

C Programming NOTES

- **What is Programming :**

Computer programming is a medium for us to communicate with computers. Just like that we use English or Bengali to communicate with each other., programming is the way for us to deliver our commands to computer.

- **What is C?**

C is a programming language.

C is one of the oldest and finest programming language.

C was developed by Dennis Ritchie at AT&T's lab, USA in 1972.

- **Uses of C:**

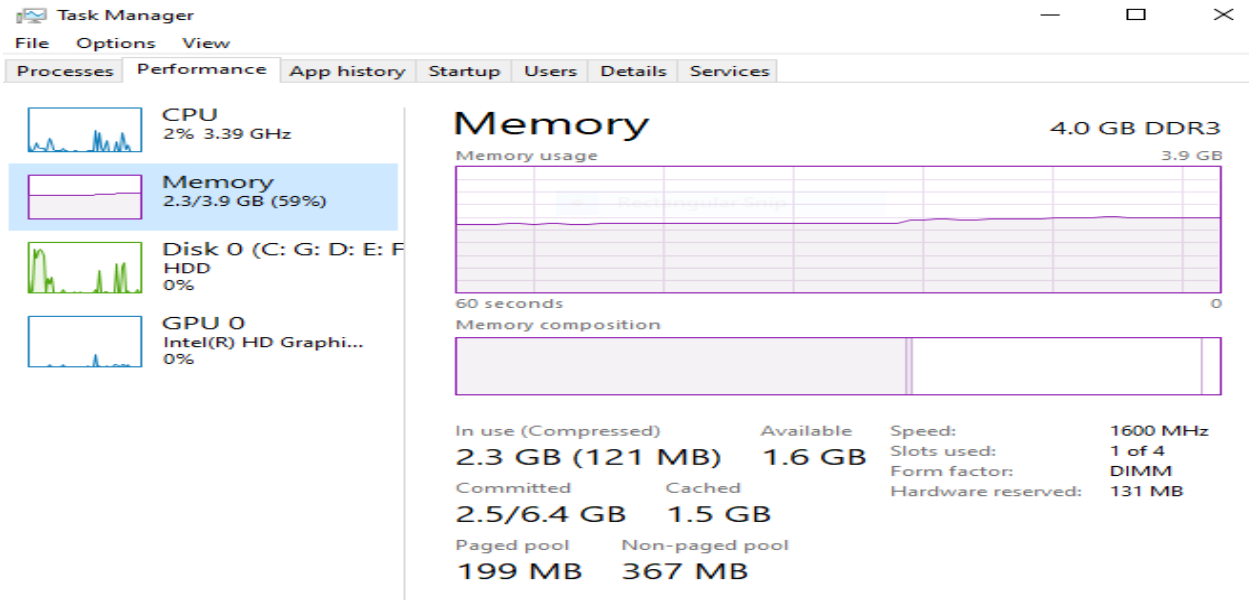
C Language is used to program a wide variety of system. Some of the uses of C are as follows.

- 1 . Major part of operating system are written in C. Such as Windows, Linux and other operating system.
2. C is used to write driver programmes for devices like tablet, printers, IOT (Internet Of Things) etc.
3. C language is used to program system where program needs to run faster in a limited memory.
4. C programme is used to develop games where latency is very important where computer is needed to react quickly on user input.

Chapter 1: Variable , Constants & Keywords

- **Variables:**

A variable is a container which stores a value in the memory. A variable is an entity where value can be changed.



As we store Rice,Dal in our kitchen with in a container.Similar to that variables in C Stores values of a contain.

Example:

```
a=3 // a is assigned '3'
b=4.7 // b is assigned '4.7'
c='A' // c is assigned 'A'
```

- **Rules For Naming Variables in C :**

1. First character must be an alphabet or a underscore(_).
2. No commas,blanks are allowed.
3. No special symbols other than(_) allowed.
4. Variable name are case sensitive.
5. We must create meaningful variable name In our programs.

- **Constant:**

An entity whose value does not change is called as a constant.

Types of Constant:

Primarily there are three types of constant.

1. Integer Constant - -> -1,6,7,9
2. Real Constant - -> -322.1 , 2.5 , 7.0
3. Character Constant - -> 'a' , '\$' , '@' (must be enclosed with inverted commas)

- **Keywords:**

Keywords are nothing but a reserved words ,whose meaning is already known to the compiler. There are total 32 keywords in C.

| 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 |

- **Our First C program:**

```
C first.c  X
C: > Users > abhi7 > OneDrive > Desktop > html+css > New folder > C first.c > main()
1  #include <stdio.h>
2  int main()
3  {
4      printf("Hello world");
5      return 0;
6  }
```

- **Basic structure of a C program**

All C programs have to follow a basic structure. A C program starts with a main function and executes instructions present inside it.

Each instruction is terminated with a semicolon(;).

There are some basic rules applicable for all C programs:

1. Every program starts with a main function.
2. All the statements are terminated with a semicolon(;).
3. Instructions are case sensitive.
4. Instructions are executed in the same order in which they are written.

• Comments:

Comments are used to clarify something about the program in plain language. It is a way for us to add NOTES to our program.

There are two types of comments in C Language.

1. Single line comment : `//This is a single line comment`

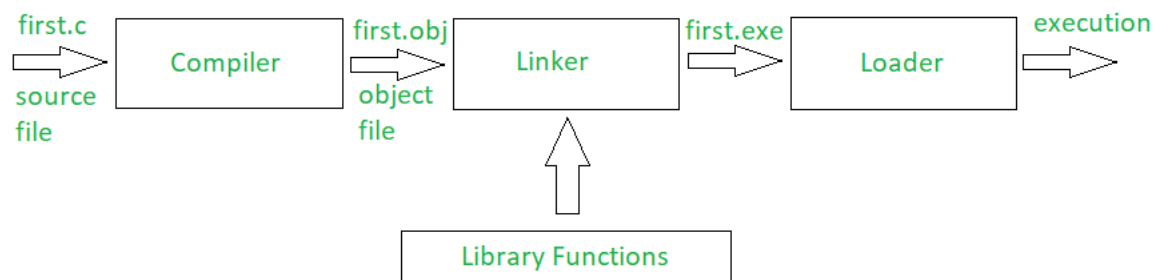
1. Multi line Comment : `/*This is a multi
line comment*/`

.

Comments in a C program are not executed and are ignored.

For multi line comment you can also select the lines and press ' `Ctrl + /` '(Forward slash).

• Compilation and execution



A compiler is a computer program which converts a C program into a machine language. So it can be easily understood by the computer.

A C program is written in plain text.

This plain text is a combination of instructions in a particular sequence.

The compiler performs some basic checks and finally converts the program into a .exe file.

- **Library Function**

C language has a lot of valuable library functions which are used to carry out some certain tasks. For instance, the "printf" function is used to print values on the screen.

```
printf(" This is %d ", i )
```

%d for integers

%f for real value

%c for character

- **Types of variable**

1. Integer variable → `int a=3;`
2. Real variable → `float a=7.7;`
3. Character variable → `char a = "B"`

- **Receiving input from User**

In order to take input from the user and assign it to a variable, we use the "scanf" function.

