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

Features of C language

- C has relatively easier syntax.
- Efficient and fast programming language
- Contains vast set of data types
- Has 32 most commonly used keywords
- Highly portable language
- has got rich set of library functions
- C is an extendible language
- Has support for graphics programming

Structure of C program

Documentation Section

Link section

Definition section

Global definition section

main() function section

{

Declaration part

Executable part

}

Subprogram section

Function 1

Function 2

-

-

Function n

Program structure

A sample C Program

```
#include<stdio.h>
int main()
{
   --other statements
}
```

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 a 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

Character Set

- Characters in C are grouped into following categories
  - Letters
  - Digits
  - Special characters
  - White spaces

Variables

- Variables are data that will keep on changing
- Declaration

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

  int a;
- Definition

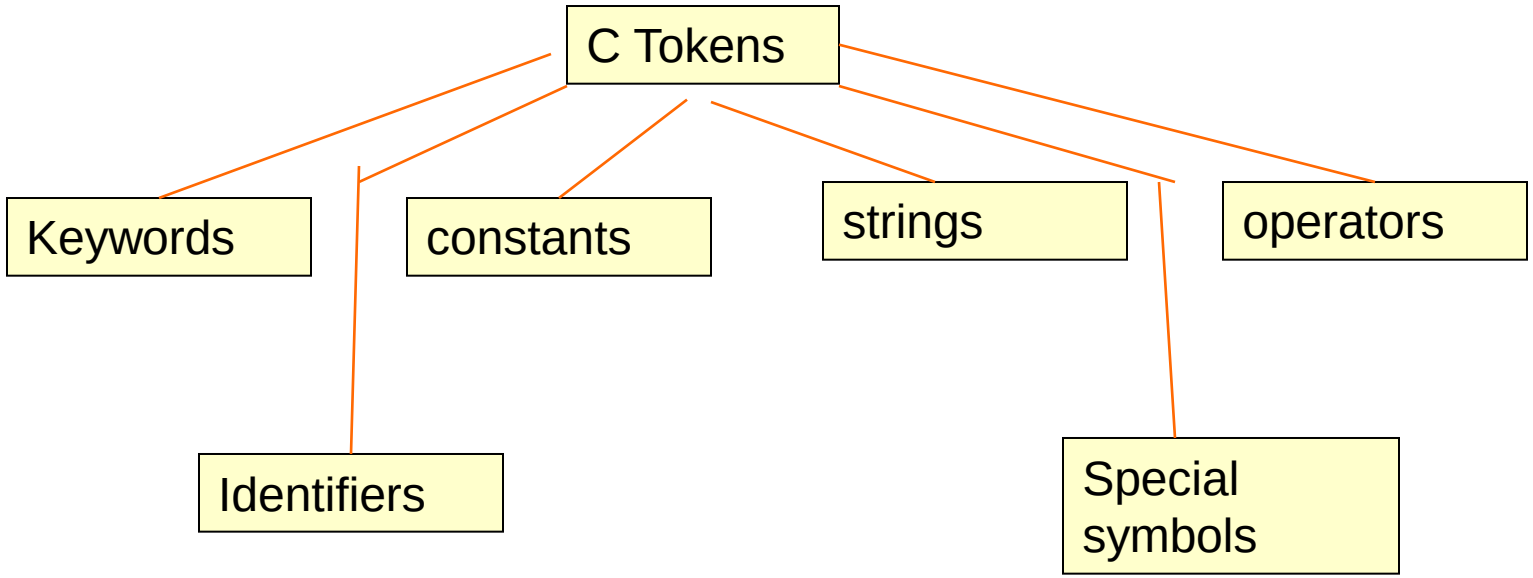
  <<varname>>=<<value>>;

  a=10;
- Usage

  <<varname>>

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

C TOKENS:


C Tokens
Keywords
constants
strings
operators
Identifiers
Special symbols

Data types in C

- Primitive data types
  - int, float, double, char
- Derived 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.

