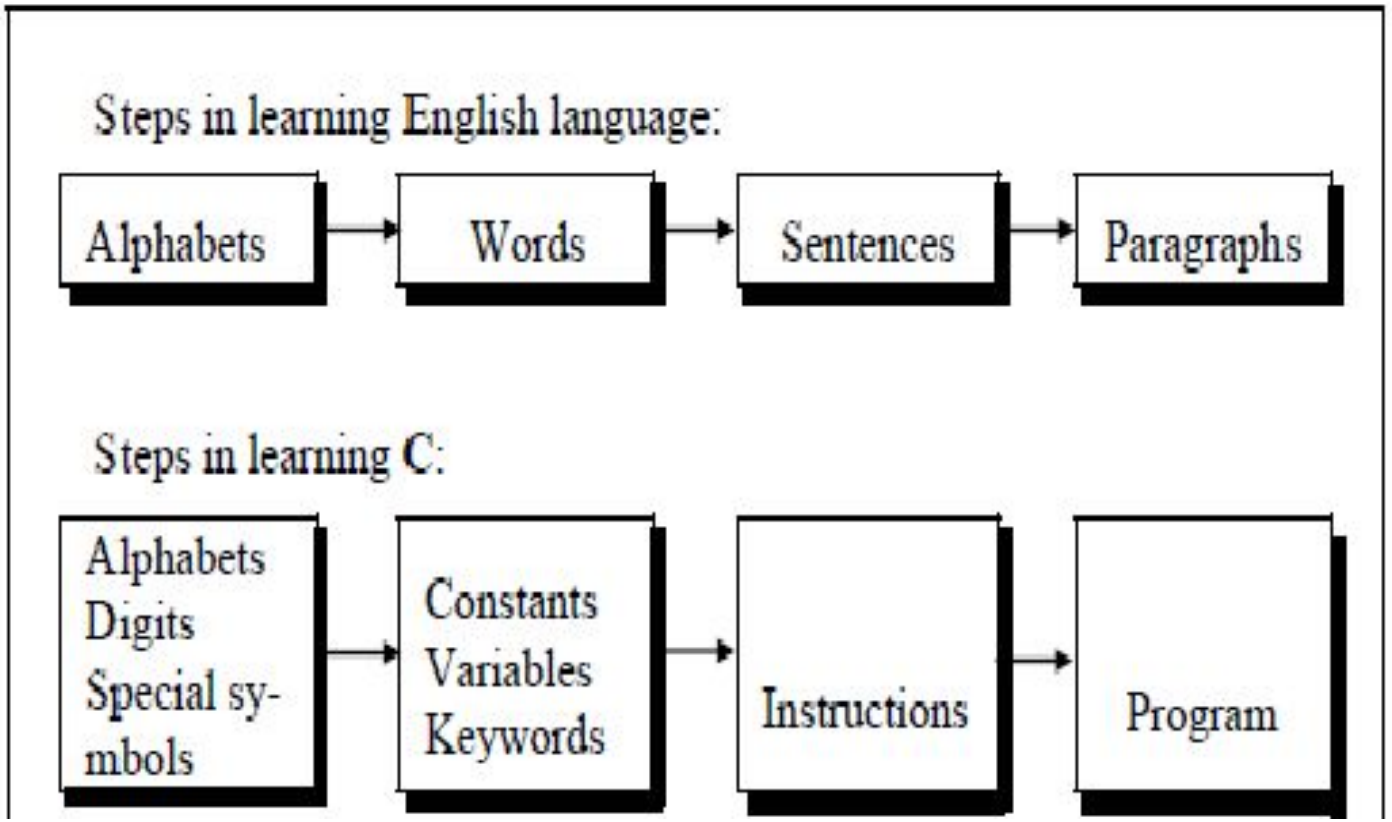
- Linked list

- Tree

- Graph



Introduction to C Programming

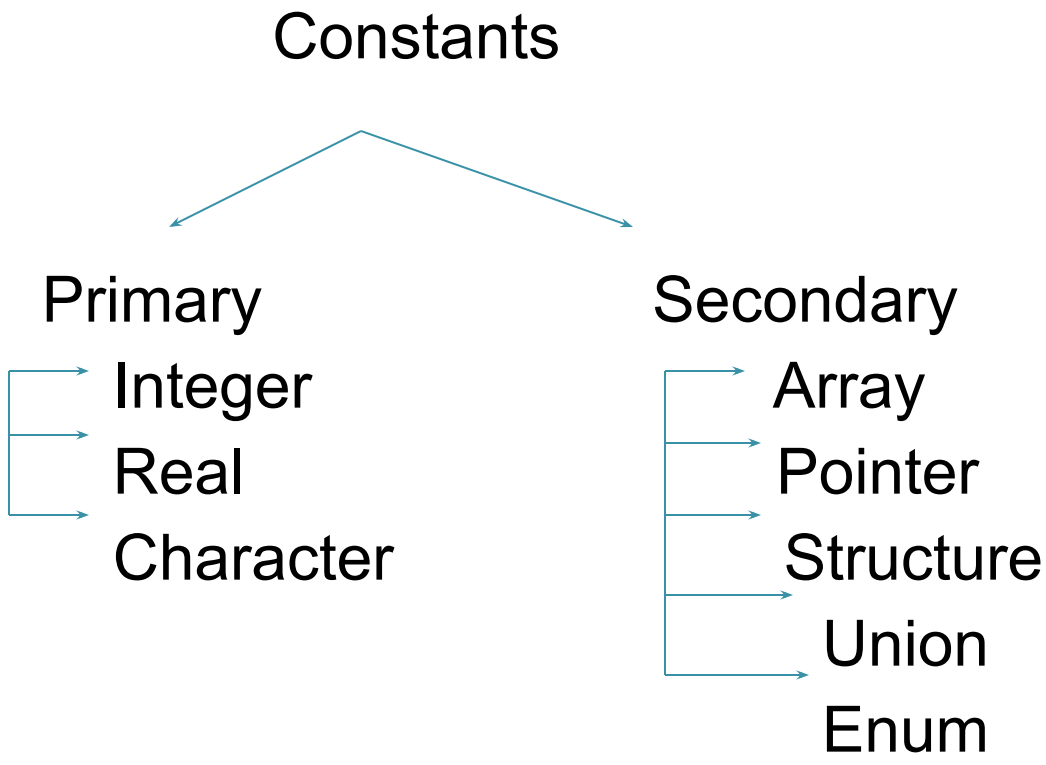


Introduction to C Programming

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

Constants and Variables

- Constant is a entity that doesn't change value
- Variable is a entity that may change value



Constants and Variables

Constants

a=10
a=10

Const a=10

Constants and Variables

Constants

a=10
a=10

Variables:

a=10
a=20

Int a=10;

a=20;

Keywords

1. Keywords are the reserved words whose meaning has already defined in the compiler.
2. Thus these words are not used as name of variable.
3. 32 keywords are available in C language.

