C LANGUAGE

Overview of C

- C is developed by Dennis Ritchie
- C is a structured programming language
- C supports functions that enables easy maintainability of code, by breaking large file into smaller modules
- Comments in C provides easy readability
- C is a powerful language

PROGRAM STRUCTURE

A sample C Program

```
#include<stdio.h>
int main()
{
   --other statements
   // Comments after double slash
}
```

HEADER FILES

- The files that are specified in the include section is called as header file
- These are precompiled files that has some functions defined in them
- We can call those functions in our program by supplying parameters
- Header file is given an extension .h
- C Source file is given an extension .c

MAIN FUNCTION

- This is the entry point of a program
- When a file is executed, the start point is the main function
- From main function the flow goes as per the programmers choice.
- There may or may not be other functions written by user in a program
- Main function is compulsory for any c program

WRITING THE FIRST PROGRAM

```
#include<stdio.h>
int main()
{
   printf(“Hello”);
   return 0;
}
```

- This program prints Hello on the screen when we execute it

RUNNING C PROGRAM

- Type a program
- Save it
- Compile the program – This will generate an exe file (executable)
- Run the program (Actually the exe created out of compilation will run and not the .c file)
- In different compiler we have different option for compiling and running. We give only the concepts.

COMMENTS IN C

- Single line comment
  - // (double slash)
  - Termination of comment is by pressing enter key
- Multi line comment

  /*….

  ……*/

  This can span over to multiple lines

DATA TYPES IN C

- Primitive data types
  - int, float, double, char
- Aggregate Data Types
  - Arrays come under this category
  - Arrays can contain collection of int or float or char or double data
- User defined data types
  - Structures and enum fall under this category.

VARIABLES

- Variables are data that will keep on changing
- Declaration

  <<Data type>> <<variable name>>;

  int a;
- Definition

  <<varname>>=<<value>>;

  a=10;
- Usage

  <<varname>>

  a=a+1; //increments the value of a by 1

VARIABLE NAMES- RULES

- Can contain letters, numbers or underscore.
- Should start with a letter or an underscore (_)
- Reserved words such as int can not be used as names
- Names cannot contain whitespaces or special characters like !, #, %, etc.
- Variable names are case sensitive
  - A and a are different.

INPUT AND OUTPUT

- Input
  - scanf(“%d”,&a);
  - Gets an integer value from the user and stores it under the name “a”
- Output
  - printf(“%d”,a);
  - Prints the value present in variable a on the screen

FOR LOOPS

- The syntax of for loop is

```
for(initialization; condition_checking; increment)
{
   set of statements
}
```

Eg: Program to print Hello 10 times

```
for(i=0;i<10;i++)
{
   printf("Hello");
}
```

WHILE LOOP

- The syntax for while loop

```
while(condition)
{
        statements;
}
```

Eg:

```
a=10;
while(a != 0)
{
        printf("%d",a);
        a--;
}
```

Output: 10987654321

DO WHILE LOOP

- The syntax of do while loop

```
do
{
    set of statements
}while(condn);
```

Eg:

```
i=10;
do
{
    printf("%d",i);
    i--;
}while(i!=0)
```

Output: 10987654321

CONDITIONAL STATEMENTS

```
•IF (CONDITION)
{
 STMT 1;                    //Executes if condition is true
 }
else
 {
  STMT 2;                   //Executes if condition is false
 }
```

CONDITIONAL STATEMENT

```
switch(var)
{
case 1:          //if var=1 this case executes
                 stmt;
                 break;
case 2:          //if var=2 this case executes
                 stmt;
                 break;
default:         //if var is something else this will execute
                 stmt;
}
```

OPERATORS

- Arithmetic (+,-,*,/,%)
- Relational (<,>,<=,>=,==,!=)
- Logical (&&,||,!)
- Bitwise (&,|)
- Assignment (=)
- Compound assignment(+=,*=,-=,/=,%=,&=,|=)
- Shift (right shift >>, left shift <<)

