

Question outline:

(chapter-1)

1. Write an algorithm and draw a flowchart to find out the sum of the following series:

$$1^2 + 3^2 + 5^2 + \dots + N \text{ terms.}$$

(Note)

⇒ Algorithm: An algorithm is a procedure that used for solving a problem or performing a computation.

Flowchart: A flowchart is a graphical or symbolic representation of a process. A flowchart is a type of diagram that represents a workflow or process.

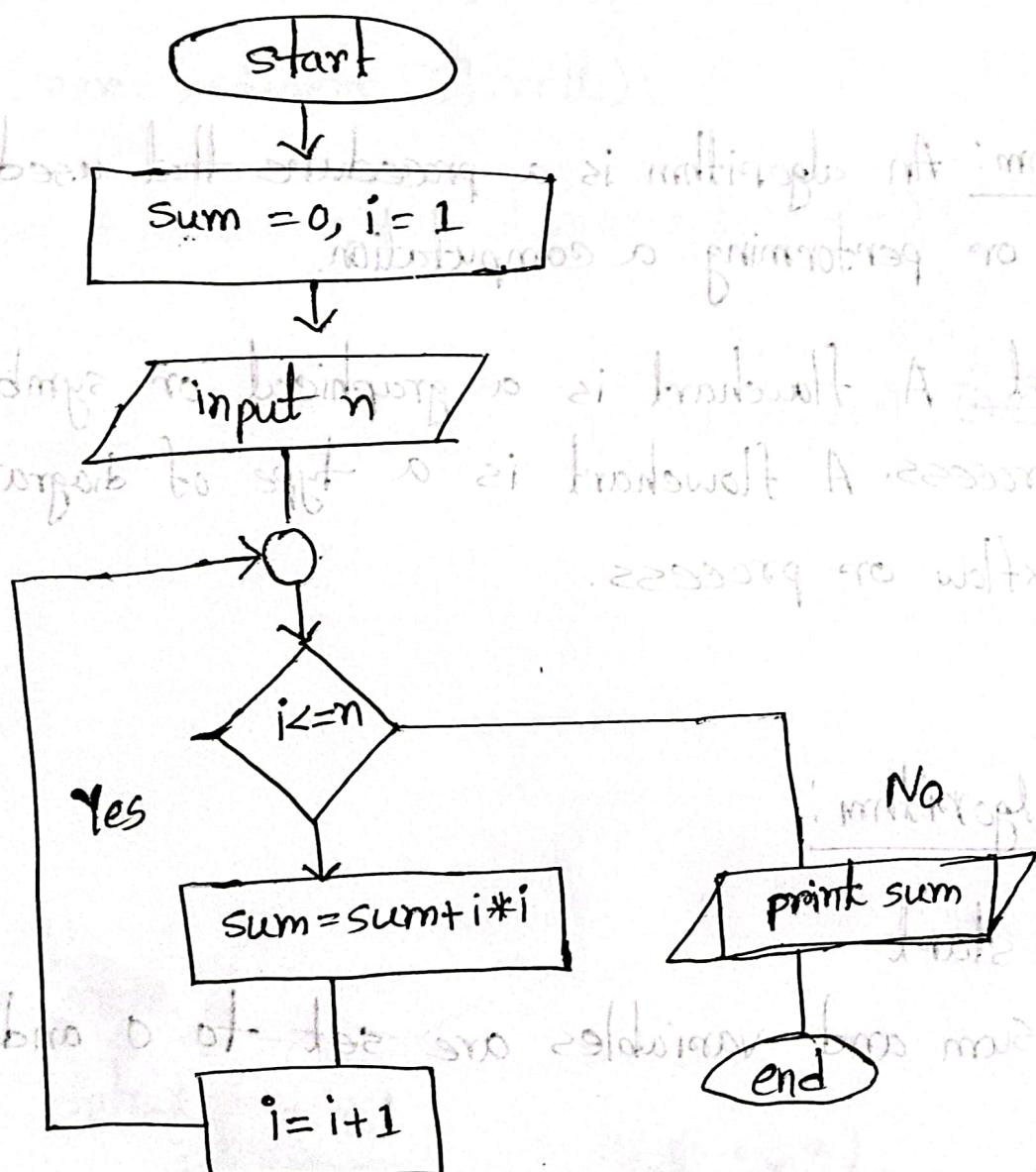
As

All Algorithm:step-1: startstep-2: sum and variables are set to 0 and 0 respectively take 1.step-3: taking the value of n as input.step-4: Compare the value of i with n. if  $i \leq n$ , then go to step 6, else go to step 5step-5: Sum variable value  $\text{sum} = \text{sum} + i * i$  and increment the variable value by 1.