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

std.Pgm template

```
#include <stdio.h>  //stdio=standard input/output.header file
#include<conio.h> //conio=console input/output.header file
void main()
{
clrscr();
scanf();  //Inputting some data
                //& required in since data need to be stored in memory
printf();//Outputting /displaying the data
              //& not required in since data need to be stored in memory
getch();
 }

void =return nothing or zero
```

Simple program

```
//a simple program that has variables

// single line comments

/* this is uses as
 a multiple line comments */

#include <stdio.h>   //Preprocessor directive
int  main()      // void main()
{
  char y;              //(1 Byte=8 bits)
  int x;               //(2Bytes=16 bits)
  float z;            //(4Bytes=32 bits)
  double t;          //(8Bytes=64 bits)

    printf("hello world…\n");

    int test;  //wrong, The variable declaration must appear first
    return 0;   // if void is used no need of this statement.
}
```

Bit Level Programming

- Bitwise logical operators
  - Bitwise AND &
  - Bitwise OR |
  - Bitwise Exclusive OR ^
- Bitwise shift operators
  - Right shift >>
  - Left shift <<
- One's complement operators
  - ~

Bitwise logical operators

a	b	a& b	a | b	a^b
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

x=21=0001 0101

y=37=0010 0101

Bitwise AND &

x & y = 0001 0101

0010 0101

00000101 =5

Bitwise OR |

x | y = 00111010=53

Bitwise Exclusive OR ^

x ^ y =00110000=48

Bitwise shift operators

- Right shift >>
- Left shift <<

x=21=0001 0101
y=37=0010 0101

Left shift <<
x<<1 =00101010=21*2=42
Y<<1=010 01010=37*2=74

Right shift >>
x >>1=00001010=21/2=10
Y>>1= 00010010=37/2=18

Bitwise shift operators

- Right shift >>
- Left shift <<

x=21=0001 0101
y=37=0010 0101

Left shift <<
x<<2 =01010100=21*4=84
Y<<2=10 010100=37*4=148

Right shift >>
x >>2=00000101=21/4=5
Y>>2= 00001001=37/4=9

One's complement operators

- ~ is a unary operator.
- inverts all the bits represented by its operand.
- often combines with the bitwise operator to turn off a particular bit.
- a=01 then ~a=10
- flag= a& ~a; =00

Bitwise operators

```
#include<stdio.h>
void main()
{
int a=10,b=20,c,d,e,f,g,h;
c=a&b;   // Bitwise AND
d=a|b;      // Bitwise   OR
e=~a;     // Bitwise  1's Complement operator
h=~b;
f=a>>1;    // Bitwise  Right shift
g=a<<1;   // // Bitwise  Left shift
printf("\n c=%d \n d=%d",c,d);
printf("\n e=%d \n h=%d",e,h);
printf("\n f=%d \n g=%d",f,g);
}
```

o/p

c=0

d=30

e=-11

h=-21

f=5

g=20

SWAP USING BITWISE OPERATORS [WITHOUT USING TEMP VARIABLE & POINTERS

- a=3 =011
- b=4=100

Method 1

- a=3 =011
- b=4=100

1. a=a^b=111=7
2. b=a^b=011=3
3. a=a^b=100=4

a=4,b=3

Method 2

- a=3 =011
- b=4=100

1. a=a+b=3+4=7
2. b=a-b=7-4=3
3. a=a-b=7-3=4

a=4,b=3

Method 3

- a=3 =011
- b=4=100

1. a=a+b=3+4=7
2. b=a-b=7-4=3
3. a=a-b=7-3=4

a=4,b=3

Control statements

- If..else
- Conditional operator
- Switch case
- Do…while
- While
- for

If..else

```
#include <stdio.h>
#include<conio.h>
void main()
{
int salary;
char grade;

clrscr();

printf("\nEnter the salary");
scanf("%d",&salary);
if (salary>10000)
    grade='A';
else
     grade='B';
printf("%c",grade);
 getch();
 }
```

if else if else statement

```
#include <stdio.h>
int main()
{
    int age;                        /* Need a variable... */
    printf( "Please enter your age" );  /* Asks for age */
    scanf( "%d", &age );                /* The input is put in age */

    if ( age < 100 )                    /* If the age is less than 100 */
      {
     printf ("You are pretty young!\n" ); /* Just to show you it works... */
    }
    else if ( age == 100 )                  /* I use else just to show an example */
      {
     printf( "You are old\n" );
    }
    else                                /* Executed if no other statement is*/
      {
     printf( "You are really old\n" );
    }
    return 0;
}
```

Conditional operator(ternary operator)

- replacement of if..else
- Syntax: exp1?exp2:exp3;

```
#include <stdio.h>
void main()
{
int salary;
char grade;
printf("\nEnter the salary");
scanf("%d",&salary);
grade=(salary >10000)?'A':'B';
printf("%c",grade);
}
```

While

It will execute only if the condition is true

```
#include <stdio.h>
int main()
{
  int x = 0;
while ( x < 10 )
{
      printf( "%d\n", x );
      x++;
  }
  return 0;
}
```

o/p

0
1
2
3
4
5
6
7
8
9

Do…while

```
/* It will execute at least one time even though the
   condition is false*/

#include <stdio.h>
int main()
{
  int x;
  x = 0;
  do
  {
   printf( "%d\n", x );
   x++;
  } while ( x != 10 );
  return 0;
}
```

o/p

0
1
2
3
4
5
6
7
8
9

for

For(exp1:exp2:exp3)

for(Initialization statement; conditional statement;Increment/decrement)
{

}

```
#include <stdio.h>
int main()
{
    int x;
   for ( x = 0; x < 10; x++ )
    {
    printf( "%d\n", x );
    }
    return 0;
}
```

o/p

0
1
2
3
4
5
6
7
8
9

Switch case

- replacement of multiple if , when multiple conditions exists.

```
#include<stdio.h>
#include<conio.h>
int main()
{
    int age;
    printf("Please enter your age");                    /* Asks for age */
    scanf("%d",&age);                                    /* The input is put in age */

switch(age)
     {
    case 20:   printf ("You are pretty young!\n");
                  break;

    case 100 :  printf("You are old\n");
                  break;

    default:    printf("You are really old\n");
                break;
    }

    return 0;
}
```

Jump statements

- break –helps to come out of the loop
- continue-go back again in to the beginning of the loop
- goto-helps to jump from one statement to the other statement

break

```
#include <stdio.h>
int main()
{
  int x;
  for(x=0;x<10;x++)
  {
       if(x==5)
       {
         break;
       }
       printf("%d\n",x);

  }
  return 0;
}
```

```
0
1
2
3
4
```

continue

```
#include <stdio.h>
int main()
{
  int x;
  for(x=0;x<10;x++)
  {
      if(x==5)
      {
        continue;
      }
      printf("%d\n",x);
  }
  return 0;
}
```

```
0
1
2
3
4
6
7
8
9
```

goto

```
#include<stdio.h>
#include<conio.h>
void main()
{
int i,j;
for(i=1;i<=2;i++)
{
 for(j=1;j<=2;j++)
  {
  if(i==j)
        goto label1;
 // else
       printf("\n i and j are different :i=%d   j=%d\n",i,j);
  }
}
label1:  printf ("\n Both i and j are same: i=%d  j= %d \n",i,j);
getch();
}
```

/*output*/

Both i and j are same: i=1 j=1

Arrays

1. *An array is a data structure*
2. *used to process multiple elements with the same data type when a number of such elements are known.*
3. *An array is a composite data structure; that means it had to be constructed from basic data types such as array integers.*

```
int a[5], i;
for(i = 0;i<5;i++)
        {
        a[i]=i;
        }
```

Arrays

TWO-DIMENSIONAL ARRAY

int a[3][2];

3	1
5	2
8	7

```
int a[3][2];
for(int i = 0;i<3;i++)
    for(int j=0;j<2 ;j++)
    {
            {
                    a[i][j]=i+j+i*j;
            }
    }
```

FUNCTION

FUNCTION

- provide modularity to the software
- divide complex tasks into small manageable tasks
- avoid duplication of work

```
int add (int x, int y)
{
    int z;
    z = x + y;
    return z;
}
```

FUNCTION

THE SEQUENCE OF EXECUTION DURING A FUNCTION CALL

- When the function is called, the current execution is temporarily stopped and the control goes to the called function. After the call, the execution resumes from the point at which the execution is stopped.
- To get the exact point at which execution is resumed, the address of the next instruction is stored in the stack. When the function call completes, the address at the top of the stack is taken.

FUNCTION

THE SEQUENCE OF EXECUTION DURING A FUNCTION CALL

- Functions or sub-programs are implemented using a stack.
- When a function is called, the address of the next instruction is pushed into the stack.
- When the function is finished, the address for execution is taken by using the pop operation.

