Assignment 1

1MV18EC057
Tejas

1) What is constant? What are different types of constants?

→ C allowes you to define certain entities in a program whose values do not change during the course of programs execution. Such entities are known as constants.

Various types of constants:

1) Integer constant -
   * Represents the Integer values.

2) Real constants
   * Represent fractional or floating point values

3) Character constants
   * Represent single characters enclosed within single quotes (' ')

4) String constant
   Represents a string of characters enclosed within double quotes (" ").

You can define constants in any following way:

(i) #define statement: You can use this preprocessor directive at the beginning of the program to define the name and value of constant

P.T.O.

Tejas

2) ~~Explain about User defined data types.~~

The directive replaces the constant by its value at each of its occurance in the program. Such constant is also reffered as symbolic constant.

Syntax : #define <const_name> <const val>

Eg :- #define g 9.8.

2) Const keyword : You can use const keyword to define a constant in a program and assign at a value.

Syntax : const datatype <const_name> <const_value>

Eg : const float g 9.8.

3) Enumerate datatype : You can use enum keyword to define a constant as an enumerated datatype and assign at a set of constant values.

Syntax : enum <const_name> <set_of_value>

Eg :- enum option {yes, no};

2) Explain about user defined datatypes.

→ * User defined datatypes represents the user-specified identifiers used in the place of existing datatypes.

For example:

```
typedef float temperature;
temperature t1, t2, t3;
```

Here temperature is user-defined datatypes used to declare float type variable.

* The enumerated data type is also another instance of user defined datatypes that allows to declare variable that can be assigned only one of the pre-specified values. These values are enclosed within a pair of brace at this time of variable declarations.

For example:

```
enum logic {HIGH, LOW};
enum logic l;
l = HIGH;
```

3) What is type declaration? What is the use of declaration of variables?

→ Type declaration is a statement specifies the type, length and attributes of object and functions. Variable declaration tells the compiler where and how much storage to create for the variable.

PTO.

(4)

* A variable declaration specifies assurance to compiler that there exists a variable with the given type and name so the compiler can proceed for further compilation without requiring the complete detail about the variable.

* A variable definition has its meaning at the time of compilation only, the compiler needs actual variable definition at the time of linking program.

* A variable declaration is useful when you are using multiple file and you define your variable in one of the files which will be available at the time of linking of the program.

4) What is an operator? also Explain what are the classifications of operator.

→ Operator are the symbols that are used to perform arithmetic and logical computations on operands. An operand can be integer, floating point or character variable; or it can be a constant. Operator and operand towards constitutes an Expression.

Types of operator.

1) Arithmetic
2) Assignment
3) Bitwise.
4) Conditional
5) Increment and Decrement
6) Relational
7) Logical
8) Special

(3) (i) Based on number of operands required, the operators can be classified as.

1) Unary operators
2) Binary operators.

* Unary operators:

* Unary operators are the operators that operates or works only with ~~sig~~ single operands.

Eg:- a++, ++a, a--, --a, -a, etc.

Binary operators.

* Binary operators are operators which is applied for two or more operands.

Eg:- a+b; 5+10, a+5+10. etc.

(ii) Based on type of operator.

(1) Arithematic operator.

→ Arithematic operators are operators which is used to operate the arethematic problems. It consists of Addition (+), Subtraction (-), multiplication (*), Division (/), Modulos (%).

* If an operator has multiple operators with same precedence, then they are executed in the order of their appearance from left to right.

PTO.

IMN 18E0057
Tejas

(ii) Assignment Operator :

→ These operators are used to assign the value of an expression, variable or constant, (Equal to (=))

→ These operators are always Binary operator.

a) Normal Assignment (=)

Assign value of expression, variable/constant to variable.

Eg :- a=b , b=10

(b) Shorthand Assignment (+=, -=):

To assign value of an expression, variable or constant to a variable and perform the specified arithematic operation on that variable.

Eg :- b+=a.

(iii) Bitwise Operators :

Bitwise Operators are used for manipulating the integers at bit level. The binary representation of the integer gets altered as per the type of binary operator applied.

a) AND (&) → To perform logical AND Eg :- a&b

b) OR (|) → To perform logical OR Eg :- ~~a+b~~

Eg :- 0101 | 0011 = 0111

(c) Ex-OR(^): To perform logical exclusive OR operations at a bit level.

Eg :- 0101 ^ 0011 = 0110.

