Chapter 2

Programming in C

Introduction:

Three important aspects of any language are the way it stores data, how it accomplishes input and output, and the operators it uses to transform and combine data.

C is a programming language, developed at AT & T's Bell Laboratories of USA in 1972. It was designed and written by a man named Dennis Rithcie.

Let us now see how does C compare with other programming languages. All the programming languages can be divided into two categories.

(a) Problem oriented language or high level language: These languages have been designed to give a better programming efficiency, that is faster program development. Example of languages falling in this categories are FORTRAN, BASIC, PASCAL etc.

(b) Machine oriented languages or low level languages: These language have been designed to give a better machine efficiency, that is faster program execution. Example of language falling in this category are Assembly language and machine language, C stands in between there two categories.

Sample C program: in turbo C++ compiler

```
#include <stdio.h>
#Include<conio.h>
        void main ( )
        {
                /* ..... printing begins .....*/
                printf("I see, I remember");
        /* ..... printing .....*/
        getch ( );
        }
```

This program when executed, will produce the following output:

I see, I remember.

The first line informs the system that the name of the program is main and the execution begins at this line. The main ( ) is a special function used by the C system to tell the computer where the program starts. Every program must have exactly one main function.

The opening brace '{' in the second line marks the beginning of the function main and the closing brace '}' in the last line indicates the end of the function.

All the statements between these two braces form the function body. The function body contains a set of instrunctions to perform the given task.

The line beginning with /* and ending with */ are known as comment lines. There are used in a program to enhance in readability and understanding. Comment lines are not executable statements and therefore anything between /* and */ ignored by the compiler.

Here printf is predefined, standard C function for printing output. Predefined means that it is a function that has already been written and compiled and linked together without program at the time of linking.

Every statement in C should end with a semicolon (;) mark.

stdio.h referes to the standard I/O header file containing standard input and output function.

Basic structure of C program.

Documentation Section
Link section
Definition Section
Global Declaration Section
Main ( ) function Section {
Declaration Part
Executable Part }
Subprogram section

Function 1
Function 2
.
.
.
Function n

$\Rightarrow$ User-defined functions

Executing A 'C' program.

Executing a program written in C involves a series of steps:

1. Creating the program.
2. Compiling the program.
3. Linking the program with functions that are needed from the C library.
4. Executing the program.

C Tokens:

In a passage of text, individual word and punctuation marks are called tokens. Similarly, in a C program the smallest individual units are known as C tokens.

C has six types of tokens:

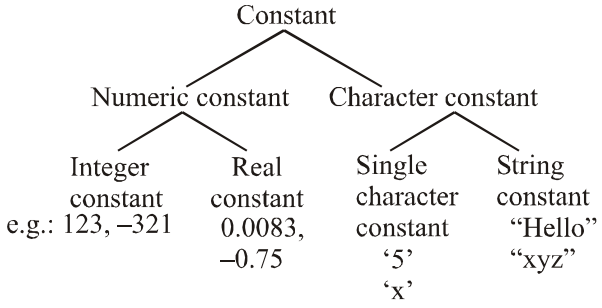
1. Keywords 2. Indentifiers 3. Constants 4. Strings
5. Special Symbols 6. Operators.

1. **Keywords:** All keywords have fixed meanings and there meanings cannot be changed. Keyword serves as basic building blocks for program statements. The list of keywords are given below:

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

2. **Identifier:** Indentifier refers to the names of variables, functions and arays. It consists of user-defined names sequence of letter and digits, with a letter as a first character. Both uppercase and lowercase letters are permitted, although lowercase letters are commonly used. The underscore character is also permitted in identifiers. It is usually used as a link between the word in long identifiers.

3. Constants: Constants in C refer to fixed values that do not change during the execution of a program.

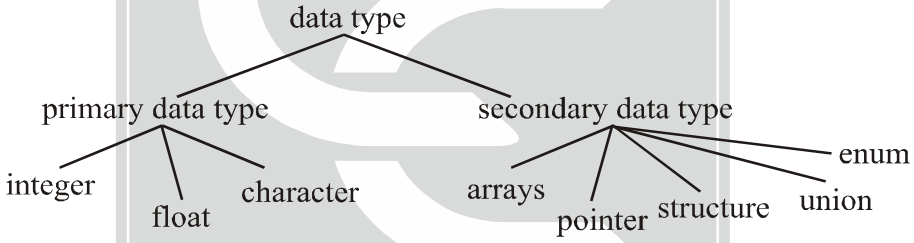


Variables: A variables is a data name that may be used to store a data value.