FUNCTION

THE SEQUENCE OF EXECUTION DURING A FUNCTION CALL

- Result:?

```
main ( )
{
     printf ("1 \n");
     printf ("2 \n");
     f1();
     printf ("3 \n");
     printf ("4 \n");
}
void f1()
{
     printf ("f1-5 \n");
     printf ("f1-6 \n");
     f2 ( );
     printf ("f1-7 \n");
     printf ("f1-8 \n");
}
void f2 ()
{
     printf ("f2-9 \n");
     printf ("f2-10 \n");
}
```

FUNCTION

PARAMETER * REFERENCE PASSING

- *passing by value*
  - the value before and after the call remains the same
- *passing by reference*
  - *changed value after the function completes*

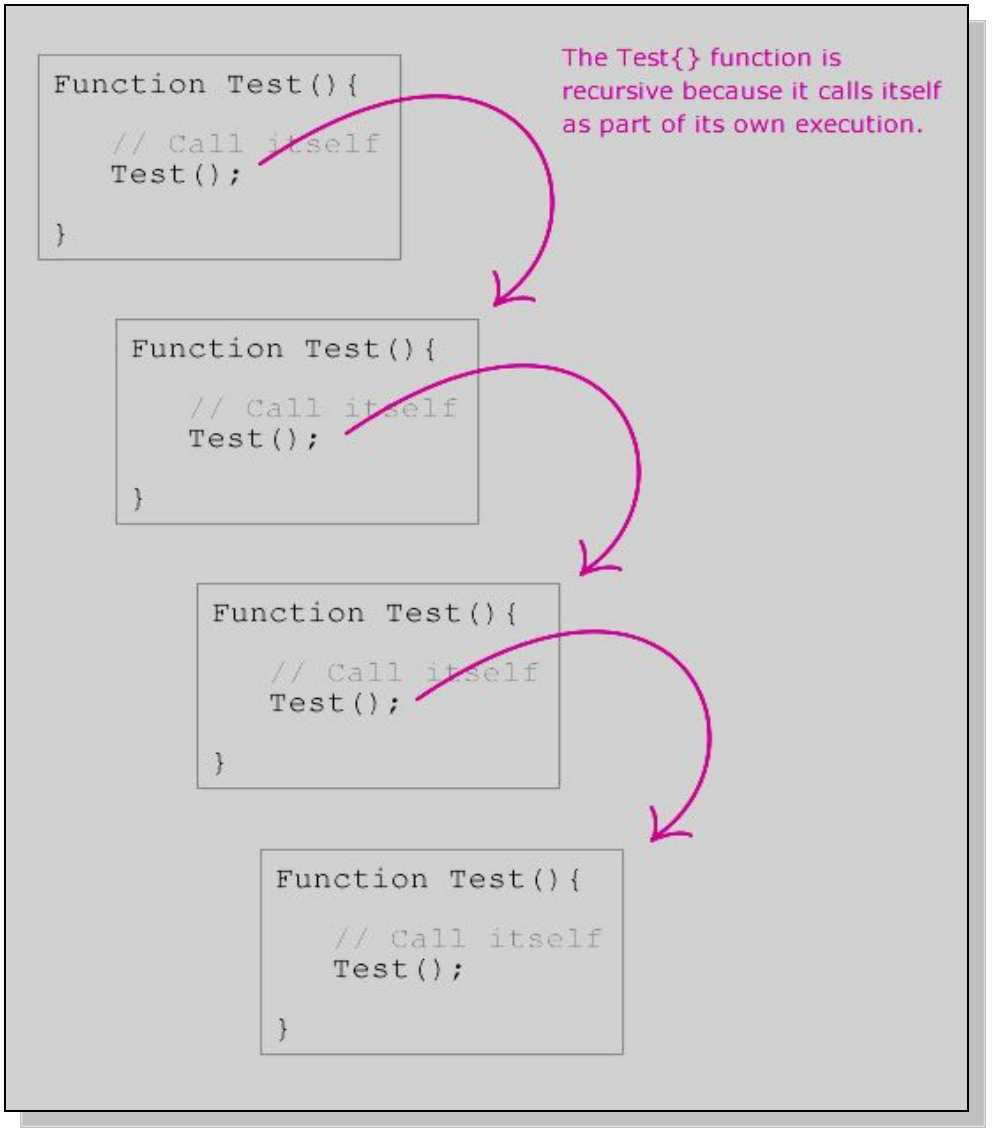
```
void (int *k)
{
     *k = *k + 10;
}
```

```
void (int &k)
{
     k = k + 10;
}
```

RECURSION

RECURSION

- A method of programming whereby a function directly or indirectly calls itself
- Problems: stop recursion?



RECURSION

Example: Factorial

```
int factorial(int n)  // assumes n >= 0
{
  if (n == 0)
    return 1;
  else
    return n * factorial(n-1);
}
```

To see how the computation is done, trace `factorial(3)`:

```
factorial(3) = 3 * factorial(2)
             = 3 * (2 * factorial(1))
             = 3 * (2 * (1 * factorial(0)))
             = 3 * (2 * (1 * 1))
```

RECURSION

```
Let n=4
int factorial (int 4)
{
if n<=1
return 1;
else
return (4*factorial(int 3);
}
```

RECURSION

Let n=4

```
int factorial (int 4)
{
if n==1
return 1;
else
return (4*
}
```

```
int factorial (int 3)
{
if n==1
return 1;
else
return (3* factorial (int 2);
}
```

RECURSION

Let n=4

```
int factorial (int 4)
{
if n==1
return 1;
else
return (4*
}
```

```
int factorial (int 3)
{
if n==1
return 1;
else
return (3*
}
```

```
int factorial (int 2)
{
if n==1
return 1;
```

RECURSION

Let n=4

```
int factorial (int 4)
{
if n==1
return 1;
else
return (4*
}
```

```
int factorial (int 3)
{
if n==1
return 1;
else
return (3*
}
```

```
int factorial (int 2)
{
if n==1
return 1;
```

RECURSION

```
Let n=4
int factorial (int 4)
{
if n==1
return 1;
else
return (4*
}
```

```
int factorial (int 3)
{
if n==1
return 1;
else
return (3*
```

```
int factorial (int 2)
```

RECURSION

Let n=4

```
int factorial (int 4)
{
if n==1
return 1;
else
return (4*
}
```

```
int factorial (int 3)
{
if n==1
return 1;
else
return (3* 2);
```

RECURSION

```
Let n=4
int factorial (int 4)
{
if n==1
return 1;
else
return (4* 6);                    ;
}
```

Recursion Excercises

- Minesweeper

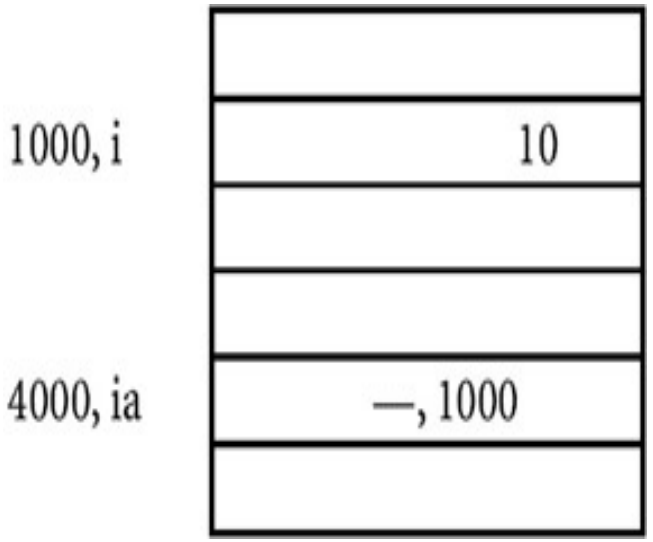
Pointers

1. ***is a variable whose value is also an address.***
2. ***A pointer to an integer is a variable that can store the address of that integer***

ia: value of variable

&ia: address of ia

***ia means you are printing the value at the location specified by ia**



Pointers

```
int i;          //A
int * ia;       //B
i = 10;         //C
ia = &i;    //D

 printf("The address of i =, %d", &i );
 printf(" The value of i=, %d", i );
 printf("The address of ia =, %d", &ia );
 printf(" The value of ia=, %d", ia );
 printf(" The value of  *ia=, %d", *ia );
```