Step-6: Display the value of sum as the result.

Step-7: Finish.

Flowchart:



## Chapter-2

Date: \_\_\_\_\_ / \_\_\_\_\_ / \_\_\_\_\_

Ques 2. Where & when do we use the #define directive?

In the C programming language the #define directive allows the definition of macros within our source code. These macro definitions allow constant values to be declared for in using our code.

Macro definition are not variables and can't be changed by our program code like variable, we generally use this syntax when creating constant that represent number, string or expression. We can use the #define directive to define a constant using an expression.

The syntax of using # define in the C language is,

#define constant\_name value.  
or

#define constant\_name expression.

Here most C program define their constant name is uppercase value contain the value of the constant. Here expression means whose value is assigned to the constant. The expression must be enclosed in program these if it contains operations or

Example: #define Age 10 |

Q3. Why and when we used #include directive?

→ #include is a way of including a standard or user-defined file in the program and is mostly written at the beginning of any C/C++ program.

#include is a processor directive that is used for file inclusion in a C-program. #include is also known as a file inclusion directive.

#include directive is used to add the content/piece of code from a reserved header file into our code file before the compilation of our C program. These header files include definition of many predefined function like printf(), scanf(), getch() etc.

The syntax for #include directive in the C language is

#include <header file>

Here the header file is that we wish to include. A header file is a file that typically ends in ".h".

Example: #include <stdio.h>

```
int main() {  
    int a=10, b=20;  
    int sum = a+b;  
    printf("the result is %d", sum);  
    return 0; }
```

In this program we are using the `#include` directive.

3. What does `void main(void)` mean?

→ The void `main()` indicated that `main()` function will not return any value. When our program is simple, and it is not going to terminate before reaching the last line of the code, or the code is error free, then we can use the void `main()`,

4. Why do we need to use comments in programs?

→ A comment is an explanation or description of the source code of a program. It helps a developer explain logic of the code and improves program readability. Proper use of commenting can make code maintenance much easier as well as helping make finding bugs faster.

Comments may be multiline or at single line, depending upon what we need. Single line comments are started with double slash. Ex. `// this is a single line comment`

Multiline comments are started with slash asterisk (`/*`) and ends with a asterisk slash (`*/`). It can be written anywhere in a program. Ex. `/* This is a multiline  
comments */`

(Page-28)

5. Describe the structure of a C program.

→ To write a C-program, we first create functions and then put them together. A C program may contain one or more sections as shown below:

### Documentation section:

The documentation section consists of a set of comment lines giving the name of the program, the author and other details, which the programmer would like to use later. The `/*` and `*/` symbols are used to denote the start and end of the documentation section.

Link section: The link section provides instructions to the compiler to link functions from the system library.

Definition section: The definition section defines all symbolic constants.

Global declaration section: There are some variables that are used in more than one function. Such variables are called global variables and are declared in the global declaration section that is outside of all the functions.

This section also declares all the user-defined functions.

Main() function section: Every C program must have one main() function section. This section contains two parts, declaration part and executable part. The declaration part declares all the variables used in the executable part. There is at least one statement in the executable part. These two parts must appear between the opening and closing braces. All parts ends with a semicolon ( ; )

Subprogram section: The subprogram section contains all the user defined functions that are called in the main function.

.36) 6. (2.1) Find errors, if any, in the following program :

```
* A simple program
int main()
{
    /* Does nothing */
}
```

6. Describe the four basic data types. How could we extend the range of values they represent?

⇒ The four basic data types is is int, char, float and void.

(i) Integer

(ii) character

(iii) Float

(iv) Void

i) Integer type: Integers are whole numbers with a range of values supported by a particular machine. Generally, integers occupy one word of storage, and since the word sizes of machines vary (typically, 16 or 32 bits) the size of an integer that can be stored depends on the computer. A signed integer uses one bit for sign and 15 bits for the magnitude of the number.