STRING FUNCTIONS

- strlen(str) – To find length of string str
- strrev(str) – Reverses the string str as rts
- strcat(str1,str2) – Appends str2 to str1 and returns str1
- strcpy(st1,st2) – copies the content of st2 to st1
- strcmp(s1,s2) – Compares the two string s1and s2

NUMERIC FUNCTIONS

- pow(n,x) – evaluates n^x
- ceil(1.3) – Returns 2
- floor(1.3) – Returns 1
- abs(num) – Returns absolute value
- log(x)- Logarithmic value
- sin(x)
- cos(x)
- tan(x)

PROCEDURES

- Procedure is a function whose return type is void
- Functions will have return types int, char, double, float or even structs and arrays
- Return type is the data type of the value that is returned to the calling point after the called function execution completes

FUNCTIONS AND PARAMETERS

- Syntax of function

Declaration section

<<Returntype>> funname(parameter list);

Definition section

<<Returntype>> funname(parameter list)

{

body of the function

}

Function Call

Funname(parameter);

EXAMPLE FUNCTION

```
#include<stdio.h>
void printName();                    //declaration
void main ()
{
      printf("Hello ");              //Call
      printName();
}
void printName()
{
       printf("Javatpoint");
}
```

ARRAYS

- Arrays fall under aggregate data type
- Aggregate – More than 1
- Arrays are collection of data that belong to same data type
- Arrays are collection of homogeneous data
- Array elements can be accessed by its position in the array called as index

ARRAYS

- Array index starts with zero
- The last index in an array is num – 1 where num is the no of elements in a array
- int a[5] is an array that stores 5 integers
- a[0] is the first element where as a[4] is the fifth element
- We can also have arrays with more than one dimension
- float a[5][5] is a two dimensional array. It can store 5x5 = 25 floating point numbers
- The bounds are a[0][0] to a[4][4]

STRUCTURES

- Structures are user defined data types
- It is a collection of heterogeneous data
- It can have integer, float, double or character data in it

•We can also have array of structures

```
struct <<structname>>
{
   members;
}element;
```

We can access element.members;

STRUCTURES

```
struct Person
{
int id;
char name[5];
}P1;
P1.id = 1;
P1.name = “vasu”;
```

TYPE DEF

- The typedef operator is used for creating alias of a data type
- For example I have this statement

  typedef int integer;

  Now I can use integer in place of int
  i.e instead of declaring int a;, I can use

  integer a;

  This is applied for structures too.

POINTERS

- Pointer is a special variable that stores address of another variable
- Addresses are integers. Hence pointer stores integer data
- Size of pointer = size of int
- Pointer that stores address of integer variable is called as integer pointer and is declared as int *ip;

POINTERS

- Pointers that store address of a double, char and float are called as double pointer, character pointer and float pointer respectively.
- char *cp
- float *fp
- double *dp;
- Assigning value to a pointer

  int *ip = &a; //a is an int already declared

EXAMPLES

```
int a;
a=10;          //a stores 10
int *ip;
ip = &a;
                    //ip stores address of a (say 1000)
```

ip : fetches 1000

*ip : fetches 10

* Is called as dereferencing operator

CALL BY VALUE

- Calling a function with parameters passed as values

```
int a=10;                    void fun(int a)
fun(a);                      {
                                    defn;
                             }
```

Here fun(a) is a call by value.