Pointers

Points to Remember

- **Pointers give a facility to access the value of a variable indirectly.**
- **You can define a pointer by including a * before the name of the variable.**
- **You can get the address where a variable is stored by using &.**

p1

```
void main ()
{
Int  x=5 ;
printf (“\nThe value of variable ‘x’ = %d”,x);
printf(“\nThe address of x is= %d”, &x);
}
```

OUT PUT:

The value of variable ‘x’ = 5

The address of x is= 1000

p2

```
void main ()
{
int  i=2 ;
int *p;//ptr declaration
p= &i; //ptr initialization &i=1000
printf (“\n %d”,i);
printf (“\n %d”,&i);
printf (“\n %d”,*p);
printf (“\n %d”,p);
}
```

OUT PUT:

2

1000

2

1000

p3

```
void main ()
{
int  x=10,y=20, z[5] ;
int  *xp;   //&x=1000   &z[0]=1008
xp=&x;
printf ("\n %d",xp);
y=*xp;
printf ("\n %d",y);
*xp=0;
printf ("\n %d",*xp);
xp=&z[0];
printf ("\n %d",xp);
}
```

OUT PUT:

1000

10

0

1008

p4

```
void main ()
{
int  x=1,y=2 ;

int  *xp,*yp;
// &x=1000   &y=2000
xp=&x;
printf (“\n %d”,xp);
yp=&y;
printf (“\n %d”,yp);
y=*yp+1 ;
printf (“\n %d”,y);
yp=xp;
printf (“\n %d”,yp);
yp=&y;
```

OUT PUT:

1000

2000

3

1000

2000

1

Find out the output of the following program. Justify your answers in steps.

```
main( )
{
int n[25];
n[0]=100;
n[24]=200;

printf("%d\n %d",*n,*(n+24)+*(n+0));
}
```

What is the output of the following program?

```
int f(int);
void main()
{
int s;
s=f(1234);
printf("%d",s);
}
int f(int n)
{
if(n==0)
      return (0);
else
    return (n%10+f(n/10));
}
```

What will be the o/p of the following program? Explain.

```
void sp(int, int, int*, int*);

void main()
{ int a=10,b=20,sum, prod;
sp(a, b, &sum, &prod);
printf("%d\n%d",sum, prod);
}
void sp(int x, int y, int *s, int *p)
{
*s=x+y;
*p=x*y;
}
```

POINTER ARRAYS

- *You can define a pointer array (similarly to an array of integers).*
- *In the pointer array, the array elements store the pointer that points to integer values.*

```
int *a[5];
main()
{    int i1=4,i2=3,i3=2,i4=1,i5=0;
     a[0]=&i1;
     a[1]=&i2;
     a[2]=&i3;
     a[3]=&i4;
     a[4]=&i5;

     printarr(a);
     printarr_usingptr(a);
}
```

STRUCTURES

- *Structures are used when you want to process data of multiple data types but you still want to refer to the data as a single entity*
- *Access data: structurename.membername*

Example

```
struct  employee{
                char name[10];
                char address[20];
                int age;
                float salary;
               } emp1;
```

or

```
struct  employee{
                char name[10];
                char address[20];
                int age;
                float salary;
               } emp1,emp2;
```

or

```
struct  employee{
                char name[10];
                char address[20];
                int age;
                float salary;
               } ;
struct employee emp1,emp2;
```

Example

```
struct  employee //Model for structure
{
    char name[10];
    char address[20];
    int age;
    float salary;
} ;    //compiler does not allocate memory
struct employee emp1; // memory is allocated to
variable emp1  like int a;
```

Accessing the members

using

1. dot operator(.)
2. pointer operator(->)

syntax:

structure_name.member_name

```
eg: struct  employee{
            char name[10];
            char address[20];
            int age;
            float salary;
              } emp1;

emp1.name;
emp1.age;
```

Example

```
main()
{
struct  sample{
            int a;
            float b;
            char c;
             };
struct  sample sam1;
printf("\nEnter the values of a, b, c");
scanf("%d    %f    %c",&sam1.a, &sam1.b, &sam1.c);
printf("\n  The value of a is %d",sam1.a);
printf("\n  The value of b is %f",sam1.b);
printf("\n  The value of c is %c",sam1.c);
}
```

Nested structure

```
struct address
{
int flatno;
char society_name[10];
char city[20];
int pin_no;
}addr;

struct  employee //Model for structure
{
      char name[10];      //emp1.name
      struct address addr;    //emp1.addr.flatno
      int age;
      float salary;
} emp1;
```

Comparison with arrays & structure

Sr. No.	Arrays	Structures
1	Used for storing homogeneous data(same data type)	Used for storing heterogeneous data (different data type)
2	Accessed in either indexed mode or using pointers. Array elements are accessed using the index operator eg: a[0]	Members of a structure are accessed using the: dot (.) operator or pointer (->) operator
3	Linear data structure can be conveniently handled using array	Non-Linear data structure like tree, graph etc , can be conveniently handled using structures.
4	All elements in an array are of same data type.	Complex data types can be declared as a composition of different data types and grouped together in a structure.
5	Syntax: data_type array_name[size]; Eg: int a[20];	Syntax: struct structure_type { datatype variables; }structure_variables;

Array of structures

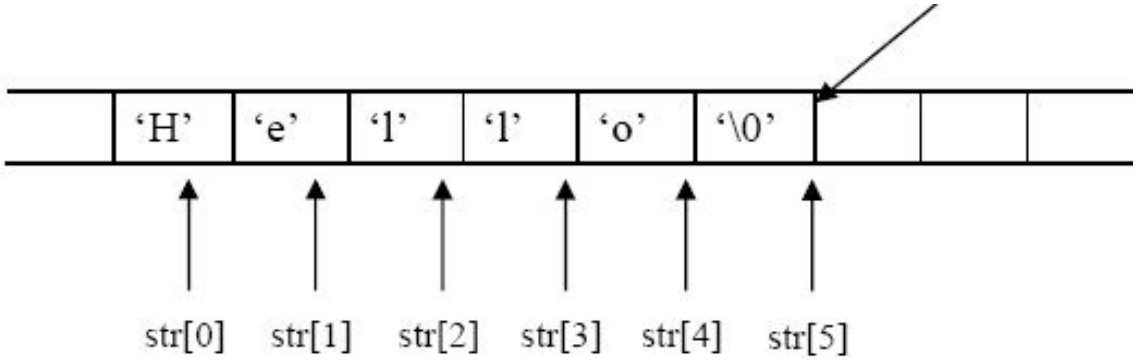
```
struct student
{
    int rollno;
     char name[10];
    float marks;
} std[10];  //array of structure
void main()
{
int i,n;
printf("\n Enter the total no. of students");
scanf("%d",&n);
printf("\n Enter the Rollno, Name & marks  of students");
for(i=0;i<n;i++)
    {
     scanf("%d %s %f ",&std[i].rollno, std[i].name, &std[i].marks);
    printf("\n %d %s %f ",std[i].rollno, std[i].name, std[i].marks);
    }
}
```