C has three classes of integer storage, namely short int, int and long int, in both signed and unsigned forms.

General form: int < variable name>;  
int num1;

ii) Character types: A single character can be defined as a character (char) type data. Characters are usually stored in 8 bits (one byte) of internal storage. The quantifiers signed or unsigned may be explicitly applied to char. While unsigned chars have values between 0 and 255, signed char have values from -128 to 127.

General form: char < variable name>; char ch = 'a';

iii) Floating point types: Floating point numbers are stored in 32 bits, with 6 digits of ~~precision~~ precision. floating point numbers are defined in C by the keyword ~~float~~ float. When the accuracy provided by a float number is not sufficient, the type double can be used to define the number. A double data type number uses 64 bits giving a precision of 14 digits.

General form: float <variable name>;

float num1;

iv) Void types: The void types has no value. This is usually used to specify the type of functions. The type of a function is said to be void when it does not return any value to the calling function.

We can use the short, long, signed and unsigned keywords to extend the primary data types. A short data type has a smaller range compared to the long data type.

(3.1)

7. What is a variable and what is meant by the 'value' of a variable?

→ A variable is a named unit of data that is assigned a value. If the value that is modified, the name does not change. Variables are used with Variable is basically nothing but the name of a memory location that we use for storing data. We can change the value of a variable in C or any other language, and we can also reuse it multiple times. We use symbols in variables for representing the memory location - so that it becomes easily identifiable by any user.

3.8)

8. How do variables and symbolic names differ?

Variables and symbolic names are both used to represent values in programming and computer science, but they differ in their usage and meaning.

A variable is a storage location in a computer's memory that holds a value. In programming, a variable is typically declared with a specific data type and assigned a value during runtime.

The value of a variable can be changed during the execution of a program, and it can be used to perform calculations, comparisons and other operations.

A symbolic name, on the other hand, is a name given to represent a value or concept in code. It is a label that is used to identify a particular value or object, but it does not hold any actual value itself. Symbolic names are typically used to make code more readable and maintainable, by providing a meaningful name that describes the purpose or function of a particular value or object.

For example, in a program that calculates the area of a circle, a variable might be declared as 'radius' and assigned a value of 5.0. The symbolic name 'PI' might also be defined in the code to represent the mathematical constant pi, which is used in the

9. What is operator in C language?

→ Operator: An operator is a symbol that operates on a value or a variable. For example: '+' is an operator to perform addition. C has a wide range of operators to perform various operations. There are eight types of operator [describing b described before] [for CT].

10. How many number of categories in C language of operators.

→ 8 categories (describe) (details).

→ [described before].

11. Explain the comma operator & size of operator with example

→ They are special type operator.

The comma operator:

The comma operator can be used to link the related expressions together. A comma-linked list of expressions are evaluated left to right and the value of right-most expression is the value of the combined expression. For example:

The statement,  $\text{value} = (x=10, y=5, x+y);$

first assigns the value 10 to x, then assigned 5 to y,

and finally assigns 15 (i.e.  $10+5$ ) to value. Some <sup>applications</sup> comma operators are:

In for loops:

`for(n=1, m=10, n<=m; n++, m++)`

In while loops:

`while (c=getchar(), c != '10')`

Exchanging values:

`t=x, x=y, y=t;`

The size-of operator:

The sizeof operator is a compile time operator and, when used with an operand, it returns the number of bytes the operand occupies. The operand may be a variable, a constant or a data type qualifier.

Examples:

`m = sizeof (sum);`

`n = sizeof (long int);`

`k = sizeof (235L);`

The sizeof operator is normally used to determine the lengths of arrays and structures when their sizes

are not known to the programmer. It is also used to allocate memory space dynamically to variables during execution of a program.

11. What do you mean by type conversion ~~in~~ expression?

In programming, Type conversion in expressions refers to the automatic or explicit conversion of one data type to another data type when they are used together in an expression.

For example, suppose we have an expression that involves both an integer and a floating-point number, like '5+2.5'. In this case, the integer 5 will be automatically converted to a floating-point number before the addition operation is performed, resulting in the expression evaluating to '7.5'.

There are two types of conversions: Implicit and explicit. Implicit conversions are performed automatically by the programming language, while explicit conversions are performed explicitly through the use of type-casting operations.