A variable name can be choosen by the programme in a meaningful way so as to reflect its function or nature in the program.

Some valid variable names are mark, value, roll_no, sum etc.
Invalid variable name:
123, (area), %

Data Types:
C consists of two major categories of data type:
(a) Primary data type
(d) Secondary data type.



Size and Range of Primary Data Type:

S.no.	Type	Size(bits)	Range
1.	int	16	–32768 to 32767
2.	char	8	–128 to 127
3.	Float	32	3.4 e –38 to 3.4e + 38

Declaration of variables:
Primary type declaration:

data-type v_1, v_2, ... v_n;

v_1, v_2, ... v_n are the names of variables.

For example
int count;
int numbers, total;
double ratio;

Storage Classes: There are four storage classes in C language.

1. auto: Local variable are known only to the function in which it is declared. Default is auto.

2. static: Local variables are the variables which exists and retains its value even after the control is transfered to the calling function.

3. extern: Global variable known to all function in the file.

4. register: Local variables are the variables which are stored in the register.

Static and extern variables are automatically initialized to zero. Automatic (Auto) variable contains undefined value (known as garbage) unless they are initialized explicity.

Reading Data from keyword:

To input data through keyboard using the scanf function. It is a general input function available in C.

General format of scanf.

scanf("control string", &v_1, &v_2 ..... &v_n);
scanf("%d", & number);
%d = Specify data type
& = Address

Example:

```
#include<stidio.h>
#include<conio.h>
        void main ( )
        {
                int x,y, sum;
                printf("Enter the vaue of x:");
                scanf("%d", &x);
                printf("Enter the value of y:");
                scanf("%d", &y);
                        sum = x+ y;
        printf("%d\n : %d", sum);
        getch( );
        }
```

Operators:

C operators can be classified into number of categories. They include:
(a) Arithmetic Operators
(b) Relational operators
(c) Logical operators
(d) Assignment operators.
(e) Increment and decrement operators
(f) Conditional operators.
(g) Bitwise operators.
(h) Special operators.

Arithmetic Operators:

Operator	Meaning
+	Addition
–	Subtraction
*	Mutliplication
/	Division
%	Modulo

For example:

a = 14, b = 4
a–b = 10
a+b = 18
a*b = 56
a/b = 3
a%b = 2

Write a program that use of integer arithmetic to convert a given number of days into month and days.

```
#include <stdio.h>
#include<conio.h>
void main ( )
{
        int months, days;
        printf("Enter days");
        scanf("%d", &days);
        months = days / 30;
        days = days % 30;
        printf("\nmonths = %d  \ndays = %d", months, days);
        getch( );
}
```

Output:

```
Enter days      :    265
Months = 8      :    days = 25
Enter days      :    364
Months = 12     :    days = 4
```

Relational Operators:

Operators	**Meanings**
<	Less than
< =	Less than or equal to
>	Greater than
> =	Greater than or equal to
= =	Equal to
! =	Not equal to

An expression such as a < b or 1 < 20 containing a relational operator is termed as a relational expression. The value of a relational expression is either one or zero. It is one if the specified relation is true and zero if the relation is false.

For example:

10 < 20	True
20 > 100	False

e.g.

4.5 < = 1000	True
–5 > 0	False
10 < 11	True

Logical operators:

C has the following three logical operators.

```
&&      :    Logical AND
||      :    Logical OR
!       :    Logical NOT
```

The logical operators && and | | are used when we want to test more than one condition and make decision. An example is:

```
a > b   &&   x = = 10
```

An expression of this kind which combines two or more relational expression is called logical expression.