Pointer to structures

```
struct  student
{          int rollno;
           char name[10];
          float marks;
} std1;
void main()
{
int n;
struct student *pt;  //pointer to structure
printf("\n Enter the total no. of students");
scanf("%d",&n);
printf("\n Enter the Rollno, Name & marks  of students");
scanf("%d %s %f ",&std1.rollno, std1.name, &std1.marks);
printf("\n %d %s %f ",std1.rollno, std1.name, std1.marks); //using structure variable
pt=&std1;
printf("\n %d %s %f ",pt->rollno, pt->name, pt-> marks); //using pointer variable
}
```

String: structure

- String
  - is array of char
  - Ending with null char \0 (size +1)
  - Example: store 10 chars:
    - char str[11];
  - "Example": string const. C/C++ add \0 automayically

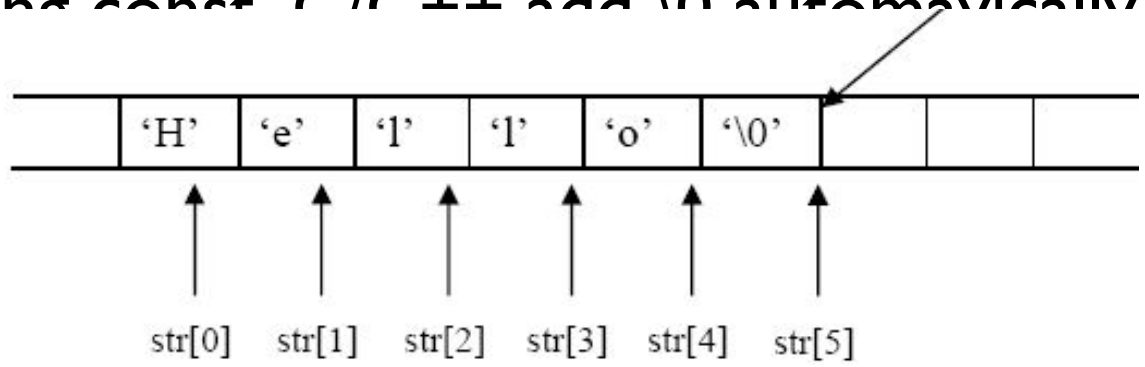


String: declare

- Declare string
  - Using array of chars
    - char str[] = {'H','e','l','l','o','\0'}; //declare with null
    - char str[] = "Hello"; //needn't null
  - Using char pointer
    - char *str = "Hello";

String: structure

- String
  - is array of char
  - Ending with null char \0 (size +1)
  - Example: store 10 chars:
    - char str[11];
  - "Example": string const, C/C++ add \0 automaxically



String: declare

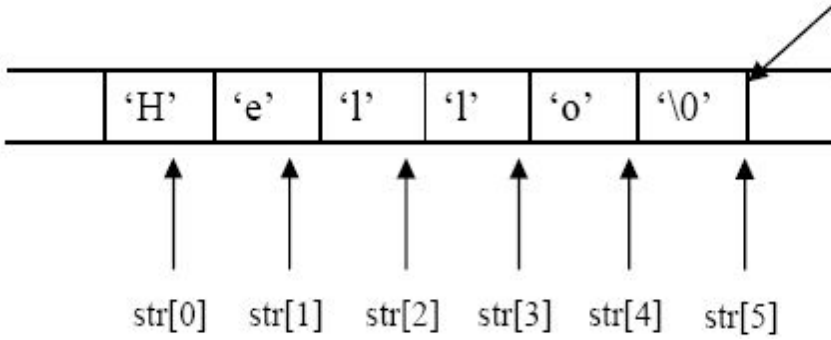
- Declare string
  - Using array of chars
    - char str[] = {'H','e','l','l','o','\0'}; //declare with null
    - char str[] = "Hello"; //needn't null
  - Using char pointer
    - char *str = "Hello";

Example: string length


```
// Function to count the number of characters in a
string

#include <stdio.h>
int stringLength (char string[])
{
int count = 0;
while ( string[count] != '\0' )
{
++count;
}
return count;
}
int main()
{
 char str[] = {'H','e','l','l','o','\0'};
 printf ("%d", stringLength (str));
 return 0;
}
```

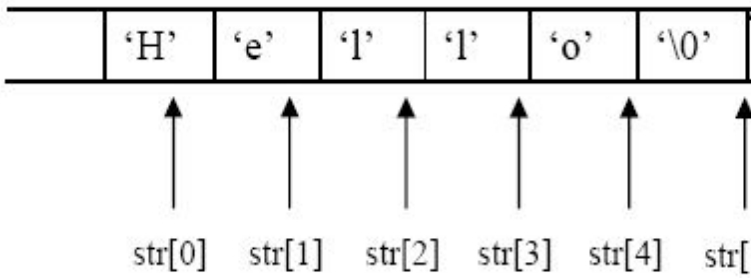
Example: string length


```
// Function to count the number of characters in a
string

#include <stdio.h>
int stringLength (char string[])
{
int count = 0;
while ( string[count] != '\0' )
++count;  return count;
}
int main()
{
 char str[] = {'H','e','l','l','o','\0'};
 printf ("%d", stringLength (str));
 return 0;
}
```



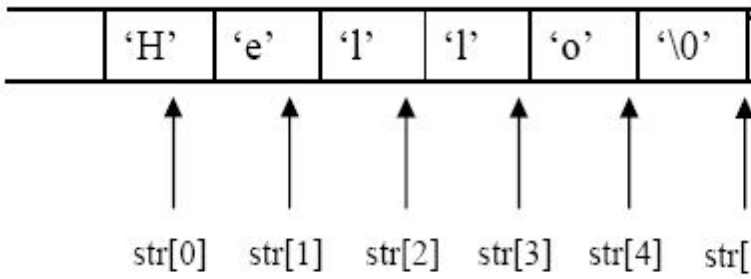
Example: string length


```
// Function to count the number of characters in a
string

#include <stdio.h>
int stringLength (char string[])
{
int count = 0;
while ( string[0] != '\0' )
count=0; return count;
}
int main()
{
 char str[] = {'H','e','l','l','o','\0'};
 printf ("%d", stringLength (str));
 return 0;
}
```



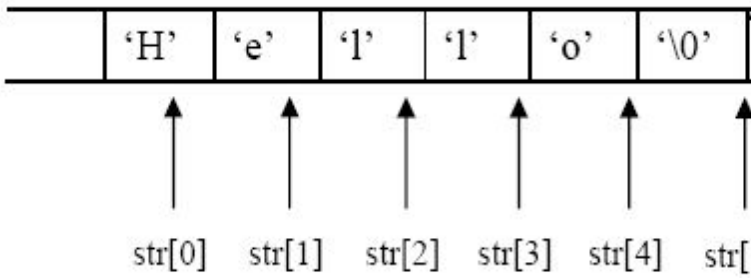
Example: string length


```
// Function to count the number of characters in a
string

#include <stdio.h>
int stringLength (char string[])
{
    int count = 0;
    while ( string[0] H != '\0' )
        count=0;
    return count;
}
int main()
{
 char str[] = {'H','e','l','l','o','\0'};
 printf ("%d", stringLength (str));
 return 0;
}
```



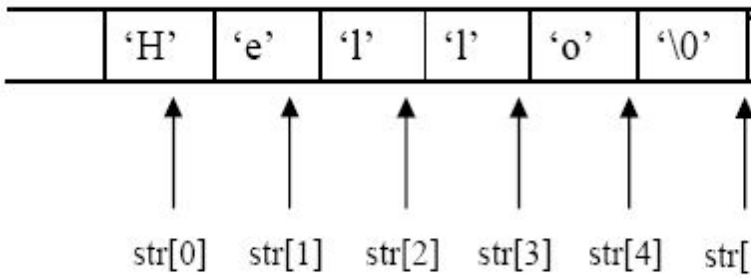
Example: string length


```
// Function to count the number of characters in a
string

#include <stdio.h>
int stringLength (char string[])
{
    int count = 0;
    while ( string[0] H != '\0' )
        count=1;
    return count;
}
int main()
{
 char str[] = {'H','e','l','l','o','\0'};
 printf ("%d", stringLength (str));
 return 0;
}
```