Syntax for using scanf :

```
scanf("%d", &i )
```

& is the address of operator and it means that the supplied value should be copied to the address which is indicated by the variable 'i'.

Practice Set For Chapter 1 :

Q1. Write a program to calculate area of a rectangle.

(a) Using hard coded inputs.

```
C practiceset.c X
c: > Users > abhi7 > OneDrive > Desktop > html+css > New folder > New folder > C practiceset.c > main()
1  #include <stdio.h>
2
3  int main()
4  {
5      int lenth=3, breadth=4;           // Value of lenth and breadth is given randomly
6      int area = lenth * breadth;
7      printf("the area of the rectangle is %d", lenth * breadth);
8      return 0;
9  }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL Rectangular Snip

```
[Running] cd "c:\Users\abhi7\OneDrive\Desktop\html+css\New folder\New folder\" && gcc pract
the area of the rectangle is 12
[Done] exited with code=0 in 1.642 seconds
```

(b) Using input supplied by User.

```
c: > Users > abhi7 > OneDrive > Desktop > html+css > New folder > New folder > C practiceset.c > main()
1  #include <stdio.h>
2
3  int main()
4  {
5      int length, breadth;
6      int area;
7      printf("Please enter the lenth of the rectangle\n",length);
8      scanf("%d" ,& length);
9      printf("please enter the breadth of the rectangle\n",breadth);
10     scanf("%d", & breadth);
11     printf("The area of the rectangle is %d",length*breadth);
12     return 0;
13 }
```

Q2. Calculate the area of a circle and modify the same program to calculate the volume of a circle given it's radius.

```
c: > Users > abhi7 > OneDrive > Desktop > html+css > New folder > New folder > C practiceset.c >
1  #include <stdio.h>
2
3  int main()
4  {
5      int radius = 3;
6      float pi = 3.14;
7      printf("The area of the circle is %f\n", pi * radius * radius);
8
9      return 0;
10 }
```

Q3. Write a program to convert celsius to fahrenheit.

```
c: > Users > abhi7 > OneDrive > Desktop > html+css > New folder > New folder > C practiceset.c > main()
1  #include <stdio.h>
2
3  int main()
4  {
5
6      float celsius = 3;
7      float far = (celsius * 9 / 5) + 32;
8      printf("The value of the celsius temperature in fahrenheit is %f", far);
9      return 0;
10 }
```

Q4. Write a program to calculate simple interest rate of a specific value of a specific interest rate for a time period using hard coded inputs.

```
c: > Users > abhi7 > OneDrive > Desktop > html+css > New folder > New folder > C practi
1  #include <stdio.h>
2
3  int main()
4  {
5      int principal = 100, rate = 5, year = 2;
6      int simpleInterest = (principal * year * rate) / 100;
7      printf("The Simple interest rate is %d", simpleInterest);
8      return 0;
9  }
```

Chapter 2 : Instruction Operators

A C program is a set of instructions . Just like a recipe which contains instructions to prepare a particular dish.

- **Types of instructions**

1. Type Declaration Instruction.
2. Arithmetic Instruction.
3. Control Instruction.

1.Type Declaration Instruction:

int a;

float b;

Other Variation:

int i=10; int j=1, int a=2;

int j= a+j-1;

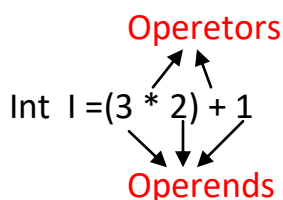
float b= a+3;

float a= 1.1 => **ERROR ! As we are trying to use before defining it.**

int a,b,c,d;

A=b=c=d=30; => **Value of a,b,c,d will be 30 each.**

2.Arithmetic Instruction :



Operands can be int/float etc.

' + , - , * , / ' are arithmetic operators.

Int b=2,c=3

Int z; z=b*c → legal

Int z; b*c=z → Illegal(not allowed)

% -> Modulo division operator.

% -> Returns the remainder.

% -> Can not be applied in float.

% -> Sign is same as of numerator.

$5/2=1$ $-5/2=-1$ $5/-2=-1$

NOTE:

1. To operator is assumed to be present.

Int i=ab; → Invalid

Int i=a*b; → Valid

2. There are no operator to perform exponentiation in C . However we can use 'Pow(x,y)' from '<math.h>'.

Type conversion :

An arithmetic operation between

Int and int → int

Int and float → float

Float and float → float

} Important

NOTE:

Int a = 3.5 [in that case 3.5(float will be demoted to 3(int) because a is not able to store float]

Float a = 8[in that case a will store 8.0(promoted to float)]

Operator precedence in C :

In C language simple mathematical rules like BODMAS , no longer applies.

The following table lists the operator priority in C.

| Priority | operator |
|-----------------|------------------|
| 1 st | $*$, $/$, $\%$ |
| 2 nd | $+$, $-$ |
| 3 rd | $=$ |

Operators of higher priority are evaluated first in the absence of parenthesis.

Operator associativity :

When operators of equal priority are present in an expression, the tie is taken care of by associativity

$$x * y / z \Rightarrow (x * y) / z$$

$$x / y * z \Rightarrow (x / y) * z$$

$*$, $/$ follows left to right associativity.

Control instructions

Determines the flow of control in a program.

Four types of control instruction in C are:

1. Sequence Control Instruction
2. Decision Control Instruction
3. Loop Control Instruction
4. Case-Control Instruction

Chapter 3: Conditional Instructions

Sometimes we want to watch comedy videos on youtube if the day is Sunday. Sometimes we order junk food if it is our friend's birthday in the hostel. You might want to buy an umbrella if its raining and you have the money. You order the meal if dal or your favorite bhindi is listed on the menu.

All these are decisions that depend on conditions being met.

In 'C' language too, we must be able to execute instructions on a condition(s) being met.

Decision making instructions in C

- If-else statement
- Switch statement

If-else statement

The syntax of an if-else statement in c looks like:

```
if ( condition to be checked) {  
    Statements-if-condition-true ;  
}  
  
else{  
statements-if-condition-false ;  
}
```

Code Example

```
int a=23;  
if (a>18){  
printf("you can drive\n");  
}
```

Note that else block is not necessary but optional.

Relational Operators in C

Relational operators are used to evaluate conditions (true or false) inside the if statements. Some examples of relational operators are:

| | |
|----|--------------------------|
| == | equals to |
| >= | greater than or equal to |
| > | greater than |
| < | less than |
| <= | less than or equal to |
| != | not equal to |

Important Note: '=' is used for an assignment whereas '==' is used for an equality check.

The condition can be any valid expression. In C a non-zero value is considered to be true.

Logical Operators :

&&, ||, and ! are the three logical operators in C. These are read as “and”, ”or”, and “not”. They are used to provide logic to our c programs.

Use of logical operators:

1. && (AND) is true when both the conditions are true

“1 and 0” is evaluated as false

“0 and 0” is evaluated as false

“1 and 1” is evaluated as true

2. || (OR) is true when at least one of the conditions is true. (1 or 0 = 1)(1 or 1 = 1)

3. ! returns true if given false and false if given true.

!(3==3) evaluates to false

!(3>30) evaluates to true

As the number of conditions increases, the level of indentation increases. This reduces readability. Logical operators come to rescue in such cases.

Else if clause:

Instead of using multiple if statements, we can also use else if along with if thus forming an if-else if-else ladder.