(d) Shift left (<<) : To shift the bits left (Unary)

00010111 << 1 = 00101110

e) Right shift (>>) : To shift the bits right (Unary)

00010111 >> 1 = 00001011

f) Complement (or) Not (~) : To negate each bit of binary number (Unary)

Eg :- ~0001 = 1110.

(iv) Conditional Operator

Conditional operators (?:) are only ternary operator supported by C. It requires three different expressions

<condition-expres> ? <Exp1> : <Exp2>

If 'condition-expres' evaluates to TRUE, then 'Expression 1' is evaluated else 'Expres2' is evaluated.

Eg : c = (a+b > 10) ? a : b where a = 5, b = 10

∴ c = a as a+b = 5+10 = 15 > 10 ✓ (TRUE)

Now a = 2 b = 3.

c = b, as a+b = 2+3 = 5 > 10 ✗ (FALSE)

P10.

(v) Increment and Decrement operators.

* Increment & decrement operator may be used prefix or a postfix as an expression.
* When used as prefix (example ++a or --b), the operator first changes the variable's value before the corresponding expression is evaluated.

Eg :- b = ++a ;

assume a = 5

b = 6 & a = 6 (After above expression is evaluated)

b = a++ assume a = 5 ;

b = 5 and a = 6 ;

(vi) Logical operators.

The outputs of an expression contain Logical operators is either '0' or '1'. The output '1' signifies a true condition while '0' signifies a false condition.

a) AND (&&) → return true if conditions are true

Eg :- a<20 && b>6

b) OR (||) → return true if atleast one of the conditions is true.

Eg :- a<20 || b>6 ;

c) NOT (!) → (unary) to negate the value of relational operator.

Eg !a<20

INNOVECOS 7
Tejas.

(VII) Relational Operators.

The output of relational operator is either 1 or 0. The output of 1 signifies a true while 0 signifies false.

(i) Less than (<)
(ii) Less than or equal to (<=)
(iii) greater than (>)
(iv) greater than or equal to (>=)
(v) Equal to (==)
(vi) Not equal to (!=)

(VIII) Special Operators.

a) Comma (,)

To link the related expressions together.

Eg: x = (a=10, b=5, c=a-b)

* Comma also used inside while and for loop.

b) Size of ();

* To retrieve the number of bytes a variable or a constant occupies in memory. (Unary)

Eg: x = sizeof (a) will assign the size of variable a to x.

c) Pointer operators: (&, *) (Unary)

Works with pointers.

Eg: &a retrieves the address of variable.
*a retrieves the value referenced by pointer p.

MVNRECOS T. Teja

5) Explain the conditional & unconditional branch statement & what are the various type.

→ C supports two types of decision control statement that can alter the flow of a sequence of instructions. These include conditional type branching and unconditional type branching.

(i) Conditional branch statement

* If is used to make decision based on the condition.
* Conditional statement executes sequentially when there is no condition around the statement.
* The following conditional statement are.

a) if statement :

* It is a one of powerful conditional statement.
* It is responsible for modifying the flow of execution of program.
* If statement is always used with condition
* The condition is evaluated first before executing any statement inside the body of 'if'.

Syntax

```
if (condition){
    instruction;}
```

Condition evaluates either true or false.
True is always non-zero value & false is the value contains zero.

1NN18EC057
Tejas

Eg :-

```
#include<stdio.h>
int main() {
        int num1 = 1;
        int num2 = 2;
        if (num1 < num2) {
                printf("num1 is smaller than num2");
        }
        return 0;
}
```

output

num1 is smaller than num2.

* In the above program, we have initialized two variables num1, and num2 with values 1, 2 respectively.
* Then, we have used if with a test-expression to check which number is smallest and which number is largest. Since value of num1 is smaller than num2 the condition evaluates to true and executes inside the conditional statement.
the conditional statement.
* The result will be print inside the block of 'if'.

(b) If-else statement
* If is also the conditional statement which is similar to if but the difference is the ~~there are~~ if the condition in if statement evaluates to false, it executes the else block.

PTO.

Syntax :

```
if (test exp)
  {
    True block statement
  }
else
  {
    False block statements
  }
```

(iii) Unconditional branch statement.

Unconditional branch statement is a statement which forces the execution of program to jump to another part of the program.