Type conversions can be important in programming because they can affect the accuracy of calculations, as well as the

efficiency and readability of code. Programmers need to be aware of the rules governing type conversions in their programming language to ensure that their code behaves as expected.

12. Differentiate between implicit and explicit type conversions.

Implicit conversion is the automatic conversion of one data type to another without any explicit code. It is also known as narrowing conversion. Explicit conversion is the manual conversion of one data type to another using a cast operator. It is also known as widening conversion. Both implicit and explicit conversions can be used to convert primitive data types to objects and vice versa. However, explicit conversion is more controlled than implicit conversion because it requires the programmer to specify the exact type conversion. This makes explicit conversion safer and more reliable than implicit conversion.

Implicit conversion is the automatic conversion of one data type to another without any explicit code. It is also known as narrowing conversion. Explicit conversion is the manual conversion of one data type to another using a cast operator. It is also known as widening conversion. Both implicit and explicit conversions can be used to convert primitive data types to objects and vice versa. However, explicit conversion is more controlled than implicit conversion because it requires the programmer to specify the exact type conversion. This makes explicit conversion safer and more reliable than implicit conversion.

## Difference between Implicit and explicit

Implicit	Explicit
i) Implicit refers to something that is implied but not said directly	① Explicit refers to something that is stated directly
ii) When someone expresses something implicitly, there is room for confusion.	② When someone expresses something explicitly, there tends to be no confusion
iii) <del>Implicit</del> is covert and not always immediately recognized.	③ Explicit is overt and easily recognized
iv) Explicit cont can't be determined easily	iv) <del>Implicit</del>
v) Implicit cont is <del>reflected</del> & does not have track and can't be determined precisely	iv) Explicit cont can be determine easily

v) An implicit function has several variables, one of which is a function of the other set of variables

v) An explicit function is one in which the dependent variable can be written explicitly in terms of the independent variable

## Chapter - 5

Date : ..... / ..... / .....

P-105,  
113

Q1 Explain formatted input & output with suitable example.

→ Formatted input refers to a way of accepting input from the user or from a file in a specific format or structure.

Formatted input refers to an input data that has been arranged in a particular format. For example, consider the following data:

15.75 123 John

This line contains three pieces of data, arranged in a particular form. Such data has to be read conforming to the format of its appearance. For example, the first part of the data should be read into a variable float, the second into int, and the third part into char. This is possible in C using the scanf function (scanf means scan formatted).

Formatted output:

→ Formatted output refers to the process of printing data to a console or file in a specific, organized format. It is often used to display in a more readable and user-friendly manner. +



What is the purpose of scanf and printf



'scanf' and 'printf' are functions in the C programming language that allow the input and output of data.

The purpose of `scanf` is to read input data from the user or a file and store it in a variable. It takes in one or more format specifiers that tell the function what type of data to expect and where to store it. For example, `scanf("%d", &myVariable)` will read an integer from the user or file and store it in the variable `myVariable`.

The purpose of `printf` is to output data to the console or a file. It takes in one or more format specifiers that tell the function what type of data to output and how to format it. For example, `printf("The value of myVariable is: %d\n", myVariable)` will output the value of `myVariable` to the console, preceded by the text "The value of myVariable is:" and followed by a newline character.

Together, `scanf` and `printf` allow programmers to create programs that can interact with users or read and write data to and from files.



15. What is the purpose of `scanf` and `printf`?   
 → pdf.

16. Write down the rules of switch statement.

Rules for switch statement:

- i) The switch statement must be an integer type.
- ii) Case labels must be constants or constant expressions.
- iii) Case label must be unique. No two labels can have the same value.
- iv) Case labels must end with colon.
- v) The break statement transfer the control out of the switch statement.
- vi) The break statement is optional. That is, two or more case labels may belong to the same statements.
- vii) The default label is optional. If present, it will be executed when the expression does not find a matching case label.
- viii) There can be at most one default label.
- ix) The default may be placed everywhere but usually placed at the end.
- x) It is permitted to nest switch statements.



write a c program to use goto statement



Sure! Here's an example program that uses the `goto` statement in C:

arduino Copy code

```
#include <stdio.h>

int main() {
    int num;

    input:
    printf("Enter a number between 1 and 10:\n");
    scanf("%d", &num);

    if (num < 1 || num > 10) {
        printf("Invalid input.\n");
        goto input;
    }

    printf("You entered %d.\n", num);

    return 0;
}
```

In this program, the `goto` statement is used to create a loop that prompts the user to enter a number between 1 and 10. If the user enters an invalid input (i.e. a number less than 1 or greater than 10), the program prints an error message and jumps back to the `input` label using the `goto` statement, allowing the user to try again. Once the user enters a valid number, the program prints a confirmation message and exits.