Variable

- A variable is a data name that may be used to store data value

Variable names- Rules

- Should not be a reserved word like int etc..
- Should start with a letter or an underscore(_)
- Can contain letters, numbers or underscore.
- No other special characters are allowed including space
- Variable names are case sensitive
  - A and a are different.

Declaring variables

data_type variable_name;

data_type v1,v2,v3…vn;

- **Example**

int roll_no;

float a,b,c;

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

OPERATORS

- C operators can be classified into following categories
  - Arithmetic operators
  - Relational operators
  - Logical operators
  - Assignment operators
  - Increment and decrement operators
  - Conditional operators
  - Bitwise operators
  - Special operators

Arithmetic operators

Operator	Meaning
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulo division

Relational operators

Opeartor	Meaning
<	Is less than
<=	Is less than or equal to
>	Is greater than
>=	Is greater than or equal to
==	Is equal to
!=	Is not equal to

Logical operator

Operator	Meaning
&&	Logical AND
||	Logical OR
!	Logical NOT

Assignment operator

Statement with simple assignment operator	Statement with shorthand operator
a=a+1	a+=1
a=a-1	a-=1
a=a*(n+1)	a*=n+1
a=a/(n+1)	a/=n+1
a=a%b	a%=b

Increment and decrement operator

Pre increment **++m;**	**Post increment** **m++;**
Pre decrement --m;	Post decrement m--;

Bitwise operators

Operator	Meaning
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR
<<	Shift left
>>	Shift right

Special operators

- The comma operator
- sizeof operator

Arithmetic Expressions

- An arithmetic expression is a combination of variables,constants and operators arranged as per syntax of the language.

Evaluation of Expression

- Expressions are evaluated using an assignment statement of the form
  - Variable= expression ;
  - Example
    - x=a*b-c;
    - y=b/c*a;
    - z=a-b/c+d;

Precendence of Arithmetic operators

High priority	*** / %**
Low priority	**+ -**

Rules for evaluation of Expression

- First, parenthesized sun expression from left to right are evaluated
- If parenthesizes are nested, the evaluation begins with the inner most sub expression
- The precedence rule is applied in determining the order of application of operators in evaluating sub expression
- Arithmetic expressions are evaluated from left to right using rules of precedence
- When parentheses are used, the expressions within parentheses assume highest priority.

Input and output operations

- Reading a character
  - Variable_name=getchar();

Example

```
char name;
name= getchar();
```

Formatted input

- Formatted input refers to an input data that has been arranged in a particular format

  formatted data with scanf() function

  scanf("control string",arg1,arg2…arg n);

  Control string specifies the field format in which data is to be entered and arguments arg1,arg2 etc specify address of location where data is stored

Parts of control string or format string

- scanf("format string",&var1,&var2,…&varn);

- Format string consists of
  - Conversion character(%)
  - Data type character(or type specifier)

  - Example
    - Scanf("%d%f%d",&a,&b,&c);

Formatted output

- printf statement provides certain features that can be effectively exploited to control the alignment and spacing of printouts on terminal
- General form of printf statement
  - Printf(“control string”,arg1,arg2…argn);
  - Parts of control string
- Format string consists of
  - Conversion character(%)
  - Data type character(or type specifier)
  - Example
    - printf(the value stored in x is%d”,x);

Flow control

- Flow control statements are used to alter the execution flow of a program.

control statements

Branching	if,if-else,if-else-if
Looping	while,do while,for
Jumping	break,continue,goto

DECISION MAKING STATEMENTS

- If statement
  - Syntax

```
if(condition)
{
    statement1;
    statement2;
    -----
}
```

Example

```
    if(marks>=35)
    {
        printf("you are pass");
    }
```

- If-else construct
  - Syntax

```
if(condition)
{
    statement block1
}
else
{
    statement block2
}
```

- If-else-if
  - Syntax

```
if(condition1)
{
    statement block1
}
elseif(condition2)
{
    statement block2
}
elseif(condition3)
{
    statement block3
}
```

Switch case construct

- Syntax
  - Switch(variable)
  - {

```
case value 1:
        statement block1
        break;
-------
-------
case value n:
        statement block n;
        break;
default:
        default statement block
        break;
}
```

Iteration or looping

- The concept of repeating the execution of a particular block of statements till a conditional expression is satisfied is called as iteration or looping.