* "goto" statements (unconditional branch statements) to branch jump to a label in a program.
* break statement is also unconditional branch statement which breaks the iterations of the loop.
* continue statements is also unconditional branch statement which stops and starts the new iterations of the looping statem

1NV18EC057
Tejas

6) What is switch statement and also explain syntax and flowchart?

-> * switch statement is a multiway selection structure that matches the given expression or variable value with one of a number of integer values, and upon successful match branches to the corresponding statement block.

* A default value is also specified with the switch statement take appropriate action in case there is a complete mismatch.

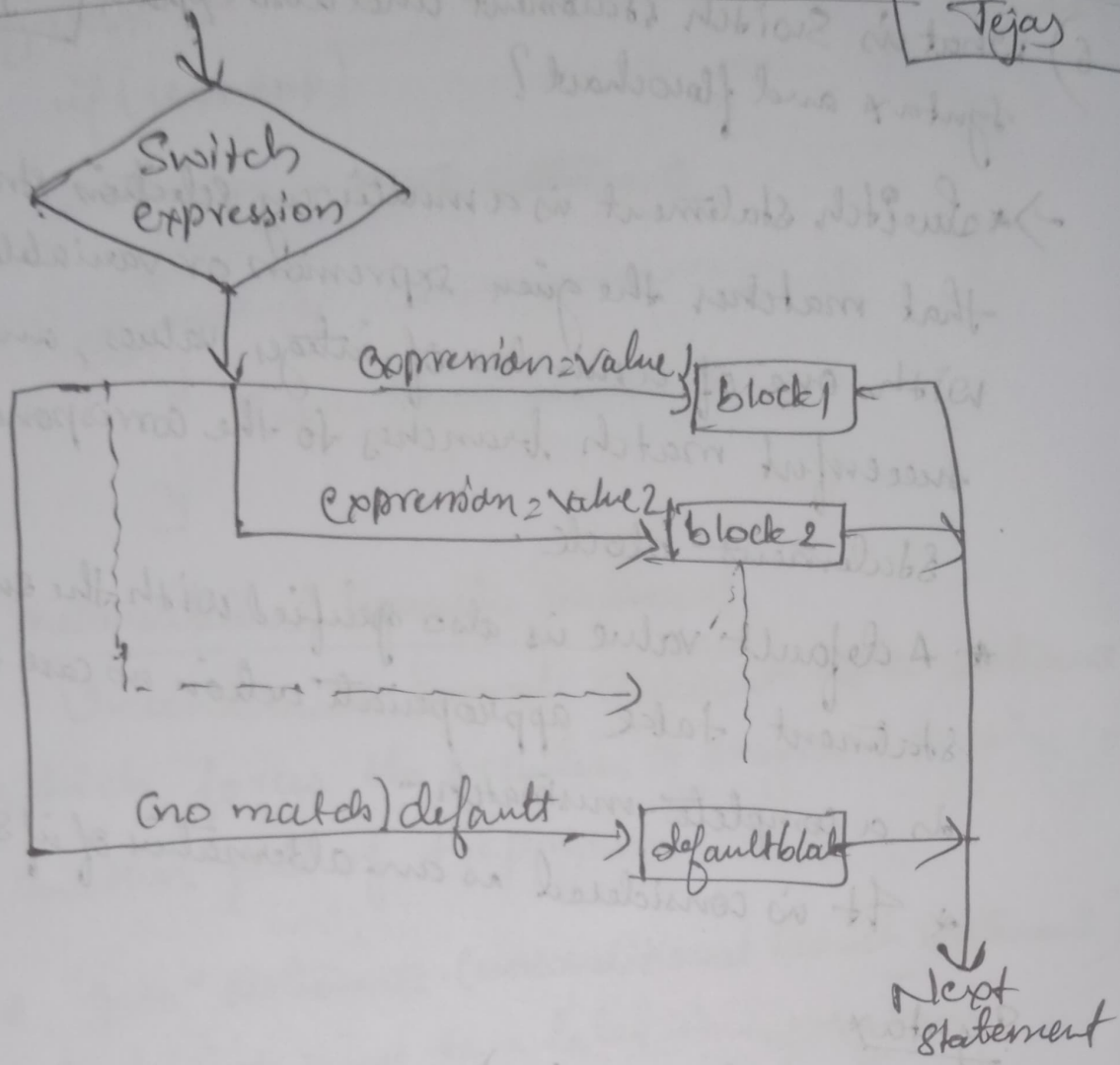
* It is considered as an alternative of if statement

Syntax

```
switch (expression)
{
    case value1 :
        <statement block>
        break;
    case value2 :
        <statement block>
        break;
    :
    case valuen : <statement block>
        break;
    default :
        <statement block>
}
```

PTO.