```
if {  
    // statements ;  
}  
  
else if { //statements;  
    }  
  
else { //statements;  
    }
```

Using if-else if-else reduces indents. The last “else” is optional. Also, there can be any number of “else if”.

Last else is executed only if all conditions fail.

Operator Precedence

| Priority | Operator |
|-----------------|------------|
| 1 st | ! |
| 2 nd | *,/,% |
| 3 rd | +, - |
| 4 th | <>, <=, >= |
| 5 th | ==, != |
| 6 th | && |
| 7 th | |
| 8 th | = |

Conditional operators

A shorthand “if-else” can be written using conditional or ternary operators.

Condition ? expression-if-true ; expression-if-false

Here, '?' and ':' are Ternary operators.

Switch case-control instruction

Switch-case is used when we have to make a choice between the number of alternatives for a given variable.

Syntax,

Switch(integer-expression)

```
{  
Case c1:  
    Code;  
Case c2:                //c1,c2,c3 are constants  
    Code;                //Code is any valid C code  
Case c3:  
    Code;  
Default:  
    Code;  
}
```

The value of integer-expression is matched against c1,c2,c3.....if it matched any of these cases, that case along with all subsequent “case” and “default” statements are executed.

Quick Quiz: Write a program to find the grade of a student given his marks based on below:

90-100-> A

80-90-> B

70-80 ->C

60-70-> D

<70->F

Important notes

- We can use switch case statements even by writing in any order of our choice
- Char values are allowed as they can be easily evaluated to an integer
- A switch can occur within another but in practice, this is rarely done

Chapter 3- Practice Set

1. What will be the output of this program?

```
int a=10;

if(a=11)
    printf("I am 11");
else
    printf("I am not 11");
```

2. Write a program to find out whether a student is pass or fail; if it requires a total of 40% and at least 33% in each subject to pass. Assume 3 subjects and take marks as an input from the user.
3. Calculate income tax paid by an employee to the government as per the slabs mentioned below:

| Income Slab | Tax |
|-------------|-----|
| 2.5L-5.0L | 5% |
| 5.0L-10.0L | 20% |
| Above 10.0L | 30% |

Note that there is no tax below 2.5L. Take income amount as an input from the user.

4. Write a program to find whether a year entered by the user is a leap year or not. Take the year as input from the user.
5. Write a program to determine whether a character entered by the user is lowercase or not.
6. Write a program to find the greatest of four numbers entered by the user.

Chapter4 : Loop Control Instructions.

Why loops?

Sometimes we want our programs to execute few set of instructions over and over again.

For example, print first 1 to 100 numbers, print First 100 even number.

Hence loops make it easy for a programmer to tell computer that a given set of instructions must be executed repeatedly.

```
> Users > abhi7 > AppData > Local > Temp > Rar$Dia12188.26666 > C 04_01_loop_intro.c > ...
1  #include<stdio.h>
2
3  int main(){
4      printf("Hello ");
5      printf("world\n");
6      int a = 1;
7      // Loops are used to repeat similar part of a code snippet efficiently
8      // (
9      // printf("%d\n", a);
10     // a++;) ----> 100 times
11
12     return 0;
13 }
```

Types of loop:

Primarily there are three types of loops in C language:

1. While loop
2. Do while loop
3. For loop

We will look into these one by one.

1. While loop:

While(condition is true){

// Code

//code. => The block keeps executing as long as the condition is true.

}

Example

Int i=0;

While (i<10){

Printf("The value of i is %d",i);

i++;

}

NOTE :

If the condition never became false, the while loop keeps getting executed. Such a loop is known as an infinite loop.

Quick Quiz :

While a program to print natural numbers from 10 to 20 when initial loop control is integrated to 0.

The loop control need to be an integer, it can be float as well.

Increment and Decrement operators.

`i++`; -----> `i` is incremented by 1.

`i--`; -----> `i` is Decrement by 1.

`Printf("--i=%d", --i);`

This first decrement `i` then prints it.

`Printf("i--=%d", i--);`

This first prints '`i`' and then decrement it.

NOTE :

`+++` Operator does not exist.

`+=` Is a compound operator like `-=`, `*=`, `/=`, `%`

```
> Users > abhi7 > AppData > Local > Temp > Kar$Dla12188.35316 > C 04_02_while_loops.c > ...
1  #include<stdio.h>
2
3  int main(){
4      int a;
5      scanf("%d", &a);
6      while(a<10){
7          // a = 11;
8          // while(a>10){ ---> These two lines will lead to an infinite loop
9              printf("%d\n", a);
10             a++;
11         }
12
13         return 0;
14     }
```

2. Do While Loop

The syntax of do while loop looks like this

`do{`

`//Code`

`//Code`

`}While(condition)`

Do while loops work very similar like while loop.

While-->Check the condition and then execute the code.

Do-While-->Execute the code then check the condition.

Quick Quiz-

Write a program to print first n natural numbers using do while loop

Hints:-

Input- 4

Output- 1
2
3
4

```
#include<stdio.h>

int main(){
    int i=0;
    int n;
    printf("Enter the value of n\n");
    scanf("%d", &n);

    do{
        printf("The number is %d \n", i+1);
        i++;
    }while(i<n);
    return 0;
}
```

3.For loop

The syntax of "For loop" looks like this:

for(initialize ; test ; increment or decrement)

```
{
//Code
//Code
}
```

Initialize- Setting a loop counter to an initial value

Test- Checking a condition

Increment or decrement - Updating the loop counter.

An example:

```
for(i=0;i<3;i++){
    Printf("%d",&i);
    Printf ("\n");
}
```

Output: 0

1
2

Quick Quiz-: Write a program to print first n natural numbers using for loop.

A case of documenting for loop

```
for(i=5 ; i ; i--)  
Printf("%d"\n,i);
```

This for loop will keep on running until i becomes 0.

The loop runs in following steps:

1. i is initialised to 5.
2. The condition " i " is tested.
3. The code is executed.
4. i is decremented.
5. Condition i is checked and code is executed if it's not 0.
6. & So on until i is non 0.

Quick Quiz-:Write a program to print n natural numbers in reverse order.

The break statement in C

The break statement is used to exit the loop irrespective of whether the condition is true or false.

Whenever a " break " is encountered inside the loop the control is sent outside the loop.

EXAMPLE

```
for(i=0; i<=1000; i++);  
Printf ("%d\n",i);  
If(i==5)  
{  
Break;  
}  
}
```

OUTPUT:-

```
0  
1  
2  
3  
4  
5
```

The continued statement in C

The continue statement is used immediately move to the next iteration of the loop.

The control is taken to the next iteration thus skipping everything bellow "Continue" inside the loop for that iteration.

Let us look at an example;

```
int skip=5;
int i = 0;
While (i<10){
If(i!=skip)
continue;
Else{
Printf ("%d",i);
}
```

OUTPUT--> 5
and not 0....9

NOTE :

1. Some time the name of the variable might not indicate the behaviour of the program.
2. Break statement completely exits the loop.
3. Continue statement skips the particular iteration of the loop.

Chapter 4 - Practice Set

1. Write a program to print the multiplication table of a given number n.
2. Write a program to print multiplication table of 10 in reversed order.