Truth Table:

&& (logical AND)

op1	op2	(op1 && op2)
non-zero	non-zero	1
non-zero	0	0
0	non-zero	0
0	**0**	**0**

|| logical OR

op1	op2	(op1 || op2)
non-zero	non-zero	1
non-zero	0	1
0	non-zero	1
0	0	0

Assignment operators:

Assignment operators are used to assign the result of an expression to a variable. It is represented by '='.

For example

1. x = 3;
2. x + = y + 1;
   equivalent to
   x = x + (y + 1);
3. x + = 3;
   equivalent to
   x = x + 3;

Write a program that prints a sequence of squares of numbers.

```
# define        N       100
# define        A       2
void main ( )
{ int a;
        a = A;
        while (a < N)
        {       printf(\n%d", a);
                a* = a;} getch( );   }
```

Output

2
4
16

Increment And Decrement Operators:

Increment : ++
Decrement : – –

Both of these unary operators are further classified as pre and post operators.

For example

– – m OR ++m [pre operator], m– – OR m + + [post operator]

++ m is equivalent to m = m + 1

– – m is equivalent to m = m –1

In pre increment and decrement operators, the value is decrement by 1 first then it is assigned to the variable. Where ask in post increment or decrement operator , the value of the variable is firstly assigned within the expression, thereafter its value is incremented/decremented by 1.

For example:

Let x = 5, y = 6 and z = 7.

x = (y– –) + (– – z)

After the execution of this expression the value of x, y and z will be

x = 12, y = 5, z = 6

Conditional Operators:

A ternary operator "?: " is available in C to construct conditional expression of the form.

exp1? exp 2 : exp 3

Where exp1, exp2 and exp3 are expression.

The operator ?: works as follows: exp1 is evaluated first.

If exp1 is true than exp2 will be evaluated otherwise exp3 will be evaluated.

For example:

a = 10, b = 15;

x = (a > b)? a:b

Here X will be assigned the value of b.

Bitwise Operator

Operator	**Meaning**
~	One's complement
> >	Right shift
<<	Left Shift
&	Bitwise AND
|	Bitwise OR
^	Bitwise XOR

These operator can operate upon ints and chars but not on floats and double.

One's Complement:

$1111 \rightarrow 0000$

$0110 \rightarrow 1001$

Right Shift Operator:

It is represented by >> and it shifts each bit in the operand. The blanks are created on left and filled with zero.

For example:

a = 26

Binary form

a = 11010

a > > 1

11010 >> 1

= 01101= 13

Left Shift Operator:

This is similar to the right shift operator, the only differences being that the bits are shifted to the left. It is represented by <<

For example:

a = 26

a = 11010

a = <<1

= 10100

= 20

Bitwise AND Operator:

This operator is represented by &

a = 10101101
b = 00001000

AND = 00001000

The best use of the bitwise AND is to check whether a particular bit of an operand is ON or OFF.

Bitwise OR Operator:

This operator is represented by 1.

a = 10101101
b = 00001000

OR = 10101101

Bitwise OR operator is usally used to put ON a particular bit in a number.

Bitwise XOR Operator:

The XOR operator is represented by ^ and is also called an exclusion OR operator.

XOR is used to toggle a bit ON or OFF i.e. if both bits are same then outcome will be zero, otherwise 1

a = 10101101
b = 0001000

XOR = 10111101

Problem: Assunming, integer is 2 byte, What will be the output of the program?

```
#include<stdio.h>
int main()
{
printf("%x\n", -1>>1);
return 0;
}
```

(a) ffff (b) 0fff (c) 0000 (d) fff0

Soln. Negative numbers are treated with 2's complement method.

1's complement: Inverting the bits (all 1s to 0s and all 0s to 1s)

2's complement: Adding 1 to the result of 1's complement.