1MV18EC057
Tejas



7) Define arrays and also explain its types?

→ Arrays is a linear data structure that groups elements of similar types and stores them at contiguous memory locations.

Types of array.

(i) Single dimension array

(ii) Multidimension array

INVISECOSZ
Tejas.

(1) Single-Dimension Array :-

* Represents elements of the array in a single column.

* Syntax :

```
<data_type> <array_name>[size];
```

Example

```
int A[10];
```

* Array Initialization

Syntax :

```
<data_type> <array_name>[size] = {exp1, ...};
<array_name>[index_number] = <element>;
```

Example

```
int A[3] = {2,4,8};
A[0] = 2;
```

* Array size (in bytes)

array size = length of array * size of datatype.

Example :-

```
int A[4] = {10, 20, 30, 40};
```

Array size = 4 * 2 = 8 bytes

* Array representation

A[i]	A[0]	A[1]	A[2]	A[3]
Memory address →	10	20	30	40
	b	b+2	b+4	b+6

P.T.O.

(ii) Multi Dimensional Array :

* Array Declaration

Syntax

<data_type> <arr_name> [row] [column];

Example :

int A[2][2];

* Array Initialization

Syntax

<array_name> [row] [column] = <element>;

<data_type> <arr_name> [row] [column] = {el1, el2, ...};

Example

A[1][0] = 3;

int A[2][2] = {1, 2, 3, 4};

The above declaration statement initialize array A in following manner;

A[0][0] = 1

A[0][1] = 2

A[1][0] = 3

A[1][1] = 4

* Array size (in bytes)

array size = row * column * size of datatype.

int A[2][2] = {10, 20, 30, 40};

array size = 2 * 2 * 2 = 8 bytes

1MV18EC057
Tejas

8) Explain ~~What are~~ built-in functions?.

→ * C supports a number of built-in function that make the job of a programmer easier. For instance, the string handling functions such as strcpy and strcmp are all built-in functions that perform standard pre-defined string manipulation task.

* The built-in functions are stored in related header file that need to be included in a program before the functions are actually called.

Header file.	functions
<stdio.h>	printf(), fprintf(), scanf(), getchar(), gets(), putchar(), puts(), fopen(), fclose(), feof(), ferror()
<stdlib.h>	atoi(), rand(), calloc(), malloc(), free(), exit(), abs()
<string.h>	strcpy(), strncpy(), strcat(), strcmp(), strncmp(), strchr(), strstr()
<time.h>	time(), difftime(), mktime(), ctime()
<math.h>	sin(), cos(), sinh(), cosh(), tanh(), log(), log10(), pow(), sqrt(), fopen(), fclose(), feof(), ferror().

INTRODUCTION TO C

9) What is function? What are different type of functions?

⇒ A function is a block of code that performs specific task.

Types of function:

1) Standard library functions (built in functions)

The standard library functions are built in functions and these functions are defined in header file.

2) User defined function

* User defined functions are custom functions designed by programmer to perform a specific task.
* To use user-defined functions we need to take care of:
  (i) function declaration
  (ii) function call.
  (iii) function definition.

Syntax: /* Function Prototype */
<return type> <fn_name>(type1 para1, type2 para2

IMV18ECO57
Teja

```
/* function call */
<type_variable> = <func_name>(value1, value2, ...);

/* Function definition */
<return_type> <func_name>(type1 para1, type2 para2)
{
    statement1
    ;
    ;
}
```

Example:

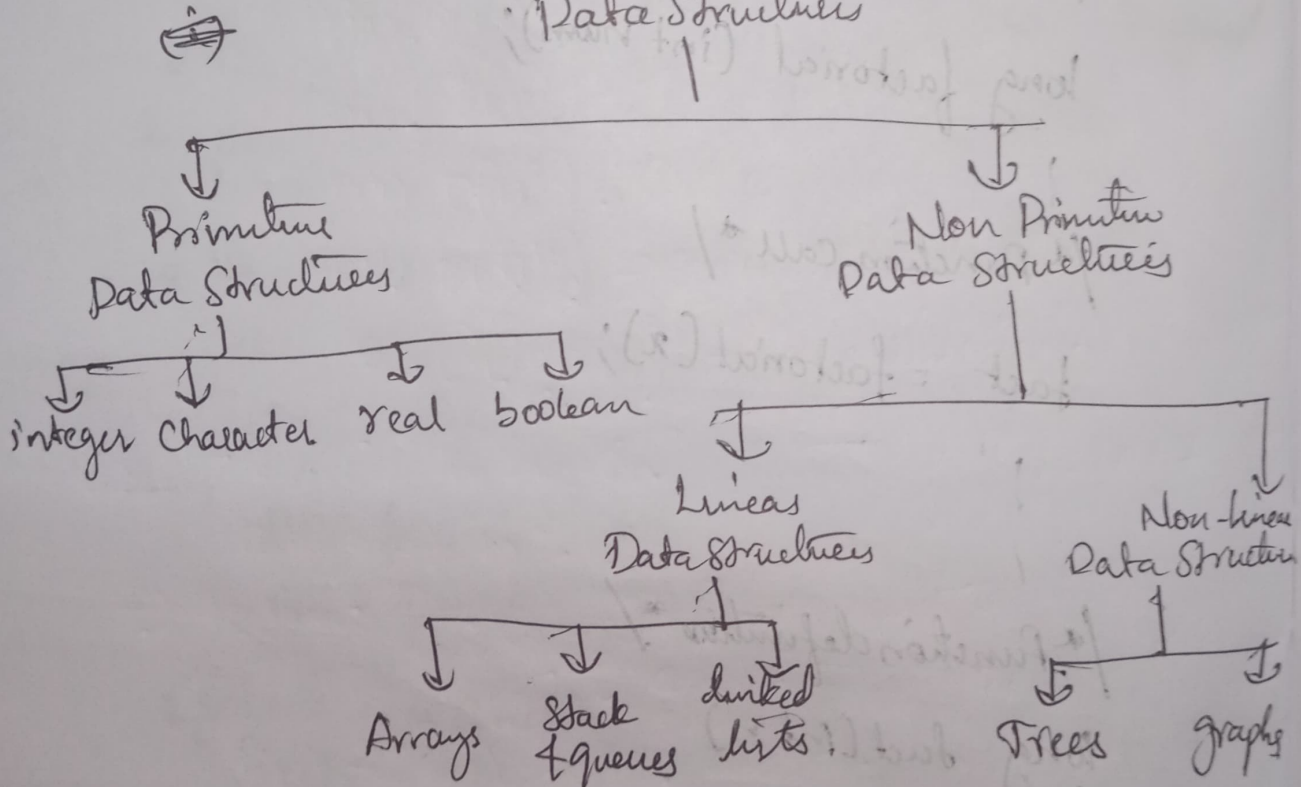
```
/* Function prototype */
long factorial (int nums);
{
/* Function call */
fact = factorial (x);
    ;
    ;
/* Function definition */
long fact(int n)
{ long f;
    if (n==0)
        return 1;
} else return n*fact(n-1);
```

10) What are data structures? What are the types of data structures

→ Data Structures can be defined as a representation of data along with its associated operations. It is the way of organizing and storing data in the computer system so that it can be used efficiently.

Types of Data Structures

* Data Structures are primarily divided into two classes,
  (i) primitive
  (ii) non-primitive.



(21)

11) Explain Asymptotic notations?

mvrit6cs7 Tejas.

-) * Asymptotic notations are the notations used to describe the behaviour of time or space complexity.
* It represents the efficiency and performance of algorithm in a systematic and meaningful manner.
* Asymptotic notations describe time complexity in terms of three common measures, best case (or 'fastest possible') , worst case (or 'slowest possible') and average case (or 'average time').

The three most important asymptotic notations are.
(i) Big-Oh notation
(ii) Omega notation
(iii) Theta notation