```
> Users > abhi7 > AppData > Local > Temp > Rar$Dla10452.9030 > C 04_11_pr02
1  #include<stdio.h>
2
3  int main(){
4      printf("****Multiplication table of 10****\n\n");
5      for(int i=10;i;i--){
6          printf("10 X %d = %d\n", i, 10*i);
7      }
8      return 0;
9  }
```

3. A do-while loop is executed:
 - At least once
 - At least twice
 - At most once
4. What can be done using one type of loop can also be done using the other two types of loops – True or False.
5. Write a program to sum the first ten natural numbers using a while loop.

```
#include<stdio.h>

int main(){
    int i=1, sum=0, n=10;

    // for(i=1; i<=n; i++){
    //     sum +=i;
    // }
    while( i<=n){
        sum +=i;
        i++;
    }
    printf("The value of sum(1 to 10) is %d", sum);
    return 0;
}
```

6. Write a program to implement program 5 using for and do-while loop.

7. Write a program to calculate the sum of the numbers occurring in the multiplication table of 8.(Consider 8x1 to 8x10)

8. Write a program to calculate the factorial of a given number using for loop.

```
#include<stdio.h>

int main(){
    // factorial(4) - 1 * 2 * 3 * 4
    // factorial(6) - 1 * 2 * 3 * 4 * 5 * 6
    int i=0, n=7, factorial=1;
    for(i=1;i<=n;i++){
        factorial *=i;
    }
    printf("The value of factorial %d is %d", n, factorial);
    return 0;
}
```

9. Repeat 8 using a while loop.

10. Write a program to check whether a given number is prime or not using loops.

```
#include<stdio.h>

int main(){
    // Prime Numbers = A prime number (or a prime) is a natural number greater than 1 that is not a product of two smaller natural numbers.
    // Disclaimer: This is not the best method to solve this problem
    int n = 2, prime=1;
    for(int i=2;i<n;i++){
        if (n%i==0 ){
            prime = 0;
            break;
        }
    }
    if (prime==0){
        printf("This is not a prime number");
    }
    else{
        printf("This is a prime number");
    }
    return 0;
}
```

11. Implement 10 using other types of loops.

Project 1: Number Guessing Game

Problem: This is going to be fun!! We will write a program that generates a random number and asks the player to guess it. If the player's guess is higher than the actual number, the program displays "Lower number please". Similarly, if the user's guess is too low, the program prints "Higher number please".

When the user guesses the correct number, the program displays the number of guesses the player used to arrive at the number.

Hints:

- Use loops
- Use a random number generator.

CODE

```
#include<stdio.h>
#include<stdlib.h>
#include<time.h>

int main(){
    int number, guess, nguesses=1;
    srand(time(0));
    number = rand()%100 + 1; // Generates a random number between 1 and
100
    // printf("The number is %d\n", number);
    // Keep running the loop until the number is guessed
    do{
        printf("Guess the number between 1 to 100\n");
        scanf("%d", &guess);
        if(guess>number){
            printf("Lower number please!\n");
        }
        else if(guess<number){
            printf("Higher number please!\n");
        }
    }
```



```
    else{  
        printf("You guessed it in %d attempts\n", nguesses);  
    }  
    nguesses++;  
}while(guess!=number);  
  
return 0;  
}
```

Chapter 5: Function and Recursion

Sometimes our programs gets bigger in size and it's not possible for a programmer to track which code is doing what.

Function is a why to break our code into chunks so that it is possible for a programmer to reuse them.

What is Function ?

A function is a block of a code which performs a particular task.

A function is reused by the programmer in a given program any number of times.

Example and syntax of a function:

```
#Includes<stdio.h>
Void display ();           ==>Function prototype
int main(){
int a;
display ();               ==>Function call
return 0;
}
Void display (){           ==>Function defination.
Printf("Hi I am display\n");
}
```

Function Prototype

Function prototype is a way to tell the compiler about the function we are going to design in the program.

Here void indicates that the function returns nothing.

Function Call

Function call is a way to tell the compiler to execute the function body at the time the call is made.

NOTE that the program execution starts from the main function in the sequence the Instructions are written.

Function defination :

This part contains the exact set of instructions which are executed during the function call. When a function is called from main(), the main function falls asleep and gets temporary suspended. During this time, the control goes to the function being called .When the function body is done executing main() resumes.

Quick Quiz- Write a program with three function

1. Good morning function which prints " Good Morning".
2. Good afternoon function which prints " Good Afternoon".

3. Good night function which prints " Good Night".

Main() should call all of these in order 1->2->3

Important Points:

- Execution of a C program starts from main()
- A C program can have more than one function.
- Every function gets called directly or indirectly from main()
- There are two types of functions in C. Lets talk about them.

Types of function-

1. Library function : Commonly required functions group together in a library file on disk.
2. User defined function : This are the functions declared and defined by user.

Why use function ?

1. To avoid re-executing the same logic again and again.
2. To keep track of what we are doing in a program.
3. To test and check logic independently.

Passing Values to function

We can pass value to a function and can get a value in return of a function.

```
int sum(int a,int b)
```

The above prototype means that sum is a Function which takes values of 'a' and 'b' and returns the value of type int.

Function definition of sum can be :

```
int sum(int a ,int b){  
    int c;  
    c=a+b; ==>a and b are the perimeter  
    return C;  
}
```

Now we can call sum (2,3); from main to get 5 in return. ==>Here 2 & 3 are the arguments
int d=sum(2,3); ==>d becomes 5

NOTE:

1. Perimeters are the value or variable placeholders in the function definition.
2. Arguments are the actual values passed to the function to make a call.
3. A function can return only one values in a time.
4. If the passed variable is changed to inside the function call doesn't change the function values.

```
int change (int a){
a=77;.
return 0;
}
```

==>Misnomer

Change is a Function which changes 'a' to 77. Now if we call it from main function like this

```
int b=22;
Change (b);
Printf("b is %d\n",b);
```

The value of b remains 22. This happens because a copy of 'b' is passed to the change function.

Quick Quiz-

Use the library function to calculate the area of a square with side a.

Recursion :

A function defined in C can call itself. This is called recursion.

A function is calling itself is also called 'recursive' function.

Example of Recursion :

A very good example of Recursion is factorial.

Factorial (n)=1*2*3*.....*n

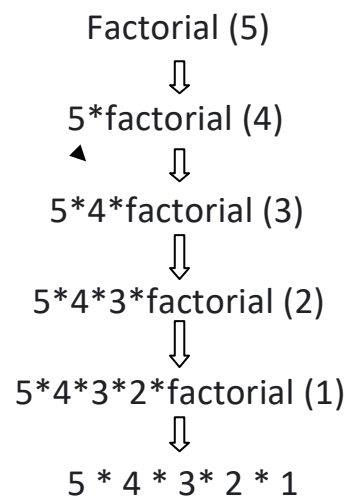
Factorial (n)=1*2*3*.....*(n-1)*n

Factorial (n)=factorial (n-1)*n

Since we can write factorial of a number is a term of itself,we can program it using recursion.