Binary of 1(2byte) : 0000 0000 0000 0001

Representing -1:

1s complement of 1(2byte) : 1111 1111 1111 1110

Adding 1 to 1's comp. result : 1111 1111 1111 1111

Right shift 1bit(-1>>1): 1111 1111 1111 1111 (carry out 1)

Hexadecimal : f f f f

(Filled with 1s in the left side in the above step)

Note:

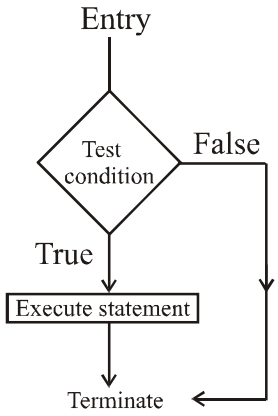
1. Fill with 1s in the left side for right shift for negative numbers.
2. Fill with 0s in the right side for left shift for negative numbers.
3. Fill with 0s in the left side for right shift for positive numbers.
4. Fill with 0s in the right side for left shift for positive numbers.

Control Statements :

In C programs, statements are executed sequentially in the order in which they apear in the program. But sometimes we may want to use a condition for executing only a part of program. Also many situations arise where we may want to execute some statements are to be executed. They define how the control is transferred to other parts of the program.

Decision Making:

1. If statement
2. Switch statement
3. Conditional operator statement.



The if statement:

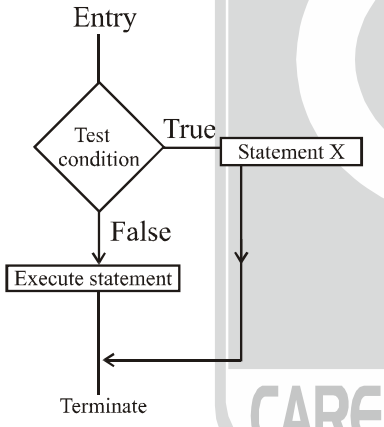
The general form of if statement

(a) If (this condition is true)

Execute this statement

(b) If (test expression)

```
{
        statement X;
}
        statement Y;
```



For example:

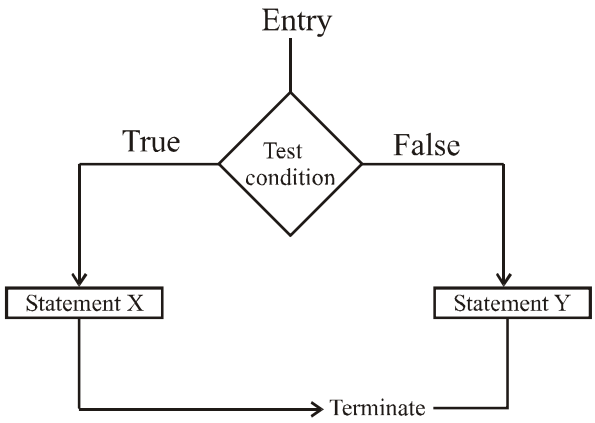
```
if (category = = sports)
{
        marks = marks + bonus_marks;
}
printf("%f", marks);
```

(c) The if ..... else statement

The if.....else statement is an expression of the simple if statement

The general form

```
if (test condition){
        Statement X(True)
}
else
{
        Statement Y(False);
}
```



For example: Write a program to check whether a number is odd or even numbers:

```
void main ( )
{
        int x;
        printf(" Enter x:");
        scanf("%d", &x);
        if (x%2 = = 0)
                printf("Even Numbers");
        else
                printf("odd number");  getch ( );  }
```

(d) Nesting of if...else statements.

When a series of decision are involved, we may have to use more than one if ...else statement in nested form as follows:

```
if (test condition 1)
{
        if(test condition2)
        {
                statement 1;
        }
        else
        {
                statement 2;
        }
}
else
{
        statement 3;   }
```