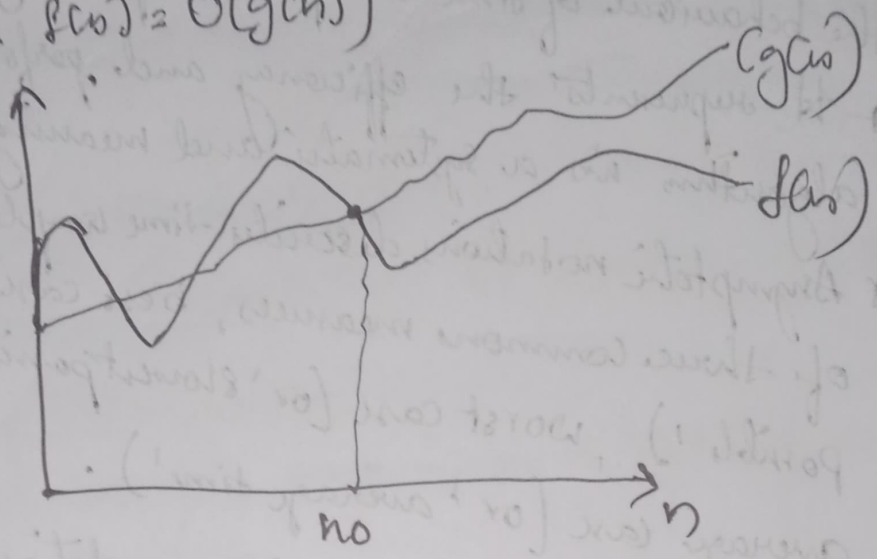
(i) Big-Oh notation
* It is a method that is used to express the upperbound of the running time of an algorithm.
* It is denoted by 'O'.
* Using this, we can compute maximum possible amount of time that an algorithm will take for its completion.

Definition: Consider $f(n)$ and $g(n)$ to be two positive functions of n, where $n \rightarrow$ size of input data. Then $f(n)$ is big-oh of $g(n)$, if and only if there exists a positive constant 'c' and an integer no, such that $f(n) \leq c \cdot g(n)$ for all $n \geq n_0$.

such that,

$f(n) < c.g(n)$ and $n > n_0$

Here $f(n) = O(g(n))$



$f(n) = O(g(n))$

(i) $O(1)$ ⇒ Constant

(ii) $O(n)$ → linear

(iii) $O(n^2)$ → Quadratic

(iv) $O(n^3)$ → Cubic

(v) $O(2^n)$ → Exponential

(vi) $O(\log(n))$ → Logarithmic.

Ex :- Denote the big-Oh notation if $f(n) = 8n + 7$ and $g(n) = n$.

→ To show $f(n)$ is $O(g(n))$, we must consider positive constant c and integer n_0, such that

$f(n) \leq c.g(n)$ for all $n > n_0$.

or $8n + 7 \leq cn$ for all $n > n_0$.

Let $C = 15$.

Now, we must show that $8n+7 \leq 15n$

or $7 \leq 7n$.

or $1 \leq n$.

Therefore, $f(n) = 8n+7 \leq 15n$ for all $n \geq 1$,

where $C = 15$ and $n_0 = 1$

Hence $f(n) = O(g(n))$

(ii) Omega Notation

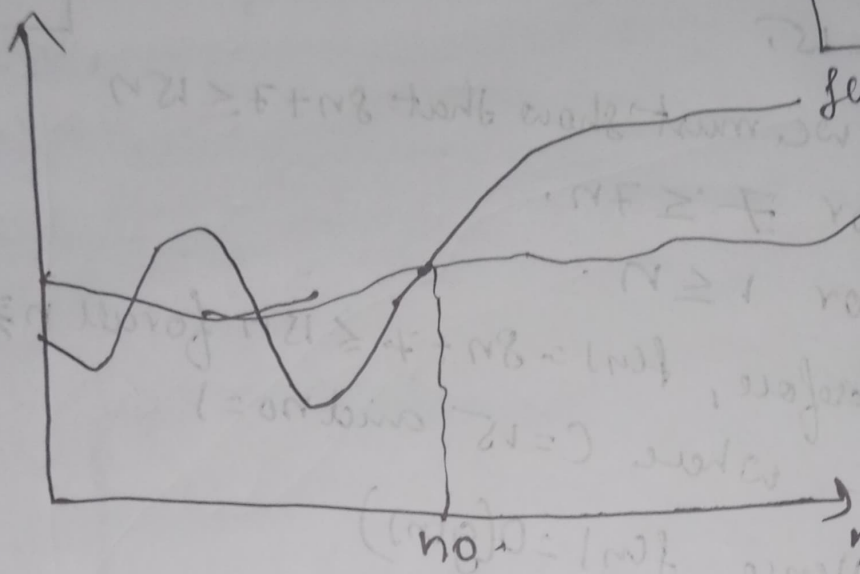
* The Omega Notation method is used to express the lower bound of running time of an algorithm.
* Omega notation is denoted by 'Ω'.
* Using this notation you can compute the minimum amount of time that an algorithm will take for its completion.