```
int factorial (int X){
int f;
if(X==0| |X==1)
return 1;
else
f=X*factorial (X-1);
return f;
}
```

How does it works ?



NOTE:

1. Recursion is sometime the most direct way to code a algorithm.
2. The condition which does not call the function further in a recursive function is called as the base function.
3. Sometimes due to a mistake made by the programmer,a recursive function can keep running resulting a memory error.

Chapter 6: Pointers

A pointer is a variable which stores the address of another variable.



j is a pointer.
j points to i.

The "address of"(&) operator.

The address of an operator is used to obtain the address of a given variable.

If you refer to the diagram above

&i=>87994

&j=>87998

Format specifier for printing pointer address is "%u".

The "value at address" operator (*)

The value at address or operator is used to obtain the above present a given memory address. It is denoted by '*'

*(&i)=72

*(&j)=87994

How to declare a pointer ?

A pointer is declared using the following syntax.

int *j; ==> Declared a variable j of type int-pointer.

j=&i; ==> Store the address of i in j.

Just like pointer of type integer, we also have pointers to char, float etc.

int *ch-ptr ; -> Pointer to integer

Char *ch-ptr ; -> pointer to character

float *ch-ptr ; -> Pointer to float

Although it's a good practice to use meaningful variable names, we should be very careful while reading and working on program from fellow programmers.

A program to demonstrate pointer:

INPUT:

```
#include<stdio.h>
int main( ){
int i=8;
int *j;
j=&i;
Printf ("Add i=%u\n",&i);
Printf ("Add i=%u\n",j);
Printf ("Add j=%u\n",&j);
Printf ("Value i=%d\n",&i);
Printf ("Value i=%d\n",*(&i));
Printf ("Value i=%d\n",*j);
return 0;
}
```

OUTPUT:

```
Add i=87994
Add i=87994
Add j=87998
Value i=8
Value i=8
Value i=8
```

The program sums it all. If you understand it,you have got the idea of Pointers.

Pointer to a Pointer

Just like j is pointing to ' i ' or storing the address of ' i ' ,we can have another variable 'k' which can further store the address of ' j ' what will be the type of 'k'

```
int **k;
k=&j;
```

We can even go further one level and create a variable l of type int *** to store the address of ' k '. We mostly use int ** sometimes in real world progress.

Types of Function calls

Bases on the way we pass arguments to the function, function calls are of two types.

1. Call by value ----->Sending the values of argument.
2. Call by reference ----->Sending the address of arguments.

Call by Value :

Here the value of the arguments are passed to the function.

Consider the example:

```
int C=sum(3,4);. ==>Assume, x=3 and y=4
```

If sum is defined as `sum(int a,int b)`, the value 3 and 4 are copied to 'a' and 'b'. Now even we change the variable nothing happened to the variable 'x' and 'y'.

This is call by value.

In C we usually make a call by value.

Call by reference :

Here the address of the variable is passed to the function as arguments.

Now since the address are passed to the function, the function can now modify the value of a variable in calling function using '*' and '&'.

```
Void swap(int*x,int*y){  
    int temp;  
    temp=*x;  
    *x=*y;  
    *y=temp;  
}
```

The function is capable of swapping the values passed to it. If `a=3` and `b=4` before a call to `Swap(a,b)`, `a=4` and `b=3` after calling `Swap`.

```
Int main(){  
    int a=3;  
    int b=4;  
    Swap(a,b)  
    return 0;  
}
```


Chapter 6 - Practice Set

1. Write a program to print the address of a variable. Use this address to get the value of this variable.
2. Write a program having a variable `i`. Print the address of `i`. Pass this variable to a function and print its address. Are these addresses same? Why?
3. Write a program to change the value of a variable to ten times its current value. Write a function and pass the value by reference.
4. Write a program using a function that calculates the sum and average of two numbers. Use pointers and print the values of sum and average in `main()`.
5. Write a program to print the value of a variable `i` by using "pointer to pointer" type of variable.
6. Try problem 3 using call by value and verify that it doesn't change the value of the said variable.

Chapter 7-Arrays

An array is a collection of similar element.

One variable => Capable of storing multiple Values

Syntax: The syntax of declaring an Array looks like this-

int marks[90]; =>Integer array

char name[20];. =>Character array or string

float percentile[90];. =>Float array

The value can now be assigned to an array like this :

marks[0]=33;

marks[1]=12;

NOTE:

It is very important to NOTE that the array index starts with 0.

Accessing elements

Elements of an array can be accessed using:

scanf ("%d",& marks[0]);. =>Input first value

Printf("%d",marks[0]);. =>Output first value of the array.

Quick Quiz-

Write a program to accept marks of 5 students in an array and print them in the screen.

Initialization of an array

There are many other ways in which an array can be initialized.

int cgpa[3]={9,8,8} => Arrays can be initialised while declaration

float marks[]={33,40}

Arrays in memory

Consider this array:

int arr[3]={1,2,3} => 1 integer = 4 bytes

This will reserve 4x3=12 bytes in memory. 4 bytes for each integer.

| | | |
|---|---|---|
| 1 | 2 | 3 |
|---|---|---|

62302

62306

62310

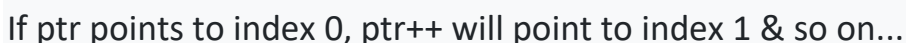
{Array in Memory}

A pointer can be incremented to point to the next memory location of that type.

Following operations can be performed on pointers:

- Quick Quiz:** Try these operations on another variable by creating pointers in a separate program. Demonstrate all the four operations.

Consider this array,



This way we can have an integer pointer pointing to the first element of the array like this:

```
int * ptr = & arr[0];    => or simply arr
```

```
ptr++;
```

*ptr => will have 9 as it's value

Passing arrays to functions

Arrays can be passed to the functions like this

