Page No.
Date

~~Microprocessor~~ Functions.

* Types of functions.

User defined — Library functions.

1. Library functions.
- It is also refered as built in functions.
- A Compiler Package That already Exits Contain's This functions. Each having Specific Meaning.
- Build in functions have advantage of being directly used without being defined.
- User defined functions must be declared before Use.

Ex. Pow(), Sqrt(), Strcmp(), Strcpy() Etc.

- C Library function are Easy To use & optimize for better Performance.
- C Library functions Save Lots of Time.
- Library functions are Convinient as They alway work.

```
#include<stdio.h>
#include<math.h>
{
  Double Number;
  Number = 49;
/* Computing Square root with The help of Predefined C lab func.
  Double Square root = Sqrt(number);
Printf("The Square root of %.2lf = %.2lf", Number,
          Square root);
  return 0;
}.
```

Output - The Square root 49.00 = 7.00

2. User defined functions.

- The functions That Programmer Creates are Called User defined functions.
- User defined functions Can be Improved & Modified according To Need of Programmer.
- Whenever we write a function That is Case-Specific & not defined in any header file, we need To Declare & define our own function According To Syntax.

* Advantages.

- Functions Can be Modified as Per Need.
- The Code of This function Can be reusable in other Programmes.
- This functions are Easy To Understand, Debug & Maintain

```
#include<stdio.h>
#include<conio.h>
int Sum(int a, int b);  // Declaration
int main()
int x;
x = Sum(5,6);           // Calling
Printf("Sum is %d", &x);
}
 int Sum(int a, int b)  // Defination.
{
   return a+b;
}
```

Page No.
Date

* Syntax of functions in C.
- The Syntax of function is classified into 3 Types.
1. Function Declaration 2. function Defination
3. function Calling.
" A function is a Block of Code That Performs a Specific Task, It Can Take Arguments Perform Calculations & written Values".

1. function Declaration
- Function Declaration (Also Known As function-Prototype) Consist of 4 Types.
- In function Declaration, We Must Provide 1. Function name, 2. Return Type, 3. Number of Parameters & There Type (Parameter list)., 4. Semicolon.
- A function Declaration Tells The Compiler That There is a function with The given Name, Defined Somewhere Else in Programme.
- Syntax:
{Return-type function_name (Parameter list);}
Ex. Int Swap (Int a, Int b);
Return Type function name Parameter list Semicolon.
Type-Int Name - a, b.
- The Parameter Name is Not Mandatory while Declaring functions.
- The we can also declare functions without using name of data Variables.
Ex. Int Swap (Int, Int); // No Name mentioned.
function Declaration with, without Parameter Name.

Note: A function in C Must always be Delard Globally before Calling.

* Various Types Of Declaration.

1> Int Swap (Int a, Int b);

" We have declared a function whose Name is Swap, It will work on Two Integer Values, or Parameter & it'll return It will return a Integer Type Value.

2> Void Swap (Int a, Int b);

" Function Name is Swap, It will work on 2 Parameters Both are Integer Type, It will Not return any Value.

3> Void Swap ( );

" Function Name is Swap, No Value will be Passed To It & No Value will be return.

4> Void Swap (Int, float);

" Function Name is Swap It will work on Two Parameters one is Int & another float, it will not return Any Value".

Note: If you want return output or Value Then declare Data Type Int Swap (int a, int b); If Not Then Use Void.

- Its Also Categorized in 4 Types.
- We will se It Later.

* Function Defination

- A function Defination is also called function Implementation.
- This Defination Section Tells The Compiler what operation have To be Perform on The given Values.
- The Defination Section Consist of Actual Statements or operation, Conditions, which will Executed when function will be Called.
- A C-Function is generally Defined & Declared in a Single Step, Because function Defination Always Start with Declaration, So we do not need To Declare It Explicitly.

Syntax

```
function_type function_name (Parameter Type Parameter Nam
{
  Local Variable Declaration;
  Executable Statements;
  :
  return Statement;
}
```

// If you have given function Type Then only use return Statement, If Void is Use Then No Need.

Ex. Void ~~Fun~~ Swap (int x, int y)

```
{
    int Temp;
    Temp = x;
    x = y;
    y = Temp;
}.
```

* Function Calling

- Function Calling is a statement that Instructs The Compiler To Execute function.
- We use The function Names & Parameters in function Call.