For loops

- The syntax of for loop is

```
for(initialisation; condition checking ;increment/decrement)
    {
       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(condn)
{
        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

Jumping statements

- goto statement

  syntax:

  goto lable

  statements

  .....

  label: statements

  statements

  .....

- break statement:

  is used to terminate the loops.

- continue statement:

  is used to skip the remaining statements in the loop body and take control to the beginning of the loop.

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 fun(int a);                    //declaration
int main()
{
   fun(10);                         //Call
}
void fun(int x)                     //definition
{
   printf("%d",x);
}
```

Actual and Formal parameters

- Actual parameters are those that are used during a function call
- Formal parameters are those that are used in function definition and function declaration

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 s1 and s2
- strcmpi(s1,s2) – Case insensitive comparison of strings

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)

Arrays

- Arrays fall under user defined data type
- 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]

Types of Array

- One dimensional array
- Two dimensional array
- Multidimensional array

Declaration of one dimensional array

type variable_name[size];

- type specifies the type of element that will be contained in the array such as int, float or char.
- Size indicates the maximum number of elements that can be stored inside the array.

example:

int group[10];

float height[50];

Two dimensional array

- C allows us to define tables of items by using two dimensional array

Declaration of two dimensional array

- Type array_name[row_size][column_size];
- Example:
  - int a[4][3];

	coloumn-0	coloumn-1	coloumn-2
Row-0	**a[0][0]**	**a[0][1]**	**a[0][2]**
Row-1	a[1][0]	a[1][1]	a[1][2]
Row-2	a[2][0]	a[2][1]	a[2][2]
Row-3	a[3][0]	a[3][1]	a[3][2]

Initialization of arrays

- Initializing one dimensional array
  - Type array_name[size]={list of values};
  - Example:

    int number[3]={0,0,0};
- Initializing twodimensional array
  - Type array_name[row][column]={list of values};
  - Example:

    int number[2][3]={{0,0,0},{1,1,1}};

Multi dimensional array

- C allows array of three or more dimensions.
- The exact limit is determined by compiler.
- General form of a multi dimensional array is:
  - Type array_name[s1][s2][s3]….[sn];
    where si is the size of the ith dimension.

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)

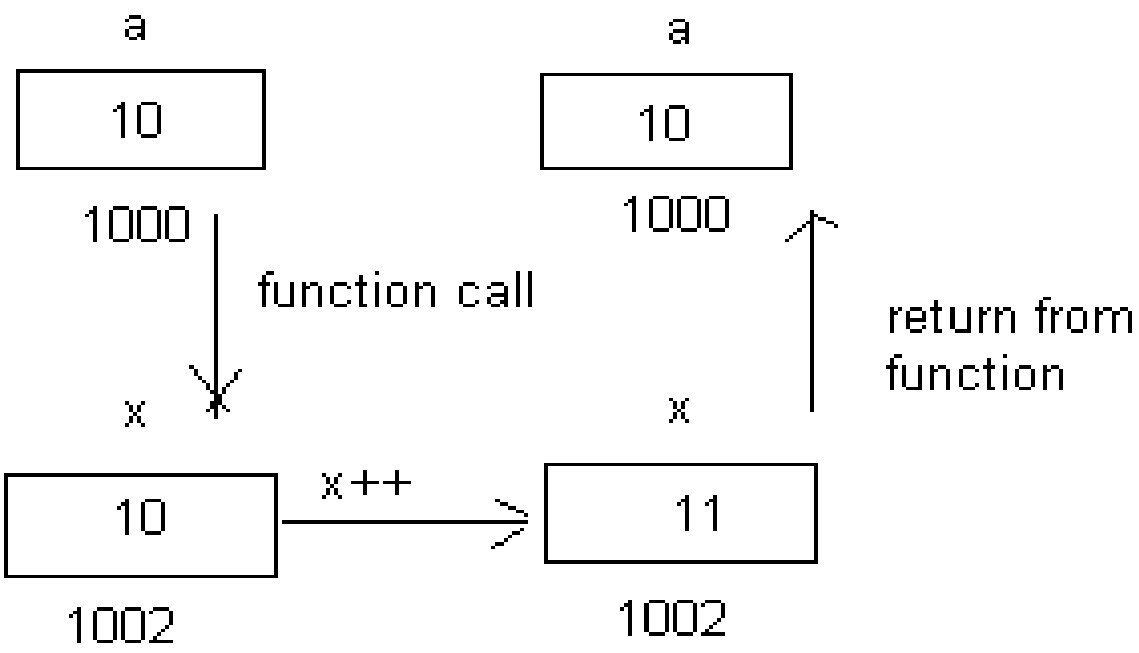
```
int a=1;                    void fun(int *x)
fun(&a);                    {
                                    defn;
                            }
```