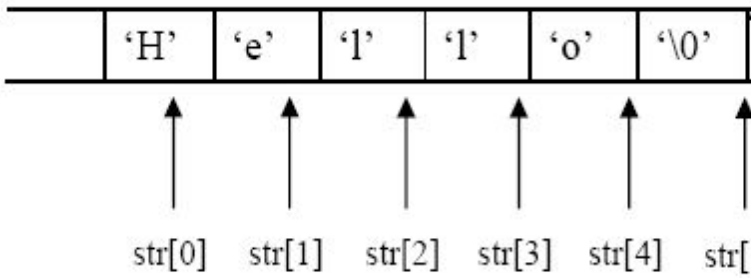
Example: string length


```
// Function to count the number of characters in a
string

#include <stdio.h>
int stringLength (char string[])
{
int count = 0;
while ( string[1] != '\0' )
count=1; return count;
}
int main()
{
 char str[] = {'H','e','l','l','o','\0'};
 printf ("%d", stringLength (str));
 return 0;
}
```



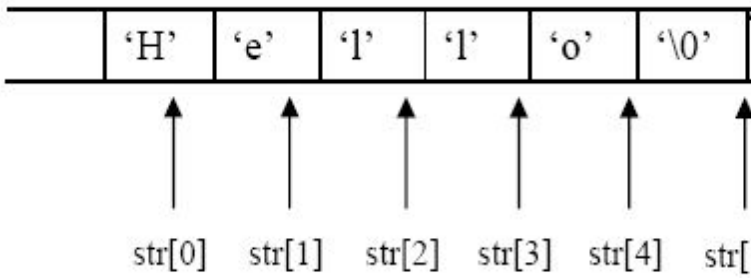
Example: string length


```
// Function to count the number of characters in a
string

#include <stdio.h>
int stringLength (char string[])
{
    int count = 0;
    while ( string[1] e!= '\0' )
        count=1;
    return count;
}
int main()
{
 char str[] = {'H','e','l','l','o','\0'};
 printf ("%d", stringLength (str));
 return 0;
}
```



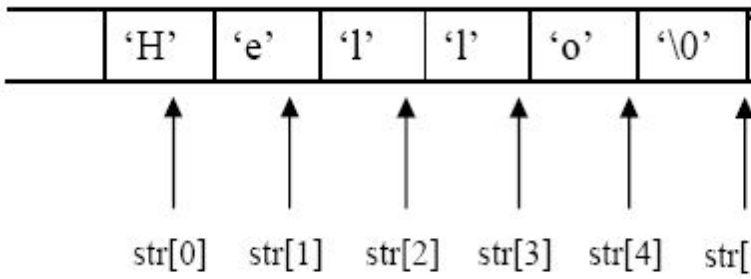
Example: string length


```
// Function to count the number of characters in a
string

#include <stdio.h>
int stringLength (char string[])
{
    int count = 0;
    while ( string[1] e!= '\0' )
        count=2;
    return count;
}
int main()
{
 char str[] = {'H','e','l','l','o','\0'};
 printf ("%d", stringLength (str));
 return 0;
}
```



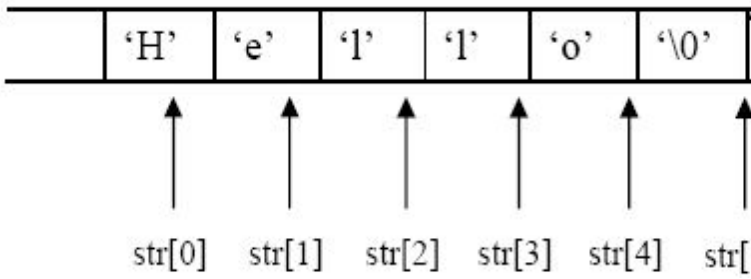
Example: string length


```
// Function to count the number of characters in a
string

#include <stdio.h>
int stringLength (char string[])
{
int count = 0;
while ( string[2] != '\0' )
count=2; return count;
}
int main()
{
 char str[] = {'H','e','l','l','o','\0'};
 printf ("%d", stringLength (str));
 return 0;
}
```



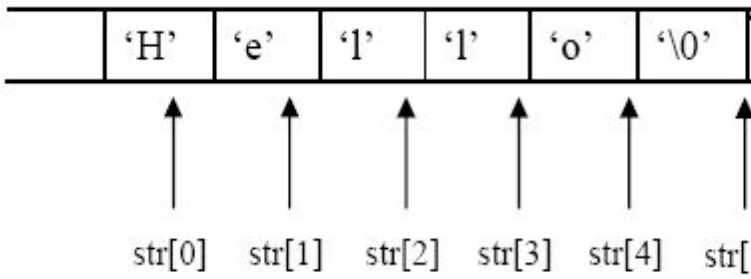
Example: string length


```
// Function to count the number of characters in a
string

#include <stdio.h>
int stringLength (char string[])
{
    int count = 0;
    while ( string[2] l != '\0' )
        count=2;
    return count;
}
int main()
{
 char str[] = {'H','e','l','l','o','\0'};
 printf ("%d", stringLength (str));
 return 0;
}
```



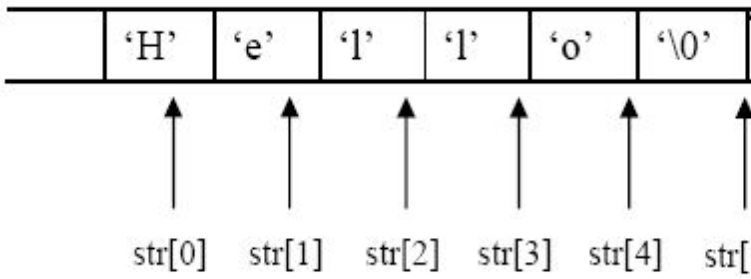
Example: string length


```
// Function to count the number of characters in a
string

#include <stdio.h>
int stringLength (char string[])
{
    int count = 0;
    while ( string[2] l != '\0' )
        count=3;
    return count;
}
int main()
{
 char str[] = {'H','e','l','l','o','\0'};
 printf ("%d", stringLength (str));
 return 0;
}
```



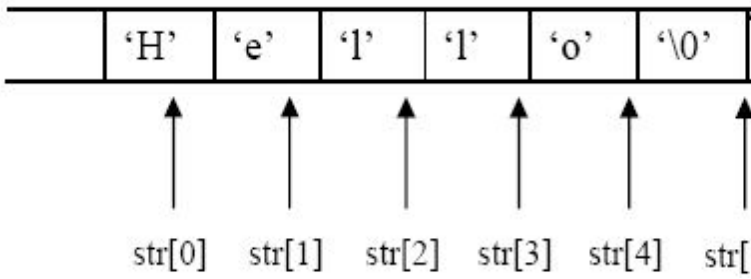
Example: string length


```
// Function to count the number of characters in a
string

#include <stdio.h>
int stringLength (char string[])
{
int count = 0;
while ( string[3] != '\0' )
count=3; return count;
}
int main()
{
 char str[] = {'H','e','l','l','o','\0'};
 printf ("%d", stringLength (str));
 return 0;
}
```



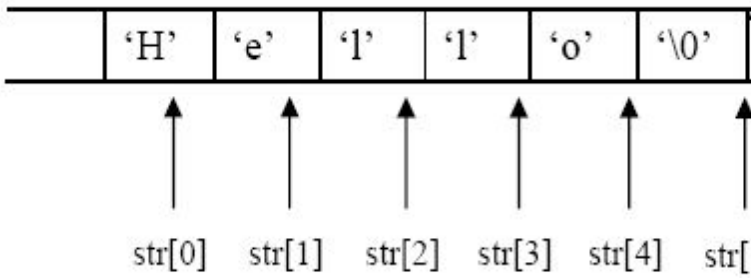
Example: string length


```
// Function to count the number of characters in a
string

#include <stdio.h>
int stringLength (char string[])
{
    int count = 0;
    while ( string[3] l != '\0' )
        count=3;
    return count;
}
int main()
{
 char str[] = {'H','e','l','l','o','\0'};
 printf ("%d", stringLength (str));
 return 0;
}
```