Syntax: function_name(Parameter list);

```
Swap(a,b);   // with Parameter.
Swap();      // without Parameter.
```

Ex. WorkKing of function in C.

```
#include<stdio.h>
Void ~~int~~ Swap(int a, int b);   // Declaration
int main()
{
  int ~~x~~ = 1, ~~y~~ = 6;
  Swap(x, y);                     // Call
  Printf("Befor x=%d, y=%d");
}
Void Swap(int a, int b)           // Define
{
  int C;
  C = a;
  a = b;
  b = C;
  Printf("output a=%d, b=%d", a, b);
}
return 0;
}.
```

Here, It Simply Calls The function. & Then Values are passed To function.

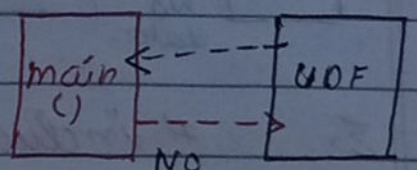
- After Calling operation got Perform & function gets Terminated.

Page No.
Date

* On The Basis Of Function Proto Type or Function Declaration Functions Are Categorized in 4 Types.

1) Function with No Return Value & No Arguments.

- No Data Communication.
- It will No give Any Value To Main function.
- It will Not Take Value from main function.



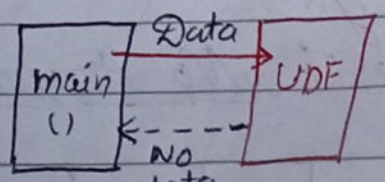
Ex. #include<stdio.h>
#include<conio.h>
Void Sum(); // Declaration
Void main()
{
Sum(); // Calling
}
Void Sum() // Define
{
int a=5, b=3;
Printf("\n Sum of a & b is %d", a+b);
}
Getch();
}.

After Calling function gradually Moves To User Defind functions.

output: 5 + 3 = 8.

(2) Function with Arguments & No return Value.

- one way Communication



"Main function is giving Value to UDF, But UDF is Not returning Value."

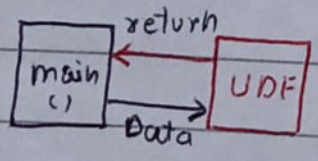
Ex.
```
#include<stdio.h>
#include<conio.h>
{ int a,  Void Sum(int, int); // Declaration
Void main();
{ int a = 8, b = 3;
  Sum Sum(a, b); // calling
}
Void Sum(int x, int y) // Defi.
{
  Printf("\n output is %d", x + y);
}
Getch(); }
```

There Values are getting printed here.

operation Performed, & output Printed.

output: 8 + 3 = 11.

<3> Function with Arguments & One return Value.



- Two way Communication.
- Both returns Values.

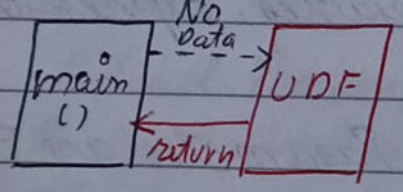
Ex.
```
#
#
Int Sum Sum(int, int); // Declaration
Void main()
{
  int a= 8, b=5, x;
  x = Sum(a, b); // call
} Printf("output is %d", x);
}
```

```
Int Sum(int x, int y) // Defi.
{
  return x + y;
}
```

output is : 13.

- Here 1st function is called, It Directly Got Moved To Defination Section, It will Complete Its Task & return The Value To main function That is x."

(4) Function with No Arguments But return Value.



- No Data will be given But UDF will return Value.

```
#
#
Int Void Multi( ); // Declaration
  Void main();
{ int x;
  x = multi(); // calling
  Printf("\n multi = %d", x);
}
Int Void multi(); // Defi.
{
  int a = 5, b = 8;
  return a x b;
}
```

output.

x = 40.

- In This Main function is Not Providing any Value To UDF, Main() is Just Calling It & Taking output from It.
- function Called It moved To UDF, Operation Perform output returned To main() That is x.

Page No.
Date

* Parameter In C.

- Actual Parameters & formal Parameters.
- In C Programming Parameters are Variables That are used To pass The Values or reference between The function.
- There are Two Types of Parameters as Mentioned Above.