Any modification done with in the function is local to it and will not be effected outside the function

CALL BY REFERENCE

- Calling a function by passing pointers as parameters (address of variables is passed instead of variables)

```
int a=1;
fun(&a);
```

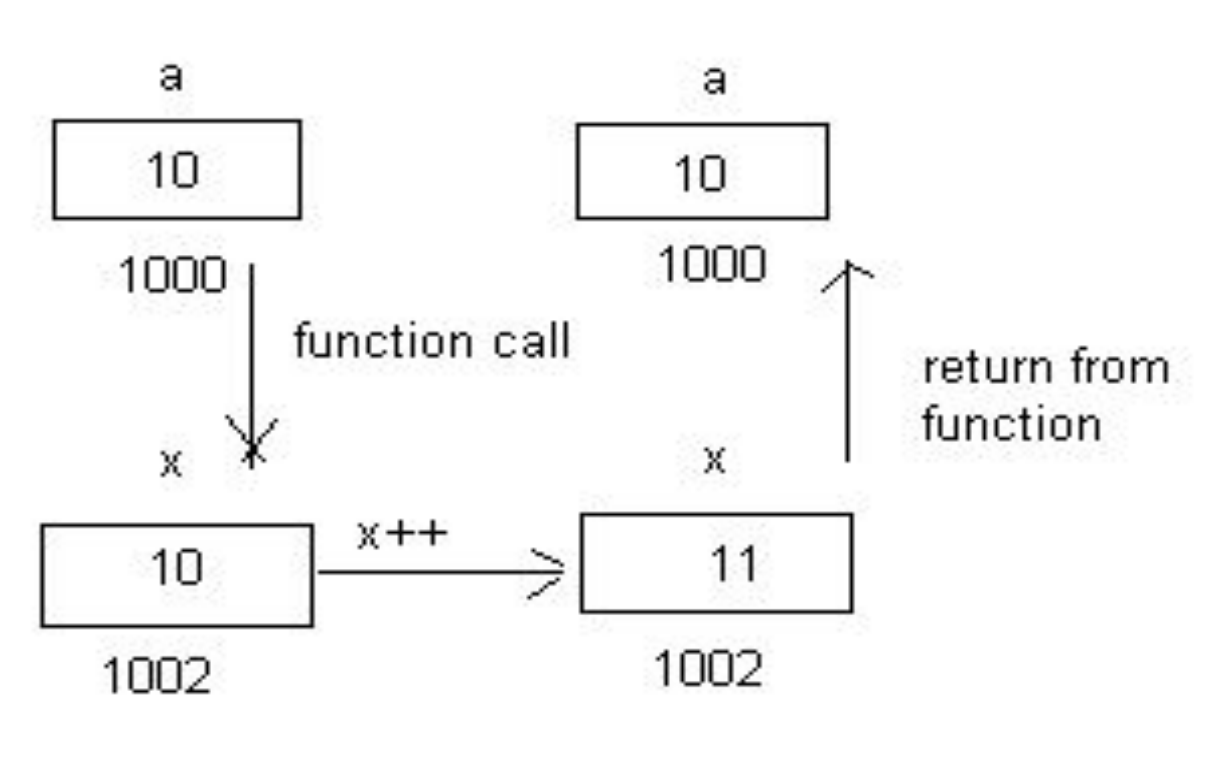
```
void fun(int *x)
{
        defn;
}
```

Any modification done to variable a will effect outside the function also

EXAMPLE PROGRAM – CALL BY VALUE

```
#include<stdio.h>
void main()
{
    int a=10;
    printf("%d",a);                    a=10
    fun(a);
    printf("%d",a);                    a=10
}
void fun(int x)
{
    printf("%d",x)                     x=10
    x++;
    printf("%d",x);                    x=11
}
```