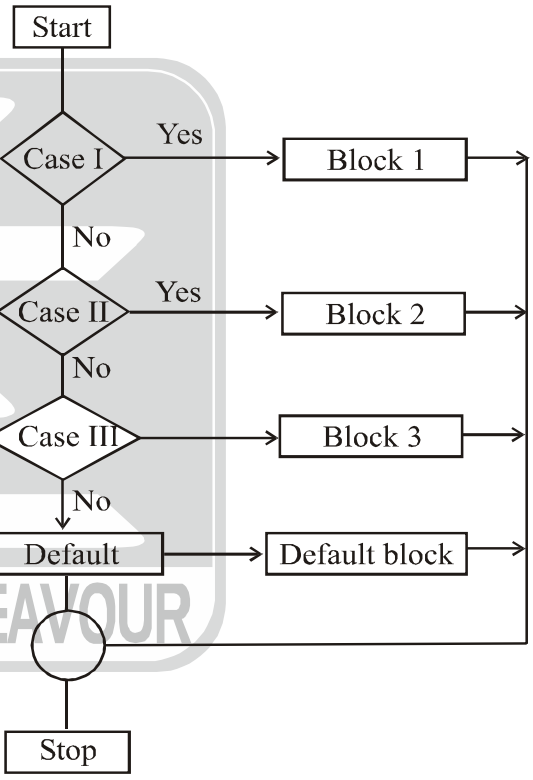
Write a program to print the largest of the three numbers.

```
void main ( )
{
        int A, B, C;
        printif ("Enter three number:");
        scanf("%d%d%d", &A, &B, &C);
        printf("The largest value is");
If (A > B)
        {
```

```
                    if (A > C)
                    printf (A>C)
                        printf("%d", A);
        }
            if (C>B)
            printf("%d", C)
            else
            printf("%d", B);  getch ( );
        }
}
```

The switch Statement:

C has a built-in multiway decision statement know as a switch. The switch statment tests the value of given variable against a list of case values and when a match is found, a block of statements associated with that case is executed.



The general form of the switch statement.

```
switch(expression)
        {
case 1:
        block 1;
        break;
case 2:
        block 2;
        break;
.
.
.
default:
        default block
        break;  }
```

For example:

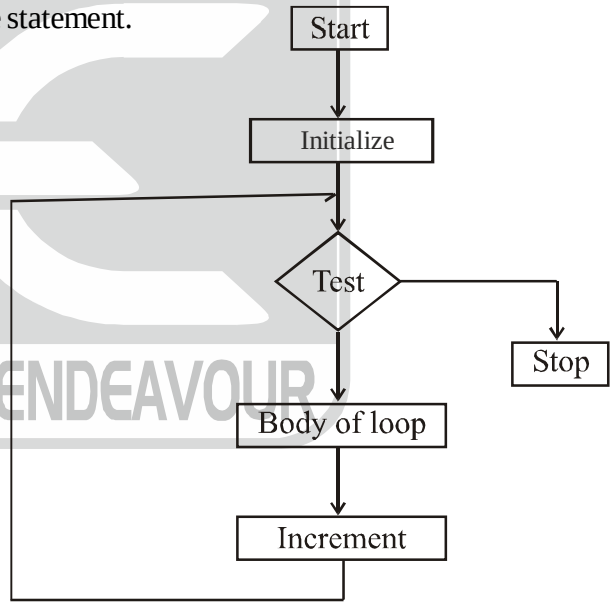
```
void main ( )
{
        int i = 2;
        switch (i)
        {
            case 1:
                printf(" I am A");
                break;
            case 2:
                printf (" I am B");
                break;
            case 3:
                printf("I am C");
                break;
            default:
                printf(" I am Z");
        }       getch ( );
}
```

Output
I am B

Important note on switch statement
1. You can put "case" in any order
2. You are also allowed to use char values in case
   case 'a'
   case 'y'
3. You can mix integer and character constant in different casesof switch
   case 'v'
   case 2.
4. The following switch statement are legal
   switch (i + y * k)
   switch (23 + 45% 4*k)
   swich (a < 4 && b > 7)
5. The switch statement is very useful while writing menu driven program.
6. Swtich is not allowed to use floating value in case.

1. while statement
2. do-while statement
3. for statement.

The while loop statement:
The simplest of all the looping structure in C is the while statement.



The general format of the while statement.

```
while (test condition)
{
        body of the loop

}
```

For example:

```
main ( )
{
        int n, h;
        int sum = 0;
        n = 1;
        while (n < = 10)
                {
                        sum = sum + n*n;
                        n = n + 1;
        }
                printf(" sum: %d", sum);
        }
```

Condition in while loop use either relational operator or logical.