1. Actual Parameters.

- Actual Parameters are Those Values That are Passed To a function when Called. They are also known as Arguments These Values can be Constant, Variables, Expressions or Even other function Calls.
- When function is Called Actual Parametes are Evaluate & There Values are Copied To formal Parameter of The Called function.
- Actual Parameters are Closed Into Parenthesis & Seprated by ~~Colounm~~ Comas, Terminated by Semicolon.

Ex. Int result = Swap (5, 6);

- Here 5, 6 are actual Parameters & Passed To function Name Swap.

2. Formal Parameters.

- Formal Parameters are The ~~Values~~ Variables Declard in The header, That are use To receive values of Actual Parameter. Through function Calls.
- They are also known The function Parameters.
- formal Parameters are use To define function Sign & To Specify The Type & Number of Arguments, That a function Can accept.

- In C formal Parameters are declared in function Declaration or Defination.

Ex. Int Swap (int a, int b)

- In This a & b are formal Parameters.

* | Actual Parameters | formal Parameters.

- The Values That are Passed To function is Called Actual Parameter.
- It is use To receive Value of Actual Parameter
- Actual Parameters Can be Expression, Values or other function Calls.
- formal Parameters are always Variables.
- Actual Parameter Can Evaluate Value at run Time.
- formal Parameter are Defined at Compile Time.

* Importance of Actual & Formal Parameters.

- The Parameter Enable The Passing of Values & reference between functions.
- Function Can be designed To Accept ~~Parameter~~ Values That are passed by Parameters, reference, Value or pointer.
- It Enables The Creation of reusable & flexible Code, That Can be Used In Different Context.

* Call by Value in C.

- It means function recieves the copies of values of Actual Parameter.
- It allows the function to modify the copy without affecting the Original Value of Actual Parameter.

Ex. a = 5, b = 6;

Swap (a, b); ← Here the values are directly getting copied here.

int swap (int x, int y);

- There are two copies of values one in Actual Parameter (Original copy) & formal parameter (function copy).

Ex.

```
# include <stdio.h>
Void Swap (int x, int y);
int main ()
{  int a=5, b=6;
   Swap (a, b); // Actual Parameter.
   Printf (" In The Caller : \n a = %d, b = %d", a, b);
   return 0;
}
Void Swap (int x, int y) // Formal Parameter.
{  int c;
   c = a;
   a = b;
   b = c;
   Printf (" Inside function x = %d, y = %d", x, y);
}.
```

- output :

In The Caller : a = 5, b = 6

Inside function a = 6, b = 5.

Exp: Int we delare a = 5, b = 6 Then called function Programe move to function, The values of Actual Parameters got copied to Formal Parameters & further Process happen.

Page No.
Date

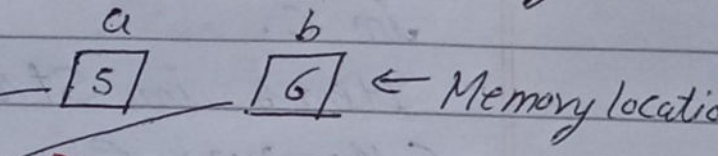
* Call by reference
- Means function recieves a reference To The Memory Location of Actual Parameter.
- It allow function To Modify The Value of Actual Parameter Directly.
- In C we use pointer (*) To Achieve Call by reference.

Ex. a = 5, b = 6.

Swap (&a, &b);

Void Swap (Int *a, Int *b)

here They are not receiving values but address.



- They are directly using This Values from Memory Location.
- Whenever Programme needs Values It will Take It from here.

- "Both Actual & Formal Parameter refers To Same Location".
- Any change Made Inside function are actually reflected in Actual Prameter of The Caller.

Ex.
```
# include <stdio.h>
Void Swap (Int *x, Int *y);
int main ()
{
  int a = 5, b = 6;
  Swap (&a, &b);
  Printf("The Caller : \n a = %d, b = %d", a, b);
  return 0;
}
Void Swap (Int *x, Int *y)
{
  int C;
  C = *a;
  *a = *b
  *b = C;
  Printf(" Function : \n x = %d, y = %d",
        x, y);
```

output. Caller a = 5 b = 6
x = 6 y = 5

C - Structure

- "The Structure in C is a user-Defined Data-Ty That Can be use To group Data-Types or Items of Different Types Into a Single Type."
- The struct Keyword is used To define Structu in C.
- Items in Structure are Called Members. (Members Can be of any Type. Int, Char, float).
- The Value of Structure are stored in Contiguous Memory Location.