```
printArray(arr,n);          => function call
```

```
void printarray(int *i,int n);    => function prototype
```

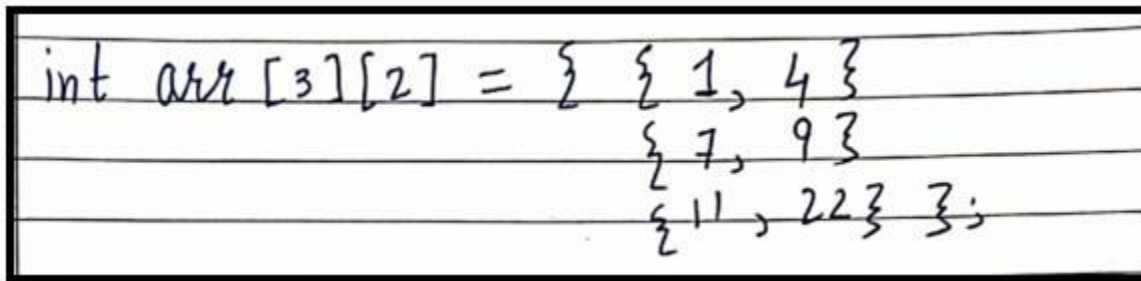
or

```
void printarray(int i[] ,int n);
```

Multidimensional arrays

An array can be of 2 dimension / 3 dimension / n dimensions.

A 2-dimensional array can be defined as:



```
int arr [3][2] = { { 1, 4 }  
                  { 7, 9 }  
                  { 11, 22 } };
```

We can access the elements of this array as arr [0] [0], arr [0] [1] & so on...

At arr [0] [0] value would be 1 and at arr [0] [1] value would be 4.

2-D arrays in Memory

A 2-d array like a 1-d array is stored in contiguous memory blocks like this:

| | | | | | |
|-------------------------|-------|-----|---|----|----|
| arr[0][0] arr[0][1] ... | | | | | |
| 1 | 4 | 7 | 9 | 11 | 22 |
| 87224 | 87228 | ... | | | |

Quick Quiz: Create a 2-d array by taking input from the user. Write a display function to print the content of this 2-d array on the screen.

Chapter 7- Practice Set

1. Create an array of 10 numbers. Verify using pointer arithmetic that $(ptr+2)$ points to the third element where ptr is a pointer pointing to the first element of the array.
2. If $S[3]$ is a 1-D array of integers then $*(S+3)$ refers to the third element:
 - True
 - False
 - Depends
3. Write a program to create an array of 10 integers and store a multiplication table of 5 in it.
4. Repeat problem 3 for a general input provided by the user using `scanf()`
5. Write a program containing a function that reverses the array passed to it.
6. Write a program containing functions that counts the number of positive integers in an array.
7. Create an array of size 3×10 containing multiplication tables of the numbers 2, 7 and 9 respectively.
8. Repeat problem 7 for a custom input given by the user.
9. Create a three-dimensional array and print the address of its elements in increasing order.

Chapter 8 – Strings

A string is a 1-d character array terminated by a null('\0') => {this is null character}

The null character is used to denote string termination, characters are stored in contiguous memory locations.

Initializing Strings

Since string is an array of characters, it can be initialized as follows:

```
char s[]={'A','B','H','I','\0'}
```

There is another shortcut for initializing strings in C language:

```
char s[]="ABHI"; => In this case C adds a null character automatically.
```

Strings in memory

A string is stored just like an array in the memory as shown below

Quick Quiz: Create a string using " " and print its content using a loop.

Printing Strings

A string can be printed character by character using printf and %c.

But there is another convenient way to print strings in C.

```
char st[] = "ABHI";
```

```
printf("%s",st); => prints the entire string
```

Taking string input from the user

We can use %s with scanf to take string input from the user:

```
char st[50];
```

```
scanf("%s",&st);
```

scanf automatically adds the null character when the enter key is pressed.

NOTE:

1. The string should be short enough to fit into the array.
2. scanf cannot be used to input multi-word strings with spaces.

gets() and puts()

gets() is a function that can be used to receive a multi-word string.

```
char st[30];
```

gets(st); => the entered string is stored in st!

Multiple gets() calls will be needed for multiple strings.

Likewise, puts can be used to output a string.

puts(st); => Prints the string and places the cursor on the next line

Declaring a string using pointers

We can declare strings using pointers

```
char *ptr= "ABHI";
```

This tells the compilers to store the string in the memory and the assigned address is stored in a char pointer.

NOTE:

1. Once a string is defined using `char st[] = "ABHI"`, it cannot be initialized to something else.
2. A string defined using pointers can be reinitialized. => `ptr="rohan"`;

Standard library functions for Strings

C provides a set of standard library functions for strings manipulation.

Some of the most commonly used string functions are:

strlen() - This function is used to count the number of characters in the string excluding the null ('\0') character.

```
int length=strlen(st);
```

These functions are declared under `<string.h>` header file.

strcpy() - This function is used to copy the content of second string into first string passed to it.

```
char source[ ]= "ABHI";
```

```
char target[30];
```

```
strcpy(target,source); => target now contains "ABHI"
```

Target string should have enough capacity to store the source string.

strcat() - This function is used to concatenate two strings

```
char s1[11]= "Hello";
```

```
char s2[ ]= "ABHI";
```

```
strcat(s1,s2);          => s1 now contains "Hello ABHI" <No space in between>
```

strcmp() - This function is used to compare two strings. It returns: 0 if strings are equal

Negative value if first string's mismatching character's ASCII value is not greater than second string's corresponding mismatching character. It returns positive values otherwise.

```
strcmp("For", "Joke");          => positive value
```

```
strcmp("Joke", "For");          => Negative value
```

Chapter 8 - Practice Set

1. Which of the following is used to appropriately read a multi-word string-
 - Gets()
 - Puts()
 - Printf()
 - Scanf()
2. Write a program to take a string as an input from the user using %c and %s. Confirm that the strings are equal.
3. Write your own version of strlen function from <string.h>
4. Write a function slice() to slice a string. It should change the original string such that it is now the sliced strings. Take m and n as the start and ending position for slice.
5. Write your own version of strcpy function from <string.h>
6. Write a program to encrypt a string by adding 1 to the ASCII value of its characters.
7. Write a program to decrypt the string encrypted using the encrypt function in problem 6.
8. Write a program to count the occurrence of a given character in a string.
9. Write a program to check whether a given character is present in a string or not.

Chapter 9 - Structures

Arrays and Strings => Similar data (int, float, char)

Structures can hold => dissimilar data

Syntax for creating Structures

A C Structure can be created as follows:

```
struct employee{  
  
int code;           => this declares a new user-defined datatype  
  
float salary;  
  
char name[10];  
  
};           (semicolon is important)
```

We can use this user-defined data type as follows:

```
struct employee e1;           // creating a structure variable  
strcpy(e1.name,"abhi");  
e1.code=100;  
e1.salary=71.22;
```

So a structure in c is a collection of variables of different types under a single name.

Quick Quiz: Write a program to store the details of 3 employees from user-defined data. Use the structure declared above.

Why use structures?

We can create the data types in the employee structure separately but when the number of properties in a structure increases, it becomes difficult for us to create data variables without structures. In a nutshell:

1. Structures keep the data organized.
2. Structures make data management easy for the programmer.

Array of Structures

Just like an array of integers, an array of floats, and an array of characters, we can create an array of structures.