Definition: Consider $f(n)$ and $g(n)$ to be two positive functions of n, where n is the size of input data. Then $f(n)$ is omega of $g(n)$, if and only if there exists a positive constant C and an integer n_0, such that $f(n) \geq C(g(n))$ and $n > n_0$.

Here, $f(n) = \Omega(g(n))$

PTO.

1NV18BC057
Tejas



$f(n) = \Omega(g(n))$

Eg :- Deduce omega notation if $f(n) = 2n^2 + 4$ and $g(n) = 6n$

⇒ Given $f(n) = 2n^2 + 4$ and $g(n) = 6n$.

for $n = 0$,

$\therefore f(n) = 2(0)^2 + 4 = 4$ $\quad g(n) = 6(0) = 0$.

$f(n) > g(n)$

for $n = 1$

$f(n) = 2(1)^2 + 4$ $\quad g(n) = 6(1)$

$= 6$ $\quad g(n) = 6$

$f(n) = g(n)$

for $n = 2$,

$f(n) = 2(2)^2 + 4$ $\quad g(n) = 6(2)$

$= 2(2(4) + 4)$

$= 12$ $\quad = 12$.

$f(n) = g(n)$

1MV18EC057
Tejas

for n = 3.

$f(n) = 2(3)^2 + 4$ $\quad g(n) = 6(3)$

$= 18 + 4$ $\quad = 18$

$= 22$

$f(n) > g(n)$.

∴ we can say $f(n) > (g(n))$, if $n > 2$.

Theta Notation

* The theta notation is a method that is used to express the running time of an algorithm between lower and upper bounds. Theta notation is denoted by 'Θ'.
* Using this notation we can compute the average time that an algorithm will take for its completion.

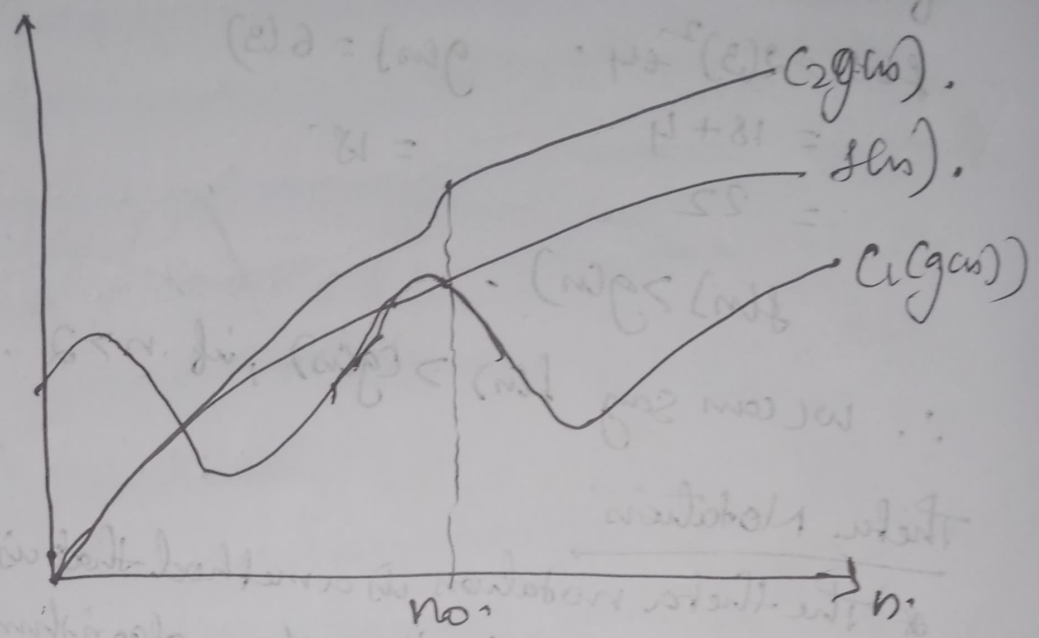
Definition: Consider $f(n)$ and $g(n)$ are two positive functions of n, where n is the size of input data. Then $f(n)$ is the theta of $g(n)$, if and only if there exists two positive constants C_1 and C_2, such that

$$C_1 g(n) \leq f(n) \leq C_2 g(n)$$

Here, $f(n) = \Theta(g(n))$

PTO.