(Introduction).

- In real world we have To Deal with Different Type of Data like Numbers, characters or factorial Values.

For Ex. - A System which have To Maintain Student Info his Name, roll no., Marks, PRN Etc as we Can See This all Information have different Type of Data like Name has characters, roll.no int, Marks float Etc... So managing such a diverse Data Can be a Challenging Task.

- So, To avoid This Kind of Problem we have Structures In C, Structure Can Contains No. of Data Types grouped Together.
- This Data Types are May or May not be Same. Syntax

Page No.
Date

C - Structure Declaration

```
Struct  ← Struct Student → Structure Name or Tag
Keyword {
            Char name[35];
            int id;          } Members or fields of
            float Marks        Structure.
        };
```

- In Structure declaration, we specifies Its Members Variables along with Their Data-Types.
- We use Struct Keyword To Declare The Structure.
- The above Syntax is Called a Structure Template or Structure Prototype.
- No Memory is allocated To Structre in Declaration.

C - Structure Defination.

- To use This Structure in our ~~otion~~ Programme we have To Define Its ~~Fonts~~ Instance, object or Variable.
- We can do it By creating Variable of Struct Type.
- It can be define In 2 Methods.

1. Structure Variable Declaration with Structure Template.
2. Structure Variable Declaration after Structure Template.

1. With Structure Template.

Syntax:

```
Struct Struct_name {
    Data_Type member_name1;
    Data_Type member_name1;
    =
} Variable 1, Variable 2, ....
```

```
Ex. Struct Student {
        int Roll no;
        Char name[35];
    } S1, S2, S3, S4;
```

We can Take as much Variable as required by us To repeat structure in Programme.

2. After Structure Template.

Syntax. // Structure is declared before.
This is in main().
Struct structure_name variable1;
// use STRUCT Key word Inside Main(), followed by
~~Ex.~~ name of structure & The Name of Variable:

Ex.
```
Struct mystruct {
    int Num;
    char Letter;
};
int main()
{   Struct mystructure S1;
    return 0;
}
```

- In this two ways we can Define The Structure Vario
& use The Structure data Type in Programmes.

• Access Structure Member.
- We can Access Structure Member by using (.) operator
// It simply means Assigning Values To The Members of
The Structure.

Syntax:
```
Structure_name.member1;
```
or.
```
Structure_variable.member1 = Value;
```

Ex:
```
S1.Num = 13; // 13 value is assign To Num
                member of Struct ~~mystr~~ mystruct
```

Page No.
Date

• Initialization of Structure Members.

- We can Initialize Structure members in 3 Ways which are as follows.

1. Using Assignment Operator.
2. Using Initializer List.
3. Using Designated Initializer List.

1. Initialization Using Assignment Operator.

Syntax:
```
Struct Structure_name str;
str.member1 = Value1;
str.member2 = Value2;
str.member3 = Value3;
```
} In This we are Initializing Each member Seperately.

Example.
```
Struct Mystruct S1;
S1.Num = 13;
S1.letter = 'B';
```

∴ This is how Initialization is Done using Assignment Operator.

2. Initialization Using Initializer List:

"In This Type of Initialization, The Values are assigned or given To Members in The Sequence or order in which They are declare in structure Templelate."

Syntax: Struct Structure_name str = {Value1, Value2, Value3};

Ex: Struct Str1 Var1 = {1, 'A', 1.00, "Hello"};

In This way Values are Assign in The Curly Braces.

3. Initialization Using Designated Initializer List.

"Designated Initializer allows Structure Members To be Initializer In Any order."
"Simply Means using Designated Intializer we can Initialize The member of structure Indenently of Their order".

Syntax:

```
Struct Structure_name Str = { .member1 = Value1,
                              .member2 = Value2,
                              .member3 = Value3 };
```

// Here we 1st write The Struct Keyword Then The Structure Name and The Variable or object Then using Simple Assignment operator and use Curly Braces, In That Curly Braces first The Dot '.' The Member-name Then Assingment operator Then Value.
This is how Designated Initializer List is Used.