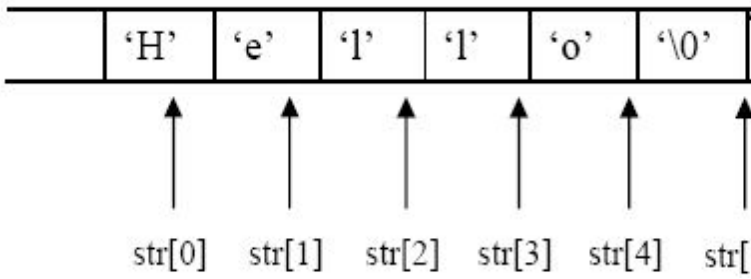
Example: string length


```
// Function to count the number of characters in a
string

#include <stdio.h>
int stringLength (char string[])
{
    int count = 0;
    while ( string[3] l != '\0' )
        count=4;
    return count;
}
int main()
{
 char str[] = {'H','e','l','l','o','\0'};
 printf ("%d", stringLength (str));
 return 0;
}
```



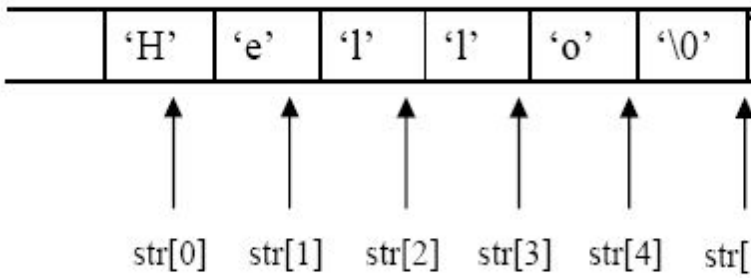
Example: string length


```
// Function to count the number of characters in a
string

#include <stdio.h>
int stringLength (char string[])
{
int count = 0;
while ( string[4] != '\0' )
count=4; return count;
}
int main()
{
 char str[] = {'H','e','l','l','o','\0'};
 printf ("%d", stringLength (str));
 return 0;
}
```



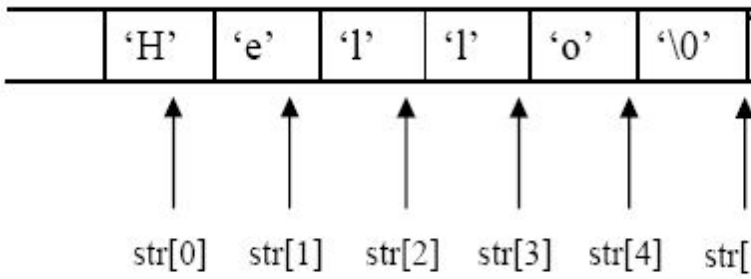
Example: string length


```
// Function to count the number of characters in a
string

#include <stdio.h>
int stringLength (char string[])
{
    int count = 0;
    while ( string[4] o!= '\0' )
        count=4;
    return count;
}
int main()
{
 char str[] = {'H','e','l','l','o','\0'};
 printf ("%d", stringLength (str));
 return 0;
}
```



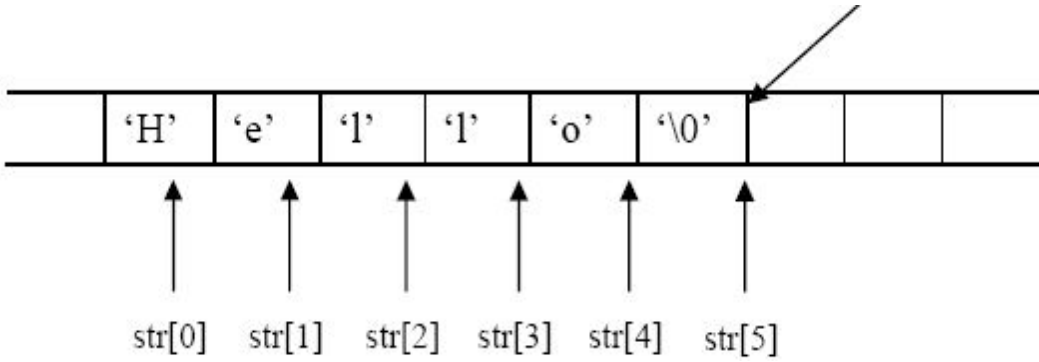
Example: string length


```
// Function to count the number of characters in a
string

#include <stdio.h>
int stringLength (char string[])
{
    int count = 0;
    while ( string[4] o!= '\0' )
        count=5;
    return count;
}
int main()
{
 char str[] = {'H','e','l','l','o','\0'};
 printf ("%d", stringLength (str));
 return 0;
}
```



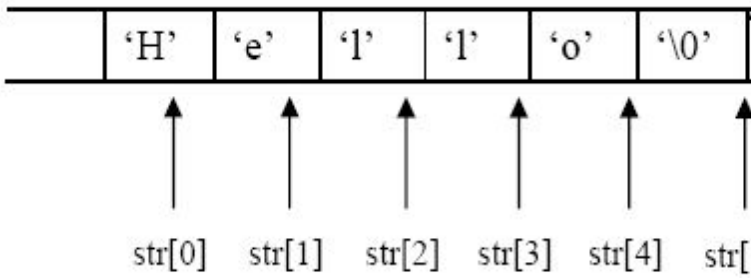
Example: string length


```
// Function to count the number of characters in a
string

#include <stdio.h>
int stringLength (char str
{
int count = 0;
while ( string[5] != '\0' )
count=5; return count;
}
int main()
{
 char str[] = {'H','e','l','l','o','\0'};
 printf ("%d", stringLength (str));
 return 0;
}
```



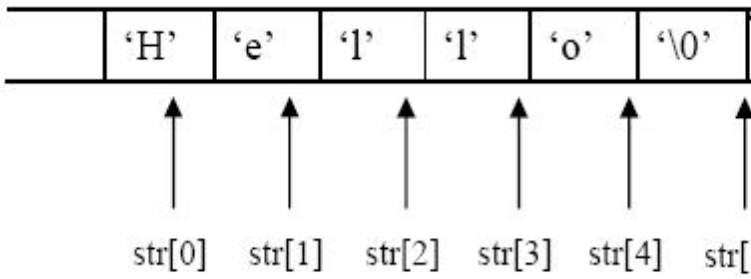
Example: string length


```
// Function to count the number of characters in a
string

#include <stdio.h>
int stringLength (char string[])
{
    int count = 0;
    while ( string[5] \0!= '\0' )
        count=5;
    return count;
}
int main()
{
 char str[] = {'H','e','l','l','o','\0'};
 printf ("%d", stringLength (str));
 return 0;
}
```



Example: string length


```
// Function to count the number of characters in a
string

#include <stdio.h>
int stringLength (char string[])
{
    int count = 0;
    while ( string[5] \0!= '\0' )
        count=5;
    return count;
}
int main()
{
 char str[] = {'H','e','l','l','o','\0'};
 printf ("%d", stringLength (str));
 return 0;
}
```



Output=5

Copying strings (Without pointer)


```
void copyString(char dest[], char srs[])
{
 int i;
 i=0;
 while ((srs[i]) != '\0')
{
dest[i]=srs[i];
 i++;
}
}
```

Copying strings (With pointer)


```
void copyString(char *out, char *in)
{
 int i;
 i=0;
 while (i!=5)
{
*out++=*in++;
 i++;
}
}
```