Write a c program using next step  
for loop



Sure, here's an example program in C that uses the "next step for loop" (also known as a "step for loop") to print out the even numbers between 0 and 20:

arduino Copy code

```
#include <stdio.h>

int main() {
    int i;

    // Loop through all even numbers
    for (i = 0; i <= 20; i += 2)
        printf("%d ", i);
}

return 0;
}
```

In this program, the loop starts at 0 and increments by 2 each time through the loop (i.e., the `i += 2` part). This means that it will only iterate through the even numbers between 0 and 20. The loop continues as long as `i` is less than or equal to 20. Inside the loop, the program prints out the current value of `i` using `printf`. Finally, the `main` function returns 0 to indicate successful program completion.

## **WORKED-OUT PROBLEM 7.7**

The program in Fig. 7.8 shows how to write a program to display a pyramid.

### **Program**

```
#include <stdio.h>
#include <conio.h>
void main()
{
    int num,i,y,x=40;
    clrscr();
    printf("\nEnter a number for \ngenerating the
pyramid:\n");
    scanf("%d",&num);
    for(y=0;y<=num;y++)
    {
        gotoxy(x,y+1);
        for(i=0-y;i<=y;i++)
        printf("%3d",abs(i));
        x=x-3;
    }
    getch();
}
```

### **Output**

Enter a number for  
generating the pyramid:

```
0
1 0 1
2 1 0 1 2
3 2 1 0 1 2 3
4 3 2 1 0 1 2 3 4
5 4 3 2 1 0 1 2 3 4 5
6 5 4 3 2 1 0 1 2 3 4 5 6
7 6 5 4 3 2 1 0 1 2 3 4 5 6 7
```

**Fig. 7.8** Program to build a pyramid

## What is one and two dimensional array? Define with example



In C programming, arrays are a way of storing multiple values of the same data type in a single variable. There are two types of arrays: one-dimensional arrays and two-dimensional arrays.

A one-dimensional array is a collection of elements of the same data type, arranged in a linear sequence. Each element in the array is accessed by an index, which represents its position in the array. Here's an example of a one-dimensional array in C:

C

[Copy code](#)

```
int arr[5] = {1, 2, 3, 4, 5};
```

This creates an array called `arr` with 5 elements, each of which is an integer. We initialize the array with the values 1 through 5. To access an element of the array, we use its index within square brackets, like this:

C

[Copy code](#)

```
int x = arr[2]; // x is now 3, since arr[2] is the third element of
```

A two-dimensional array is a collection of elements of the same data type, arranged in a grid of rows and columns. Each element in the array is accessed by two indices: one for its row and one for its column. Here's an example of a two-dimensional array in C:

```
c                                     Copy code

int mat[3][3] = {
    {1, 2, 3},
    {4, 5, 6},
    {7, 8, 9}
};
```

This creates a 3x3 matrix called `mat`, with 9 integer elements. We initialize the matrix with the values 1 through 9, arranged in rows and columns. To access an element of the matrix, we use its row and column indices within square brackets, like this:

```
c                                     Copy code

int x = mat[1][2]; // x is now 6, since mat[1][2] is the element in
```

In summary, one-dimensional arrays are used to store a linear sequence of elements, while two-dimensional arrays are used to store elements in a grid-like structure with rows and columns.