Ex.

```
Struct str2 {
  int ii;
  char cc;
  float ff;
} Var; // Variable Declaration with structure Template.
int main() {
  Struct str2 var1 = { .ii = 5, .ff = 5.00, .cc = 'h' };
printf("Struct 2 \n\t i = %d, c = %c, f = %f, var1.ii,
                                  var1.cc, var1.ff);
return 0; }.
```

Page No.
Date

Output :-
Struct 2
i = 5, c = a, f = 5.000000.

// C Programme To Illustrate The Use of Structure.

```
#include<stdio.h>
Struct Str1 {
   int i;
   char c;
   float f;
   char s[30]; };
Struct Str2 {
   int ii;
  char cc;
  float ff;
} Var; // Variable Declaration with struct. Template.
Int main()
{. // Variable declaration after struct. Template.
   // Initialization with Initializer list & designated IL:
   struct str1 var1 = {1, 'A', 1.00, "Geeks for Geeks"};
                var2;
   Struct Str2 var3 = {.cc = 'n', .ff = 5.00, .ii = 13};

var2 = var1; // Copying Structure using Assignment operator.

Printf("Struct1:\n\t i=%d, c=%c, f=%f, s=%s\n", var1.i,
                        var1.c, var1.f, var1.s);
Printf("Struct2:\n\t i=%d, c=%c, f=%f, s=%s\n", var1.i,
                        var1.c, var1.f, var1.s);
Printf("Struct3:\n\t i=%d, c=%c, f=%f\n", var3.ii,
                        var3.cc, var3.ff);
return 0; }.
```

Output

Struct 1:

i = 1, c = A, f = 1.000000, S = Geeks for Geeks

Struct 2:

i = 1, c = A, f = 1.000000, S = Geeks for Geeks

Struct 3:

i = 13, c = n, f = 5.000000

- **Nested Structures.**

- C Language allows Us To Insert One Structure into another as a member.
- This Process is called Nesting & Such Structure are Called nested Structures.
- There are 2 ways ~~to~~ in which we can nest one Structure into another.

1. Embedded Structure Nesting
2. ~~Sem~~ Seperate Structure Nesting.

	Structure	Union.
Defi.	① A structure is a user defined datatype That group different data Type into a single Entity	① A Union is a User-defined datatype That allow Storing different data Type at same Memory location.
Keyword	② 'Struct'.	② 'Union'.
Size	③ The Size is sum of all members, with Padding if necessary.	③ Size is Equal To Size of largest member, with possible padding.
	④ allocated Unique Storage Area to ~~Each item~~ of location To members.	④ Memory allocation is shared by individual Union Members.
	⑤ No data overlap members are Independent.	⑤ Full data Overlope as members share Same memory.
	⑥ Individual member can be	

'Pointers'

Pointer

- A Pointer is a derived data Type in C.
- It is build from one of the fundamental data Types in C.
- Pointers Contains the Memory address as the Value.

"Since this data address as & are the locations in Computer Memory where Program Instructions & data are stored, Pointer can be used To Acess & manipulate data stored in Memory."

"A Pointer is a Variable that stores memory address of any another Variable. Instead of holding a direct Value, It holds the address where Value is stored in Memory."

- There are 2 Important Operators that we will use in pointer Concept.
  - Deferencing Operator (*)
  - Address operator (&).

Ex.

Variable	Value	address
Var1	80	80502
Pointer → Var2	80502	80504

Type Variable.

- Advantages.

1) By using Pointer we Can Access The Data which is Available Outside The function.
2) By using pointer we can implement Dynamic Memory allocation.
3) Pointer Provides a way To return More Than one Value To a functions.
4) Pointer reduces The Storage Spac & Complexity of Programme
5) Pointers reduces The length of Programme & Its Execution Time As well.

- Accessing The Address of a Variable.

Syntax: & Variable. &a → address of a.

↳ address operator
↳ "address of"

- Declaring Pointer.