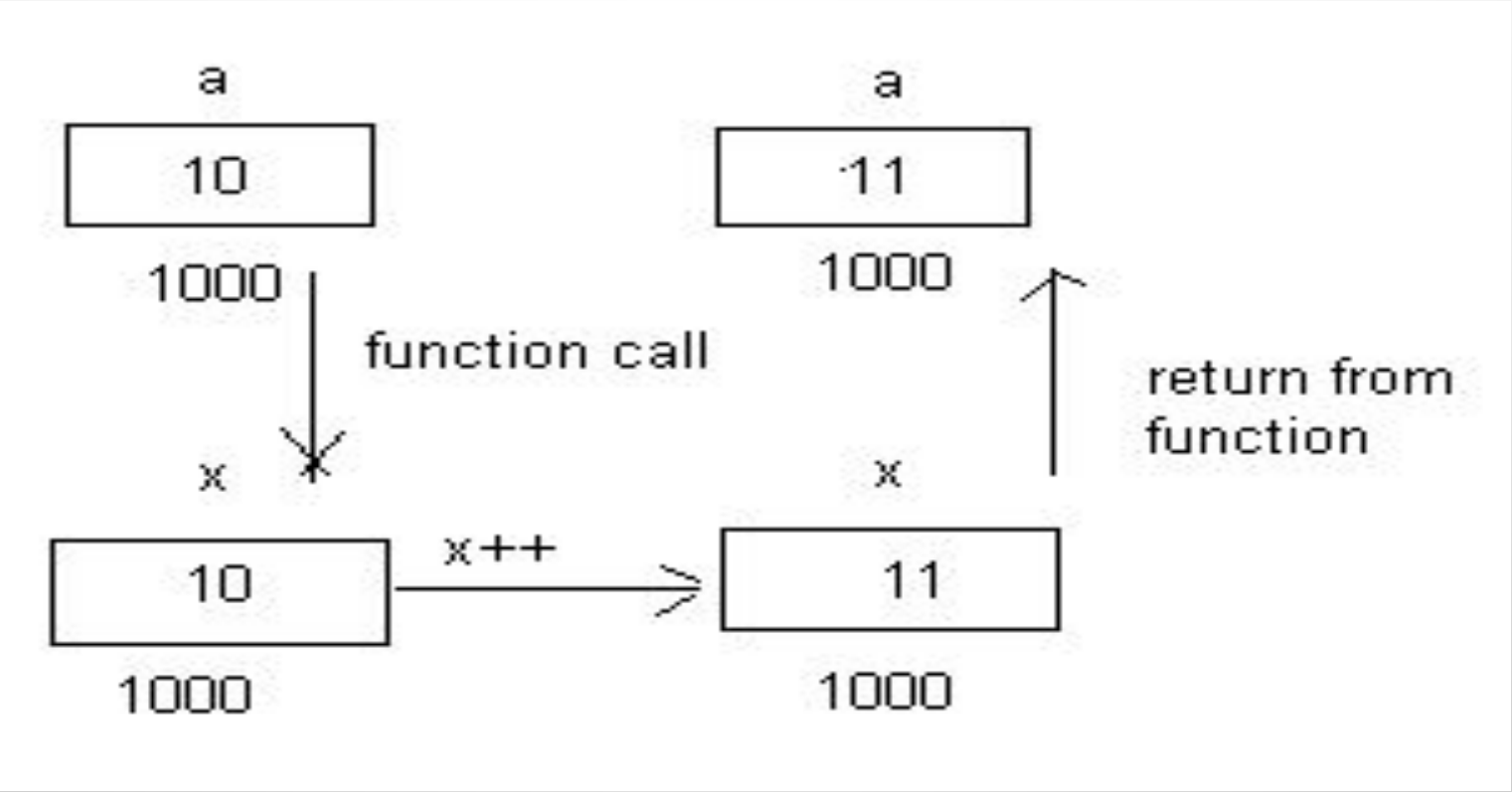
EXPLANATION



EXAMPLE PROGRAM – CALL BY REFERENCE

```
#include<stdio.h>
void main()
{
    int a=10;
    printf("%d",a);          a=10
    fun(a);
    printf("%d",a);          a=11
}
void fun(int x)
{
    printf("%d",x)           x=10
    x++;
    printf("%d",x);          x=11
}
```

EXPLANATION



a and x are referring to same location. So value will be over written.

CONCLUSION

- Call by value => copying value of variable in another variable. So any change made in the copy will not affect the original location.
- Call by reference => Creating link for the parameter to the original location. Since the address is same, changes to the parameter will refer to original location and the value will be over written.