## Write a program to transpose matrix.

```
#include <stdio.h>

#define ROWS 3
#define COLS 3

void transpose(int mat[][COLS], int rows,
int cols);

int main() {
    int mat[ROWS][COLS] = {
        {1, 2, 3},
        {4, 5, 6},
        {7, 8, 9}
    };

    printf("Original matrix:\n");
    for (int i = 0; i < ROWS; i++) {
        for (int j = 0; j < COLS; j++) {
            printf("%d ", mat[i][j]);
        }
        printf("\n");
    }

    transpose(mat, ROWS, COLS);

    printf("Transposed matrix:\n");
    for (int i = 0; i < ROWS; i++) {
        for (int j = 0; j < COLS; j++) {
            printf("%d ", mat[i][j]);
        }
        printf("\n");
    }

    return 0;
}

void transpose(int mat[][COLS], int rows,
int cols) {
    int temp;
    for (int i = 0; i < rows; i++) {
        for (int j = i+1; j < cols; j++) {
            temp = mat[i][j];
            mat[i][j] = mat[j][i];
            mat[j][i] = temp;
        }
    }
}
```

Dynamic array is an array that can be resized during program execution. It is implemented using pointers and dynamic memory allocation functions such as `malloc()` and `realloc()`. A dynamic array allows for more flexibility and efficient use of memory compared to static arrays, as the size of the array can be adjusted as needed at runtime.

## 9th chapter

\* state the action for the following function

### Function

### Action

strcat()

concatenates two strings

strcmp()

compares two strings

strcpy()

copies one string over another

strlen()

finds the length of a string.

String: A string is a sequence of

characters that is treated as a single data item.

## II How to read a string from terminal

The familiar input function `scanf` can be used with %s format specification to read in a string of characters.

Example:

char address[10] [s|w|e|a]

`scanf ("%s", address);`

The problem with the `scanf` function is that it terminates its input on the first white space it finds. A whitespace includes blanks, tabs, carriage returns, form feeds and new lines. Therefore if the following line of the text is typed in at the terminal:

NEW YORK

Then only the string "NEW" will be read into the array address since the blank space after the word "New" will terminate the reading of string.

The `scanf` function automatically terminates the string that is read with a null character. The character array should be large enough to hold the input string plus the null character.

The address array is created in the memory as shown below

N	E	W	L	O	Z	Y	Y	Z	Y	Y	?	Y	Y	Y	Y
0	1	2	3	4	5	6	7	8	9	F	8	9	0	1	2

0 1 2 3 4 5 6 7 8 9 F 8 9 0 1 2

If we want to read the entire line "NEW YORK", then we may use two character arrays of appropriate sizes that is, char adr1[5], adr2[5]; if we assign the string "NEW" to adr1 and "YORK" to adr2.

### Chapter - 10

# user defined function: user defined function is a block of code written by the user to perform specific task.

# necessity of user defined function: C allows programmers to write their own functions, also known as user defined function. main is the example of user defined function. Every program must have a main function to indicate

Where the program has to begin  
its execution. There are many programs  
in c that are very difficult to understand,  
debug, test and maintain.  
so to avoid these difficulties we  
use user defined functions. user  
defined function helps us to  
decompose a large program into  
small segment. There are times  
when certain type of operation or  
calculations are repeated at many  
points throughout a program. we might  
used the factorial of a number  
at several point in the program.

In such situation we may repeat  
the program statements whenever  
they are needed. To design a  
function and called used user  
defined function.

# modular programming : modular programming  
is a strategy applied to the design and development of software systems.

# characteristics of modular programming

- ① each module should do only one thing
- ② communication between modules is allowed only by a calling module.
- ③ A module can be called by one and only one higher module
- ④ no communication can take place directly between modules that do not have calling-called relationship.
- ⑤ All modules are designed as single-entry single exit systems using control structures.

# elements of user defined function  
295 page.

# include <stdio.h>

```
int factorial (int n);
int main()
{
    int num, fact;
    printf ("Enter any number: \n");
    scanf ("%d", &num);
    fact = factorial (num);
    printf ("Factorial of %d = %d\n", num, fact);
    return 0;
}

int factorial (int n)
{
    int fact;
    if (n == 0)
        return 1;
    else
        return n * factorial (n - 1);
}
```

Input: Enter any number = 300

Output: 6 steps to pass an array

# writes <sup>the</sup> rules to pass an array  
to a function.

317 page গুরুত্বে 300 মাত্র ; সংজ্ঞা

# sorting: sorting is an operation  
that is needed to arrange the data set  
in a specified order.

searching: searching is an operation  
that is used when a look-up needs to  
be conducted on a set of data to  
locate a particular element.