Syntax: Data Type * pointer-name ;

(Int *var ;)

(i) asterik (*) % Tells The Variable Pointer-name is pointer Variable.
(ii) Pointer-name needs a memory allocation.
(iii) Pointer-name points to Variable of Same datatype it have.

Ex: int *p // Integer Type pointer.

- Till we didnt Intialize It, It will points To unKnown Location.

Page No.
Date

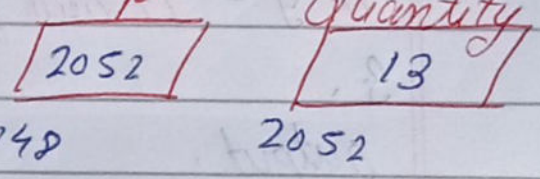
- Pointer declaration Style:

int* p;
int *p; → Most Popular.
int * p;
int*p; → Invalid X

"In Pointer declaration we use '*' asterik Symbol before Var_name."

- Initialization.

Ex. `int *P, Quantity;` → declaration.
`*P = &Quantity`
`// Initialization`



"Pointer Initialization is the process when we assign some Initial value to pointer variable.
we use the (&) address of operator to get the memory address of a Variable & Then Store It in Pointer".

Programme:

```
Void main () {
int x, y;   // Variables.
int * Ptr;  // Pointer Ptr declared.
x = 10;     // Intialize Variable x with Value 10.
Ptr = &x;   // Initialized The Pointer Ptr with address of x.
y = * Ptr;  // Storing Value of asteric '*' Ptr in y.
```

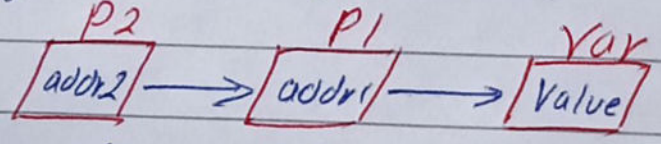
('Pointer' means "when only Ptr = it will give memory dereferencing). address of x" & when " * Ptr = It will give Value of Variable whose address is store in Ptr.

```
Printf ("Value of x is %d \n", x);
Printf ("%d is stored at address %U \n", x, &x);
                // Value of x & address of x.
```

Printf("%d is Stored at address %u \n", *&x, &x);
// Here like *ptr, *&x will also gives its Value not address
&x will give address.
Printf("%d is Stored at address %u \n", *ptr, ptr);
// *ptr - will give Value of x, ptr = address of x
Printf("%d is stored at address %u \n", ptr, &ptr);
// ptr - address of x, &ptr - It will give address of ptr itself
Printf("%d is Stored at address %u \n", y, &y);
// y - It will give Value stored in It, &y address of y.
3.
output.

- Chain of Pointers.

It is Possible To make a pointer To another pointer Thus Creating a chain of Pointer.



Here Pointer Variable P2 Contains The ~~Value~~ address of Pointer Variable P1 & P1 Contains The address of Variable.
This is Known as multiple Indirections.

Ex.
```
#include<stdio.h>
#include<conio.h>
Void main()
{
int x, *P1, **P2;
x=100;
P1=&x
P2=&P1;
Printf("x=%d", x);
Printf("P1=%d", P1);
Printf("P2=%d", P2);
```

** - To hold address of pointer So we have to Use double astric.

x = 100
P1 = -12
P2 = -14

Page No.
Date

C Pointer Arthematic.

- The Pointer Arthematic refers To the legal or valid arthematic operations That can be Perform on a Pointer.
- It is slightly different from one that we generally use for mathematical calculations, as only limited set of operation can be Performed on the Pointer.

Ex.

- Increament / Decreament by 1
- Addition / Subtraction of Integer
- Subtracting Two pointers of same Type.
- Comparing Two pointer of same Type.
- Comparing / Assigning with NULL.

```
#include<stdio.h>
#include<conio.h>
 void main() {
int *P1, *P2, a, b, x, y, Sum =0;
a=10;
b=20;
P1= &a;
P2= &b;
x = *P1 * *P2;
Sum = Sum + *P1;
y = *P1 / *P2;
*P2 = *P2 + 5;
Printf("\n x=%d", x);
Printf ("\n Sum =%d", Sum);
Printf ("\n y =%d", y);
Printf("\n *P2 =%d", *P2);
```

- output:

x = 200
Sum = 5
y = 0
*P2 = 30.