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
}
```

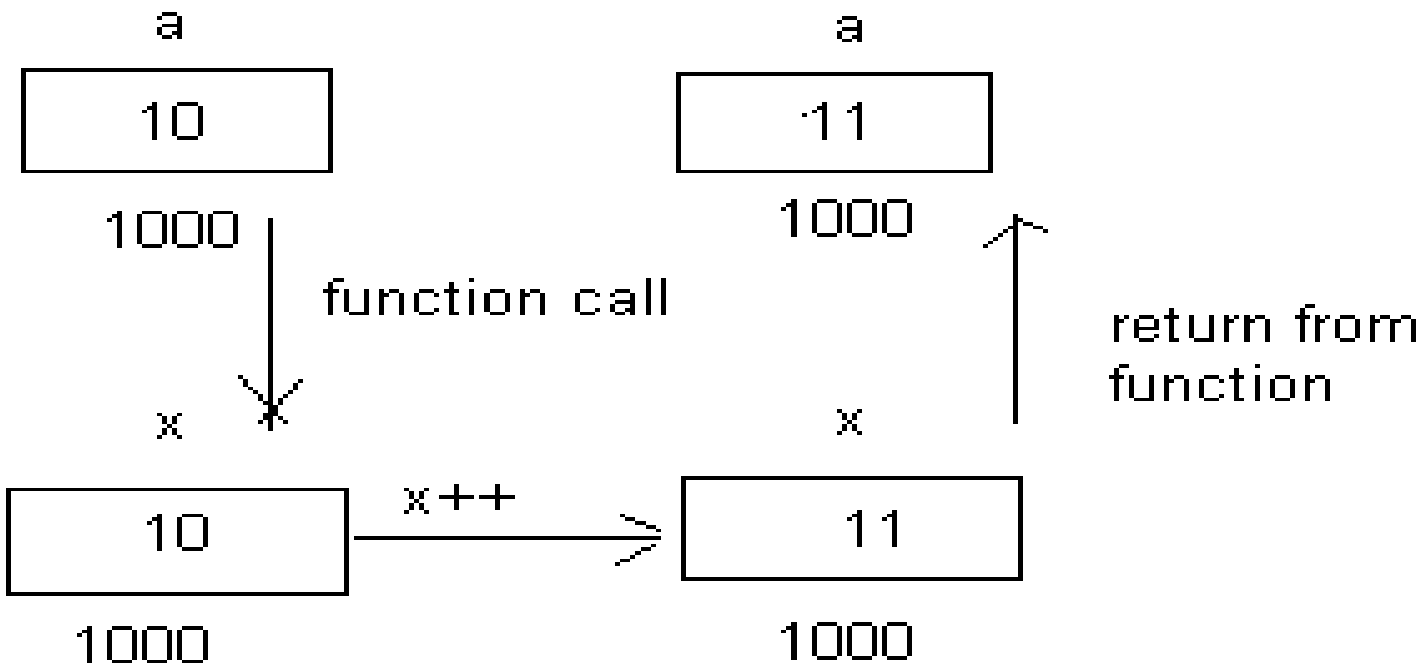
Explanation



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.