Recursion: When a called function in turn  
calls another function a process of  
chaining occurs. Recursion is a special  
case of this process, where a function  
calls itself.

scope: scope is defined as the area in which the declared variable is available. There are five scopes in C program, file, function, block and prototype.

visibility: visibility is the accessibility of the variable declared.

Lifetime: the lifetime of a variable is the period of time in which the variable is allocated a space. There are three lifetime in C static, automatic, and dynamic.

# Describe the two ways of passing parameter to function. When do you prefer to use each of them.

# In C programming language, there are two ways of passing parameters from calling function to called function.

they are trying to . Both are diff of no. of arguments

① pass by value : memory location with passing

② pass by reference : address retained ad

① pass by value : In call by value

parameters passing method the copy of actual

parameters values are copied to formal

parameters and these formal parameters

are used in called function. The changes

made on the formal parameters does not

affect the values of actual parameters.

that means after the execution control

comes back to the calling function. the

actual parameters values remain same.

② pass by reference : In call by reference

parameters passing method , the memory

location address of the actual parameters

is copied to formal parameters. This address

is used to access the memory locations

of the actual parameters in called

function in this method. of parameter passing, the formal parameters must be pointer variable.

When I prefer to use each of them?

passing by value copies the data, so passing large data structures by value can inhibit performance passing by reference passes only a address to the data for large data structure.

this can greatly improve performance passing by value copies the data so if the target code modifies that copy. it will not affect the original

Passing by reference passes only the address of the data, so modification made against that performance will be visible to the calling code.

## what is prototyping in c program



Prototyping in C programming refers to the practice of declaring a function before it is defined or used in the code. A function prototype is a declaration of the function's signature, which includes its name, return type, and parameter types. It serves as a sort of "forward declaration" of the function, allowing the compiler to know how to call it and how to handle its return value and parameters before it is defined later in the code.

Here's an example of a function prototype:

c

Copy code

```
int add(int a, int b); // function prototype
```

This prototype declares a function named `add` that takes two integer parameters and returns an integer value. It can be used anywhere in the code before the actual function definition, such as in other functions that call `add`.

Prototyping is useful because it allows you to split your code into separate files or modules, with each module providing function prototypes for the functions it defines. This makes it easier to organize and maintain your code, as well as to prevent errors and bugs caused by incorrect function signatures or parameter types.

Chapter - 11

## Structures And Unions

1) What is structure?

Ans: A structure is a key word that creates user defined data type in C/C++. A structure creates a data type that can be used to group items of possibly different types into a single item type.

2) Differentiate between array vs structure?AnsArray

- a) An array is a collection of related data elements of same type.
- b) An array is a derived data type.
- c) we have to do is to declare an array variable and use it.

structure

- a) Structure can have elements of different type
- b) Structure is a programmer defined one.
- c) we have to design and declare a data structure before the variables of that type are declared and used.

51 what is union?

Ans; union is an user defined datatype in c programming language. It is a collection of variables of different datatypes in the same memory location.

61 Differentiate between structure and union?

Ans.

structure

- a) A structure is a key word that create user defined data type in c/c++.
- b) The keyword struct is used to define a structure.
- c) The size of structure is greater than or equal to the sum of size of it's members.
- d) Individual members can be accessed at a time.
- e) Several members of a structure can initialize at once.

union

- a) union is a collection of variables of different data types in the same memory location.
- b) The keyword union is used to define a structure union.
- c) The size of union is equal to the size of it's largest member.
- d) Only one member can be accessed at a time.
- e) only the first member of a union can be initialized.

## Q1 Rules for Initializing structures?

- Ans:
- a) we cannot initialize individual members inside the structure template.
  - b) it is permitted to have a partial initialization
  - c) we can initialize only the first few members and leave the remaining blank.
  - d) The uninitialized members will be assigned default values as follows
    - zero for integers and floating point number
    - "0" for characters and strings.

AR

what is pointer and give some advantage of it



A pointer in C programming is a variable that stores the memory address of another variable. In other words, a pointer points to a specific location in memory where the value of a variable is stored.