- Pointer Dereferencing.

- Dereferencing The Pointer is The Process of accessing The value Store in memory address specified in Pointer.
- Here, we use The Same Dereferencing operator (*) That we used in pointer declaration.

Exp.

```
# include<stdio.h>
  int main() {.
 int x=100, *p; int y;
   p = &x;
  // y = p - Here we give only P so as Its a pointer
```

This will not run in Programme. It will only give The memory address of x not value.

```
  y = *p - Here we use ~~def~~ derferencing with P, now
           it will give value of variable x not
           address.
           // This is ~~access~~ accessing value Store in
              memory address specified in Pointer.
 Printf("y = %d", y);
  return 0;
 }.
```

Page No.
Date

String functions.

~~Syntax :-~~ ① Strcat() - Concatenates Two strings.

Syntax - Strcat(string1, string2)

- This function is used To Merge Values of Two Existing strings.
- Ex String2 values is going merge with String1 in Syntax.

Ex.
```
#include<stdio.h>
#include<conio.h>
#include<string.h>
Void main()
{
char string1[] = "Welcome";
char string2[] = "To Elg";
Strcat(string1, string2);
puts(string1);
Getch();
}.
```

output:
Welcome To Elg.

② Strcmp() - Compare Two Strings.

Syntax:- Strcmp(String1, String2);

- If Value < 0 Than It indicates it that String1 is less than String2.
- If Value > 0 Than It indicates that String is Greater Than String2.
- If Value = 0 Than It indicates that String is Equal To String2.

Ex.- Strcmp (String1, String2);
(Both are Values)
- Strcmp (name1, "Sandeep");
one Variable one string.
- Strcmp ("Sushil", "Tony");
Both String.

```
#include<string.h>
#include<conio.h>
#include<stdio.h>.
Void main()
{
Char name1[30], name2[30];
Printf("Enter 1st String :\n");
Gets(name1);
Printf("Enter 2nd String :\n");
if(strcmp(name1, name2)==0)
{
   Printf("\n Both names are Equal");
}
Else {
   Printf("\n Names are Unequal);
}.
Getch();
}.
```


output:
Enter 1st String: sushil
Enter 2nd String: SUSHIL
Names are Unequal.

③ Strcpy() - Copies one String over another.

Syntax: Strcpy (String1, String2);

destination — Sourse string

String2 copies in string1.

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main()
{
char city1[40]="Delhi";
char city2[40]="Bhopal";
Printf("\n city1 = %s", city1);
Strcpy(city1, city2);
Printf("\n After changes");
Printf("\n city1 = %s", city1);
Getch();
}
```

output:
- Delhi
- After changes
- Bhopal.

④ Strlen() = finds the length of a String.

Syntax :- Count = Strlen(string);

└ It will returns us a Integer Value.

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
Void main()
{
 Char String[50];
 Printf("\n Enter String");
 Gets(String);
 Printf("\n length of String = %d", Strlen(String);
Getch(); }.
```

output:
Sushil
length of string = 6.

- Storage Classes.

C-Programming, Variable behave differently based on where & how they ~~data~~ are Declared. This are determine by Storage classes In C, which define a Variable Scope, lifetime, Visibility & Memory Location.

Understanding Storage classes helps in writing Efficient and optimized programmes. There are four Types of Storage classes in C language: Auto, resistor, Static & Extern, Each Serving a specific purpose in Memory Management & data accessibility.

- Types of Storage classes in C
- 4 Types.
- Auto for Temporary local Variable.
- resistor for high speed Variables Stored in CPU resistors.
- Static - for Persistent local or global Variable.
- Extern - for sharing global Variables across multiple files.

Auto Storage classes in C
- The auto Storage classes in C is default storage classes for local Variables.
- Scope (local To function).
- Life Time - Created when function is called, destroyed when function Exits.
- Memory location : RAM.

Page No.
Date

Ex. of auto storage classes.

```
# include <Stdio.h>
Void Examplefunction ()
{
 auta int num = 10; // 'auto' Key word is optional.
Printf("Value of num :/. d\n", num);
}
int main()
{
  Example function ();
  retuen 0;
}.
```

Register Storage Classes.