```
struct employee facebook[100];    =>an array of structures
```

We can access the data using:

```
facebook[0].code=100;
```

```
facebook[1].code=101;
```

.....and so on.

Initializing structures

Structures can also be initialized as follows:

```
struct employee abhi={100,71.22,"Abhi"};
```

```
struct employee shubh={0};           // All the elements set to 0
```

Structures in memory

Structures are stored in contiguous memory locations for the structures e1 of type struct employee, memory layout looks like this:

In an array of structures, these employee instances are stored adjacent to each other.

Pointer to structures

A pointer to the structure can be created as follows:

```
struct employee *ptr;
```

```
ptr=&e1;
```

Now we can print structure elements using:

```
printf("%d",*(ptr).code);
```

Arrow operator

Instead of writing *(ptr).code, we can use an arrow operator to access structure properties as follows

*(ptr).code or ptr->code

Here -> is known as an arrow operator.

Passing Structure to a function

A structure can be passed to a function just like any other data type.

`void show(struct employee e);` =>Function prototype

Quick Quiz: Complete this show function to display the content of employee.

Typedef keyword

We can use the typedef keyword to create an alias name for data types in c.

typedef is more commonly used with structures.

```
struct complex{
    float real;                // struct complex c1,c2; for defining
    complex numbers
    float img;
};

typedef struct complex{
    float real;
    float img;                // ComplexNo c1,c2; for defining complex
    numbers
}ComplexNo;
```

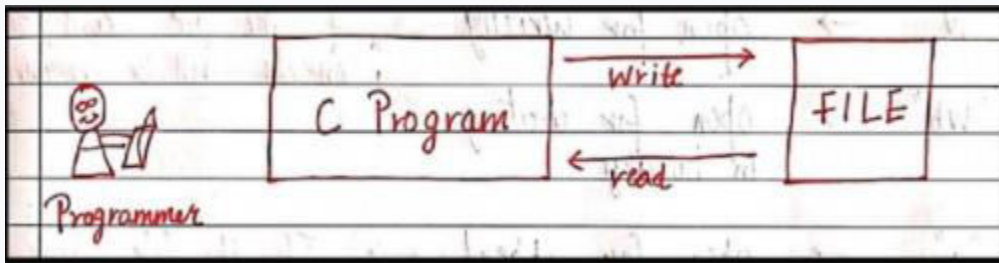
Chapter 9- Practice Set

1. Create a two-dimensional vector using structures in C.
2. Write a function SumVector which returns the sum of two vectors passed to it. The vectors must be two-dimensional.
3. Twenty integers are to be stored in memory. What will you prefer- Array or Structure?
4. Write a program to illustrate the use of an arrow operator -> in C.
5. Write a program with a structure representing a Complex number.
6. Create an array of 5 complex numbers created in problem 5 and display them with the help of a display function. The values must be taken as an input from the user.
7. Write problem 5's structure using typedef keyword.
8. Create a structure representing a bank account of a customer. What fields did you use and why?
9. Write a structure capable of storing date. Write a function to compare those dates.
10. Solve problem 9 for time using typedef keyword.

Chapter 10 - File I/O

The random access memory is volatile and its content is lost once the program terminates. In order to persist the data forever, we use files.

A file data stored in a storage device. A C program can talk to the file by reading content from it and writing content to it.



File pointer

The “File” is a structure that needs to be created for opening the file. A file pointer is a pointer to this structure of the file.

File pointer is needed for communication between the file and the program.

A file pointer can be created as follows:

```
FILE *ptr;
ptr=fopen("filename.ext","mode");
```

File opening modes in C

C offers the programmers to select a mode for opening a file. Following modes are primarily used in c File I/O

| | | |
|------|------------------------------|--|
| "r" | → open for reading | → If the file does not exist, fopen returns NULL |
| "rb" | → open for reading in binary | → |
| "w" | → open for writing | → If the file exists, the contents will be overwritten |
| "wb" | → open for writing in binary | → |
| "a" | → open for append | → If the file does not exist, it will be created |

Types of Files

There are two types of files:

1. Text files(.txt, .c)
2. Binary files(.jpg, .dat)

Reading a file

A file can be opened for reading as follows:

```
FILE *ptr;  
ptr=fopen("Abhi.txt","r");  
int num;
```

Let us assume that "Abhi.txt" contains an integer

We can read that integer using:

```
fscanf(ptr,"%d",&num);
```

=> fscanf is file counterpart of scanf

This will read an integer from the file in the num variable.

Quick Quiz: Modify the program above to check whether the file exists or not before opening the file.

Closing the file

It is very important to close file after read or write. This is achieved using fclose as follows:

```
fclose(ptr);
```

This will tell the compiler that we are done working with this file and the associated resources could be freed.

Writing to a file

We can write to a file in a very similar manner as we read the file

```
FILE *fptr;  
fptr=fopen("Abhi.txt","w");  
int num=432;
```



```
fprintf(fptr,"%d",num);  
fclose(fptr);
```

fgetc() and fputc()

fgetc and fputc are used to read and write a character from/to a file.

fgetc(ptr); => Used to read a character from file

fputc('c',ptr); => Used to write character 'c' to the file

EOF: End of File

fgetc returns EOF when all the characters from a file have read. So we can write a check like below to detect the end of file.

```
while(1){  
ch=fgetc(ptr);           // When all the content of a file has been read,  
break the loop  
if(ch==EOF){  
break;  
}  
//code  
}
```

Chapter 10 - Practice Set

1. Write a program to read three integers from a file.
2. Write a program to generate a multiplication table of a given number in text format. Make sure that the file is readable and well-formatted.
3. Write a program to read a text file character by character and write its content twice in a separate file.
4. Take name and salary of two employees as input from the user and write them to a text file in the following format:

name1, 3300

name2, 7700

5. Write a program to modify a file containing an integer to double its value.

If old value = 2, then new file value = 4

Project 2: Snake, Water, Gun

Snake, Water, Gun or Rock, Paper, Scissors is a game most of us have played during school time. (I sometimes play it even now :))

Write a C program capable of playing this game with you.

Your program should be able to print the result after you choose Snake/Water or Gun.

Code:

```
#include<stdio.h>
#include<stdlib.h>
#include<time.h>

int snakeWaterGun(char you, char comp){
    // returns 1 if you win, -1 if you lose and 0 if draw
    // Condition for draw
    // Cases covered:
    // ss
    // gg
    // ww
    if(you == comp){
        return 0;
    }

    // Non-draw conditions
    // Cases covered:
    // sg
    // gs
    // sw
    // ws
    // gw
    // wg
```

```

    if(you=='s' && comp=='g'){
        return -1;
    }
    else if(you=='g' && comp=='s'){
        return 1;
    }

    if(you=='s' && comp=='w'){
        return 1;
    }
    else if(you=='w' && comp=='s'){
        return -1;
    }

    if(you=='g' && comp=='w'){
        return -1;
    }
    else if(you=='w' && comp=='g'){
        return 1;
    }
}

int main(){
    char you, comp;
    srand(time(0));
    int number = rand()%100 + 1;

    if(number<33){
        comp = 's';
    }
    else if(number>33 && number<66){
        comp='w';
    }
}

```