Copying strings (With pointer)


```
#include <stdio.h>
void copyString(char *out, char *in)
{
 int i;
 i=0;
 while (i!=5 )
{
*out++=*in++;
 i++;
}
}
int main()
{
  char str[] = {'H','e','l','l','o','\0'};
  char dest[5];
  char *in, *out;
  int i;
  in=&str[0];
  out=&dest[0];
  copyString (out, in);
```

Reversing strings (Without pointer)


```
void RevString(char dest[], char srs[])
{
 int i;
 i=0;
 while ((srs[i]) != '\0')
{
dest[l-i-1]=srs[i];
 i++;
}
}

Hello
olleh
```

Reversing strings (With pointer)


```
#include <stdio.h>
void copyString(char *out, char *in)
{
 int i;
 i=0;
 while (i!=5 )
{
*out--=*in++;
 i++;
}
}
int main()
{
  char str[] = {'H','e','l','l','o','\0'};
  char dest[5];
  char *in, *out;
  int i;
  in=&str[0];
  out=&dest[1-1];
```

Comparing strings (Without pointer)


```
#include <stdio.h>
int stringLength (char string[])
{
int count = 0;
while (string[count] != '\0' )
{ count++;}
return count;
}
void ComString(char dest[], char srs[],int l1, int l2)
{
 int i=0;
 if(l1==l2)
 { for (i=0;i<l1;i++) {
                        if(dest[i]==srs[i])
                                  {
                                  }
                        else      { break;
                                  }
                       }
 }
 if(i==l1)
    printf("The strings are same");
 else
    printf("The strings are not same");
}
```

Comparing strings (Without pointer)


```
int main()
{
  char srs[10];
  char dest[10];
  int l1,l2,i;

  printf("Enter the first string");
  scanf("%s",&srs);

  printf("Enter the second string");
  scanf("%s",&dest);

  l1=stringLength (srs);
  l2=stringLength (dest);

  printf ("\n First length=%d\n",l1 );
  printf ("\n Second length=%d\n",l2 );

 ComString(dest,srs,l1,l2);

 return 0;
}
```

Comparing strings (With pointer)


```
#include <stdio.h>
int stringLength (char *string)
{
int count = 0;
while (*string != '\0' )
{ count++;
  string++;
}
return count;
}
void ComString (char *d1,  char *s1,int l1, int l2)
{
 int i=0;
  if(l1==l2)
{
 for (i=0;i<l1;i++) {
                        if(*d1==*s1)
                                  {   d1++;
                                      s1++;
                                  }
                        else      { break;
                                  }
                       }
}
if(i==l1)
printf("Strings are same");
else
Printf("Strings are not same");
}
```

Comparing strings (With pointer)


```
int main()
{
  char srs[10], *s1;
  char dest[10], *d1;
  int l1,l2,i;

  printf("Enter the first string");
  scanf("%s",&srs);
  printf("Enter the second string");
  scanf("%s",&dest);

  s1=&srs[0];
  d1=&dest[0];


  l1=stringLength (srs);
  l2=stringLength (dest);

  printf ("\n First String length=%d\n",l1 );
  printf ("\n Second String length=%d\n",l2 );
  ComString (d1,s1,l1,l2);
 return 0;
}
```

Palindrome (Without pointer)


```
#include <stdio.h>
int stringLength (char string[])
{
int count = 0;
while (string[count] != '\0' )
{ count++;}
return count;
}
void PanlinString (char dest[], char srs[],int l1)
{
 int i=0;
 for (i=0;i<l1;i++) {
                        if(dest[i]==srs[i])
                                  {
                                  }
                        else      { break;
                                  }
                       }

 if(i==l1)
    printf("The string is palindrome");
 else
    printf("The string is not palindrome");
}
```

Palindrome (Without pointer)


```
int main()
{
  char srs[10];
  char dest[10];
  int l1,i;

  printf("Enter the first string");
  scanf("%s",&srs);

  l1=stringLength (srs);

  printf ("\n First length=%d\n",l1 );
  for(i=0;i<l1;i++)
{
dest[l1-1-i]=srs[i];
}
 PanlinString (dest,srs,l1);

 return 0;
}
```

Palindrome (With pointer)


```
#include <stdio.h>
int stringLength (char *string)
{
int count = 0;
while (*string != '\0' )
{ count++;
  string++;
}
return count;
}
void PanlinString(char *d1,  char *s1,int l1)
{
 int i=0;
 for(i=0;i<l1;i++)
 {
     *d1=*s1;
     d1--;
     s1++;
 }
d1++;
for(i=0;i<l1;i++)
{     s1--;
}
 for (i=0;i<l1;i++) {
                        if(*d1==*s1)
                                  {   d1++;
                                      s1++; }
                        else      { break;
                                  }
```

Palindrome (With pointer)


```
if(i==l1)
    printf("The string is palindrome");
 else
    printf("The string is not palindrome");
}
int main()
{
  char srs[10], *s1;
  char dest[10], *d1;
  int l1,l2,i;

  printf("Enter the string");
  scanf("%s",&srs);
  s1=&srs[0];

  l1=stringLength (s1);
d1=&dest[l1-1];
  printf ("\n String length=%d\n",l1 );

 PanlinString(d1,s1,l1);

 return 0;
}
```

Substring (Without pointer)


```
#include <stdio.h>
int stringLength (char string[])
{
int count = 0;
while (string[count] != '\0' )
{ count++;}
return count;
}
void SubString (char srs[],int l2)
{
 int i=0;
 printf("The substring is");
 for (i=0;i<l2;i++) {
                    printf("%c",srs[i]);
                    }
}
```

Substring (Without pointer)


```
int main()
{
  char srs[10];
  int l1,l2,i;

  printf("Enter the string");
  scanf("%s",&srs);

  l1=stringLength (srs);

  printf ("\n First length=%d\n",l1 );
  printf("\n Enter the length");
  scanf("%d",&l2);
  SubString (srs,l2);

 return 0;
}
```

Substring (With pointer)


```
#include <stdio.h>
int stringLength (char *string)
{
int count = 0;
while (*string != '\0' )
{ count++;
  string++;
}
return count;
}
void SubString (char *s1,int l2)
{
 int i=0;
 printf("The substring is");
 for (i=0;i<l2;i++) {
                    printf("%c",*s1);
                    s1++;
                    }
}
```

Substring (With pointer)


```
int main()
{
  char srs[10],*s1;
  int l1,l2,i;

  printf("Enter the string");
  scanf("%s",&srs);
  s1=&srs[0];
  l1=stringLength (s1);

  printf ("\n First length=%d\n",l1 );
  printf("\n Enter the length");
  scanf("%d",&l2);
  SubString (s1,l2);

 return 0;
}
```

Unions

- Similar to **structures**.
- The syntax to declare/define a union is also similar to that of a structure.
- The only differences is in terms of storage.
- In **structure** each member has its **own storage** location, whereas **all members** of **union** uses a **single shared memory** location which is equal to the size of its **largest data member**.

```
struct st
{
int x;
 int y;
    }s1;
s1.x=10;
```

```
union un
{
    char x;
    int y;
     }u1;
u1.x=10;
```

Enumerated data type

- You can use the enumerated type to declare **symbolic names** to represent integer constants.
- By using the `enum` keyword, you can create a new "type" and specify the values it may have.
- Actually, enum constants are type int; therefore, they can be used wherever you would use an int.
- The purpose of enumerated types is to enhance the readability of a program.

```
enum primaryColor { red, yellow, blue };
enum primaryColor myColor, Color;
myColor = red;
```

Example: enum


```
#include <stdio.h>
int main ()
{
enum week{Mon, Tue, Wed, Thur, Fri, Sat, Sun};
    enum week day;
    day = Wed;
    printf("%d",day);

    return 0;
}
```