Some advantages of using pointers in C are:

1. Dynamic memory allocation: Pointers allow us to dynamically allocate memory at runtime using functions such as `malloc()` and `calloc()`. This is useful when we don't know the exact amount of memory required for a variable beforehand.
2. Passing parameters by reference: Pointers allow us to pass variables by reference to functions. This means that the function can modify the value of the variable passed as an argument.
3. Efficient memory usage: Pointers can be used to create complex data structures such as linked lists, trees, and graphs. These structures can be very memory-efficient compared to arrays.
4. Accessing hardware resources: Pointers can be used to access and manipulate hardware resources such as I/O ports and memory-mapped devices.
5. Implementing advanced algorithms: Pointers can be used to implement advanced algorithms such as sorting, searching, and graph traversal. These algorithms often require pointer arithmetic and indirect addressing.



Sure, here is a list of rules for working with pointers in C programming:

1. Declaring a pointer using the `\*` operator before the variable name.
2. Initializing a pointer by assigning it the address of a variable using the `&` operator or a specific address using a cast.
3. Dereferencing a pointer to access the value it points to using the `\*` operator before the pointer variable.
4. Pointer arithmetic, which allows a pointer to be incremented or decremented to point to the next or previous memory location.
5. Null pointers, which are pointers that point to no memory location and can be initialized to null using the value `NULL`. They can be used to check if a pointer is valid before dereferencing it.



Sure! Here's an example program that uses pointers to swap the values of two integers:

c

[Copy code](#)

```
#include <stdio.h>

void swap(int *a, int *b);

int main() {
    int x = 10;
    int y = 20;

    printf("Before swapping: x = %d, y = %d\n", x, y);

    swap(&x, &y);

    printf("After swapping: x = %d, y = %d\n", x, y);

    return 0;
}

void swap(int *a, int *b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}
```

[Regenerate response](#)

## 91 Rules for Pass by pointers

- a) The types of the actual and formal arguments must be same.
- b) The actual arguments must be the addresses of variables that are local to the calling function.
- c) The formal arguments in the function header must be prefixed by the indirection operator.
- d) In the prototype, the arguments must be prefixed by the symbol.
- e) we must use the corresponding formal argument prefixed with indirection operator.



In C programming, you can define, open, and close files using the `FILE` data type and a set of functions from the standard I/O library. Here's an example program that demonstrates how to do this:

arduino

Copy code

```
#include <stdio.h>

int main() {
    // Define a FILE pointer variable
    FILE *fp;

    // Open a file for writing
    fp = fopen("output.txt", "w");

    // Check if the file was opened successfully
    if (fp == NULL) {
        printf("Failed to open file.\n");
        return 1;
    }

    // Write some text to the file
    fprintf(fp, "Hello, world!\n");

    // Close the file
    fclose(fp);

    return 0;
```

Regenerate response

1



```
// Check if the file was opened successfully
if (fp == NULL) {
    printf("Failed to open file.\n");
    return 1;
}

// Write some text to the file
fprintf(fp, "Hello, world!\n");

// Close the file
fclose(fp);

return 0;
}
```

In this program, we start by defining a `FILE` pointer variable `fp`. We then use the `fopen` function to open a file named "output.txt" for writing, with the mode "w". If the file cannot be opened, we print an error message and return from the program.

After the file is successfully opened, we use the `fprintf` function to write the string "Hello, world!" to the file. Finally, we close the file using the `fclose` function.

Note that when opening a file, the mode parameter specifies how the file will be used. The "w" mode used in this example means that the file will be opened for writing and any existing content in the file will be erased. Other modes include "r" for reading, "a" for appending, and "w+" or "r+" for reading and writing.

Q. What are the fprintf and fscanf function?

Write down the general form:

→ The functions fprintf and fscanf perform input output operations that are similar to the printf and scanf functions. The functions fprintf and fscanf work on files. The first argument of these functions is a file pointer. The general form of fprintf is

```
fprintf(fp, "control string", list);
```

where fp is a file pointer that has been opened for writing. The control string contains output for the item in the list. The list may include variables, constants and strings.

For example,

```
fprintf(f1, "%s %d %.f", name, age, 7.5);
```

Here, name is an array variable of type char and age is an int variable.

The general form of fscanf is

```
fscanf(fp, "control string", list);
```

This statement would cause the reading of the items in the list from the file fp. For example,

```
fscanf(f2, "%s %d", item, &quantity);
```