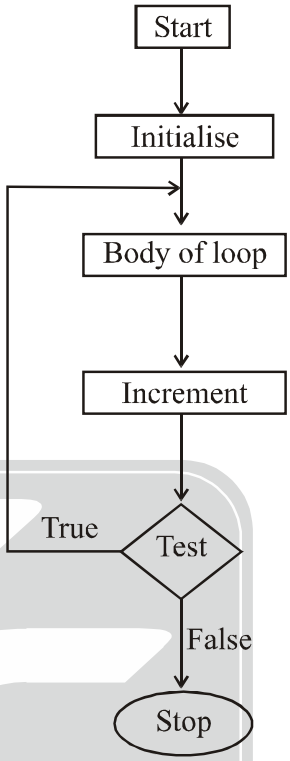
```
while (i < = 10)
while (i > = 10  &&  j < = 15)
```

The do-while loop statement:

The do-while loop would execute in statements at least once, even if the condition fails, for the first time itself.

The general format of the do-while loop

```
do
{
        body of loop
}
        while (test condition);
```



For example:

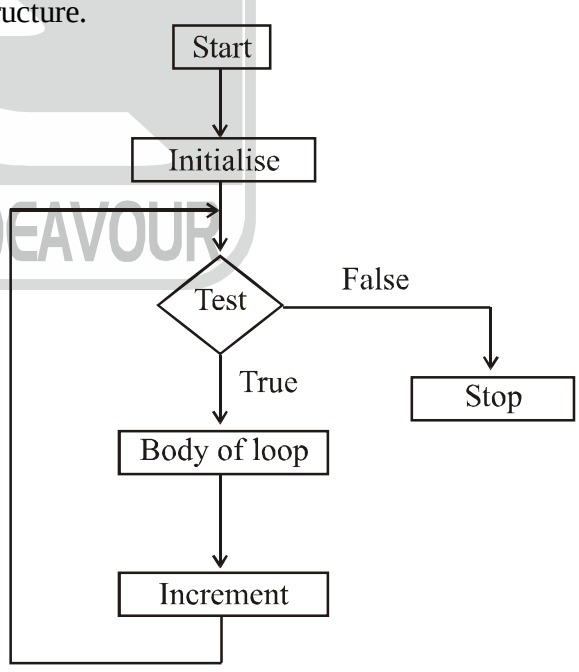
```
void main ( )
{
        int i, sum;
        i = 1;
        sum = 0;
        do
        {
                sum = sum + i;
                i = i + 2;
        }       while (sum < 40 || i < 10);
                printf ("%d %d", i, sum);
        }
getch( );
}
```

For loop statement:

The for loop is provides a more concise loop control structure.

The general form of the for loop

```
for(initialization; test condition; increment)
        {
                body of loop;
        }
```



For example:

```
void main ( )
{
        int i;
        for (i = 1; i < = 10; i = i++)
        {
                printf("%d", i);
        }       getch ( ); }
```

Some important piont of four loop

(a)

```
        void main ( )
        {
                int i;
                for (i = 1; i < = 10; )
        {
                printf("%d", i);
                i = i+1;
                getch ( ); }
```

The increment is done within the body of the loop. The semicolon after the condition is necessary.

(b)
```
void main ( )
{
        int i = 1;
        for (; i < = 10; i = i +1)
        printf (“%d”, i);
        getch ( );
}
```

Here the initialisation is done in the declaration statement itself. But still the semicolon before the condition is necessary.

(c)
```
void main ( )
{
        int i = j;
        for (; i < = 10;)
        { printf(“%d”, i);
                i = i+1;
        }
         getch ( );
}
```

Here, neither the initialisation, nor the increment is done in the for statement but still the two semicolons are necessary.

(d)
```
void main ( )
{
        int i;
        for (i = 0; i++ < 10;)
                printf(“%d”, i);
        getch ( );
}
```

Here, the comparision as well as the increment is done through the same statement, i++ < 10. Since the ++ operator come, after i firstly comaprision is done, followed by incrementation. Note that it is necessary to intialize i to 0.

FUNCTIONS

A function is a self-contained subprogram that performs some specific, well-defined task. Any C program consists of one or more functions. If a program has only one function then it must be the main ( ) function. It is the first function to be executed when the program starts, all other functions are called directly or indirectly.