INTRODUCTION
Tg's



$f(n) = \Theta(g(n))$

Ex:- Deduce the theta notation of $f(n) = 2n+8$.

→ let $f(n) = 2n+8 > 5n$ where $n \geq 2$.

Nly $f(n) = 2n+8 > 6n$ where $n \geq 2$.

& $f(n) = 2n+8 < 7n$ where $n \geq 2$.

Thus $5n < 2n+8 < 7n$ for $n \geq 2$

Here $C_1 = 5$ $C_2 = 7$

Hence $f(n) = 2n+8 = \Theta(n)$

INVISECOS 2
Tejas

12) What is Unions and pointers? Explain with Examples.

(i) Unions

* Like Structures, unions also allows us to group together dissimilar type elements and ~~is~~ acts as single unit.
* The size of the union is size of the largest sized element. This is because unions allows only one member to be utilized at any given point of time.

* Syntax

```
union <union_name>
{
    type1 var1;
    type2 var2;
    .
    .
    typen varn;
};
```

Eg:-

```
union result
{
    int marks;
    char grade;
    float percent;
};
```

(28)

INVISCOSZ
Tejas

Write a program to demonstrate the use of unions.

```
/* Program for demonstrating use of unions */
#include<stdio.h>
#include<conio.h>
void main()
{
    /* Union Declaration */
    union student
    {
        int roll_no;
        char result;
    } st1, st2;

    clrscr();
    // Initializing union variables.
    st1.roll_no = 20;
    st2.result = 'p';
    printf("\n Roll No : %d", st1.roll_no);
    printf("\n Result : %c", st2.result);
    printf("\n\n\n");
    printf("\n Roll no : %d", st2.roll_no);
    printf("\n Result : %c", st1.result);
    getch();
}
```

Output:

Roll No : 20

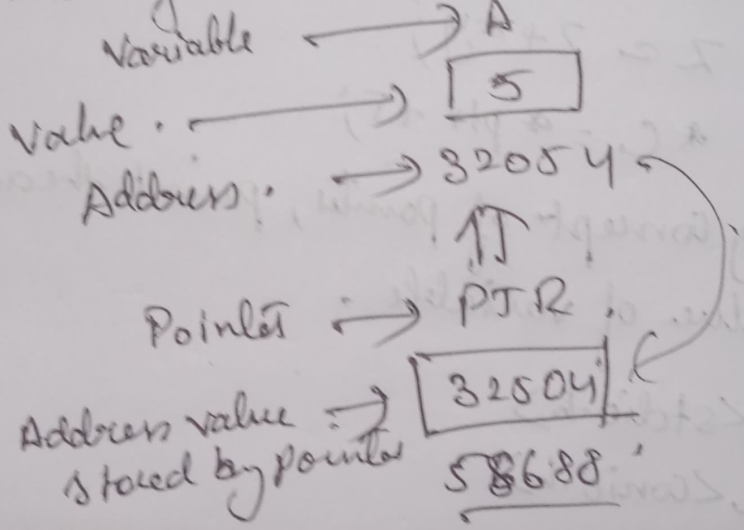
Result : P

Rollno : 12880

Result : Fail

(ii) Pointers

* Pointers is a derived datatype that stores memory address of its value. It points or indicates the location where another variable is stored.
* Pointers are particularly used for dynamic memory management.



* Pointers declared with help of * operator and initialized with help of & operator.

```
int num=10;
int *ptr; //Pointer declaration
ptr=&num; //Pointer initialization
```

PTO.

* In the above code, the address of num variable is allocated to the pointer variable ptr.
* Whenever * is preceeded with variable name, it refers to the value stored at the location being pointed by variable.
* Whenever & is preceded with variable name, it refers to the memory address of the variable.
* Pointer variable can also used in expression to form pointer expression.

```
a = *b + *c;
z = z + *y;
*c = *ptr + 5;
```

Eg :- Using concept of pointer, print the address and value of variable.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int a;
    int *ptr; //Declaration of pointer
    clrscr();
    a=50;
```

```
ptr = &a; // Pointer allocation
printf("address of a = %u \t value of a = %d \n", ptr, *ptr);
printf("address of ptr = %u \t value of ptr = %u",
                                    &ptr, ptr);
getch();
}
```

Output

address of a = 65524 value of a = 50

address of ptr = 65522 value of ptr = 65524