```
else{
    comp='g';
}

printf("Enter 's' for snake, 'w' for water and 'g' for gun\n");
scanf("%c", &you);
int result = snakeWaterGun(you, comp);
if(result ==0){
    printf("Game draw!\n");
}
else if(result==1){
    printf("You win!\n");
}
else{
    printf("You Lose!\n");
}
printf("You chose %c and computer chose %c. ", you, comp);
return 0;
}
```

Chapter 11 - Dynamic Memory Allocation

C is a language with some fixed rules of programming. For example: changing the size of an array is not allowed.

Dynamic Memory Allocation:

Dynamic memory allocation is a way to allocate memory to a data structure during the runtime we can use DMA function available in C to allocate and free memory during runtime.

Function for DMA in C

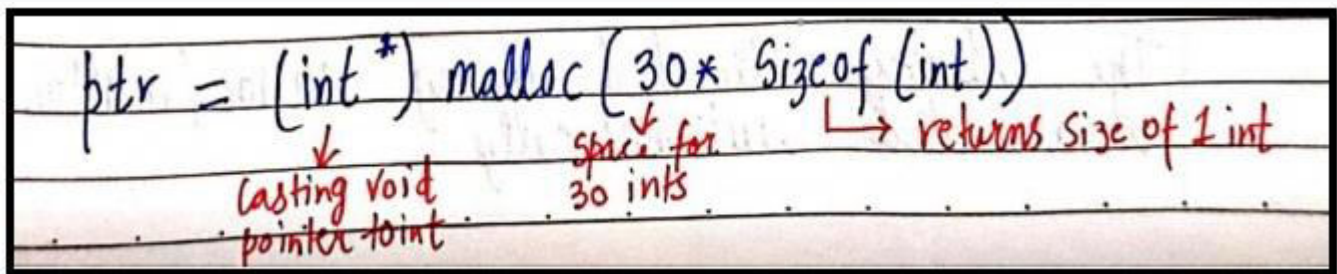
Following functions are available in C to perform dynamic memory allocation:

1. malloc()
2. calloc()
3. free()
4. realloc()

malloc() function

Malloc stands for memory allocation. It takes number of bytes to be allocated as an input and returns a pointer of type void.

Syntax:



The image shows a handwritten example of the malloc function syntax on lined paper. The code is: `ptr = (int *) malloc(30 * sizeof(int))`. Red annotations explain the parts: an arrow points from `(int *)` to the text "casting void pointer to int"; another arrow points from `30` to the text "space for 30 ints"; and a third arrow points from `sizeof(int)` to the text "returns size of 1 int".

The expression returns a NULL pointer if the memory cannot be allocated.

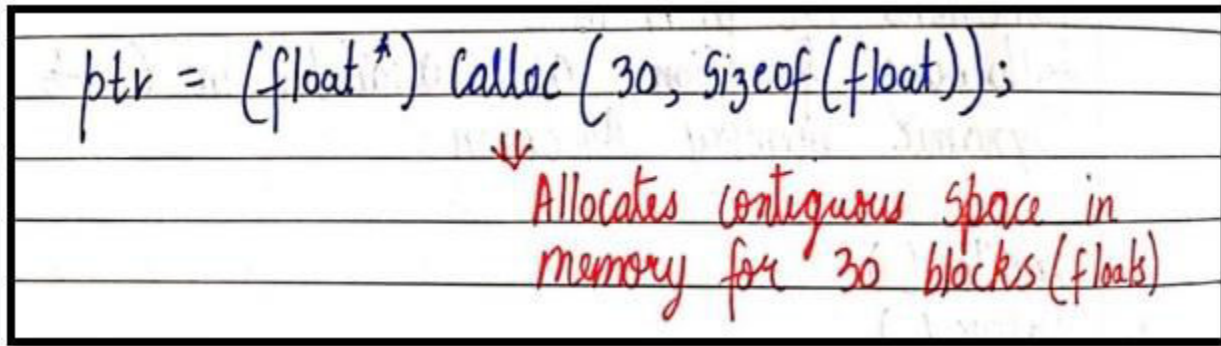
Quick Quiz: Write a program to create a dynamic array of 5 floats using malloc().

calloc() function

calloc stands for continuous allocation.

It initializes each memory block with a default value of 0.

Syntax:



```
ptr = (float *) calloc(30, sizeof(float));
```

↓
Allocates contiguous space in memory for 30 blocks (floats)

If the space is not sufficient, memory allocation fails and a NULL pointer is returned.

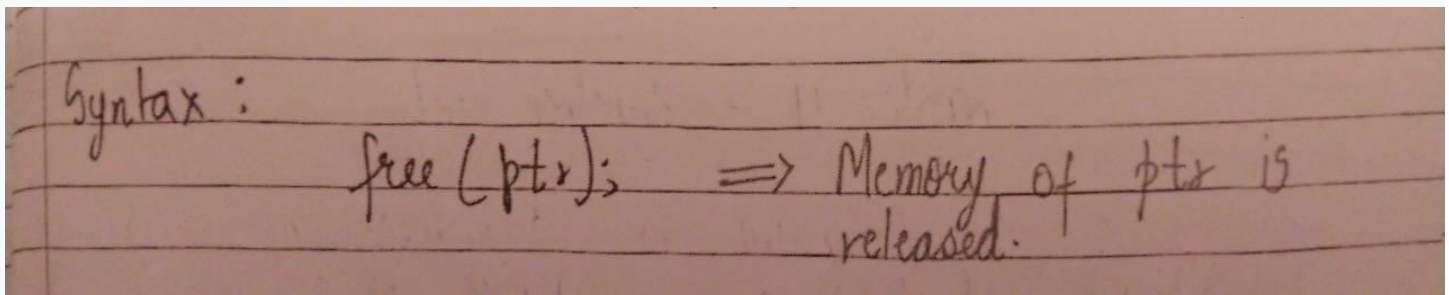
Quick Quiz: Write a program to create an array of size n using calloc() where n is an integer entered by the user.

free() function

We can use free() function to allocate the memory.

The memory allocated using calloc/malloc is not deallocated automatically.

Syntax:



```
free(ptr);
```

⇒ Memory of ptr is released.

Quick Quiz: Write a program to demonstrate the usage of free() with malloc().

realloc() function

Sometimes the dynamically allocated memory is insufficient or more than required. realloc is used to allocate memory of new size using the previous pointer and size.

Syntax:

```
ptr = realloc(ptr, newSize);  
ptr = realloc(ptr, 3 * sizeof(int));
```

⇓

ptr now points to this new block of memory capable of storing 3 integers.

Chapter 11 - Practice Set

1. Write a program to dynamically create an array of size 6 capable of storing 6 integers.
2. Use the array in Problem 1 to store 6 integers entered by the user.
3. Solve problem 1 using calloc().
4. Create an array dynamically capable of storing 5 integers. Now use realloc so that it can now store 10 integers.
5. Create an array of the multiplication table of 7 upto 10 ($7 \times 10 = 70$). Use realloc to make it store 15 numbers (from 7×1 to 7×15).
6. Attempt problem 4 using calloc().