- A function receives zero or more parameters, performs a specific task, and returns zero or one value.
- A function is invoked by its name and parameters.
    - No two functions have the same name in your C program.
    - No two functions have the same name and parameter types in your C++ program.
    - The communication between the function and invoker is through the parameters and the return value.
- A function is independent:
    - It is “completely” self-contained.
    - It can be called at any places of your code and can be ported to another program.
- Functions make programs reusable and readable.

Syntax:

• **Function Prototype:**

return_type function_name (type1 name1, type2 name2, ..., typen namen);

Where, Parameters = type1 name1, type2 name2, ...,typen namen

• **Function Definition:**

```
return_type function_name (type1 name1, type2 name2, ...,typen namen)
{
    ....statements...
}
```

Where, Function headers= return_type function_name (type1 name1, type2 name2, ...,typen namen)

Parameters = type1 name1, type2 name2, ...,typen namen

Some Examples:

• Function Prototype Examples

```
double squared (double number);
void print_report (int);
int get _menu_choice (void);
```

• Function Definition Examples

```
double squared (double number)
{
    return (number * number);  }

void print_report (int report_number)
    {
    if (report_nmber == 1)
    printf("Printer Report 1");
    else
    printf("Not printing Report 1");
}
```

Where, return type void means it returns nothing.

e.g. void print_report (int) & void print_report (int report_number)

void parameter list means it takes no parameters.

e.g. int get _menu_choice (void)

Passing Arguments:

• Arguments are passed as in Java and Pascal

• Function call:

```
func1 (a, b, c);
```

• Function header

```
int func1 (int x, int y, int z)
```

• Each argument can be any valid C expression that has a value:

• For example:

x = func1(x+1,func1(2,3,4),5);

• Parameters x y z are initialized by the value of a b c

• Type conversions may occur if types do not match.

Parameters are Passed by Value:

• All parameters are passed by value!!

(i) This means they are basically local variables initialized to the values that the function is called with.

(ii) They can be modified as you wish but these modifications will not be seen in the calling routine!

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

Returning a Value:

• To return a value from a C function you must explicitly return it with a return statement.

• **Syntax:**

```
 return <expression>;
```

- The expression can be any valid C expression that resolves to the type defined in the function header.
- Type conversion may occur if type does not match.
- Multiple return statements can be used within a single function (eg: inside an "if-then-else" statement…)

Local Variables:

• Local Variables

```
   int func1 (int y)
 {
    int a, b = 10;
    float rate;
    double cost = 12.55;
    .....
 }
```

• Those variables declared "within" the function are considered "local variables".

• They can only be used inside the function they were declared in, and not elsewhere.

A Simple Example:

```
#include <stdio.h>
int x=1;   /* global variable - bad! */
void demo(void);
        int main() {
        int y=2;  /* local variable to main */
        printf ("\nBefore calling demo(), x = %d and y = %d.",x,y);
        demo();
        printf ("\nAfter calling demo(), x = %d and y = %d.\n",x,y);
        return 0; }
void demo () {
        int x = 88, y =99;  /* local variables to demo */
        printf ("\nWithin demo(), x = %d and y = %d.",x,y);
}
```

Placement of Functions:

- For large programs
  - Manage related functions in a .c file
  - Write a .h file containing all the prototypes of the functions
  - #include the header file in the files that uses the functions.
- For small programs, use the following order in the only one file:
  - All prototypes
  - main() function
  - function definitions
- mymath.h

```
int min(int x,int y);
int max(int x,int y);
```

- mymath.c

```
int min(int x,int y)
{
        return x>y?y:x;
}
        int max(int x,int y)
{
        return x>y?x:y;
}
```

Recursion: The function calling itself is called recursion.

For example:

```
        unsigned int factorial(unsigned int a);
int main () {
unsigned int f,x;
        printf("Enter value between 1 & 8: ");
        scanf("%d", &x);
        if (x > 8 || x < 1)
        printf ("Illegal input!\n");
         else {
         f = factorial(x);
        printf ("%u factorial equals %u\n", x,f);
        }
        }
unsigned int factorial (unsigned int a) {
if (a==1)
        return 1;
else {
        a *= factorial(a-1);
return a;
         }  }
```