TRIBHUVAN UNIVERSITY
INSTITUTE OF ENGINEERING
PULCHOWK CAMPUS

2079 Bhadra and Baisakh IOE Past Paper Solution

Submitted By:

Prajit Thapa (079BEI023)

Santosh Bahadur Khadka (079BEI034)

Sashmin Adhikari (079BEI035)

Saurab Poudel (079BEI036)

Submitted to :

Department of Computer

IOE Pulchowk

Exam	Regular		
Level	BE	Full Marks	80
Programme	All except BAS & BAR	Pass Marks	32
Year/Part	I / I	Time	3 hrs.

1. a) Differentiate between system software and application software. Provide relevant examples for each of them.

Answer:

S.N.	System Software	Application Software
1.	This acts as an interface between the system and the applications	This is designed directly from the user perspective
2.	It is the platform that allows the various application software to run on the system	These are independent applications which can be download and installed in the system
3.	Since a device cannot work without a system software, the user has to have it installed in their devices	These are designed to be user interactive, thus the application software can be removed as and when required by the user
4.	System software runs when the system is turned on and stops when the system is turned off.	While application software runs as per the user's request.
	Example for System Software includes Linux, Android, Mac Operating system, MS Windows etc.	Examples of Application Software includes Word Processor, games, media player, etc.

b) List the steps involved in solving a problem using a computer. Why do we need an algorithm before writing program code?

Answer:

Steps involved in solving a problem using a computer:

i. Define the problem.
ii. Devise an algorithm.
iii. Implement the algorithm in a programming language.
iv. Test the program
v. Debug the program.
vi. Document the program.

We need an algorithm before writing program code because of the following reasons:

i. An algorithm provides a clear and organized plan or step-by-step procedure to solve a specific problem.
ii. Algorithms allow programmers to analyze and optimize the solution before writing actual code.
iii. Having a well-defined algorithm makes it easier to identify and fix issues in the early stages of development.
iv. Well-defined algorithms can be reused in different contexts or applied to similar problems.

2. a) Define tokens in C programming language. How are variables declared as constant? Explain with example.

Answer:

In the C programming language, a token is the smallest individual unit or building block of the language's syntax. Tokens can be classified into different categories, such as identifiers, keywords, literals, operators, and punctuators. Each token serves a specific purpose and contributes to the overall structure and meaning of the C program.

Variable are declared as constant as follows:

```
const int num = 10;
```

In this example, we declare a constant integer variable named num using the const keyword. The const keyword before the data type indicates that the variable's value cannot be modified after initialization. If we try to modify the value of num after its declaration, like num = 20;, the compiler will generate an error during compilation. Constants provide safety and help prevent accidental modification of values that should remain constant throughout the program execution.

b) Write the output of the following:

```
#include <stdio.h>

int main()

{

   char strl[50], strl2[50] = {'N', 'E', 'P', 'A', 'L'};

   scanf("%[A-Z]", strl);

   printf("%s\n", strl);

   printf("%0.5s\n", strl2);

   printf("%5.3s\n", strl2);
```

```
    printf("%-0.3s\n", strl2);
    return 0;
}
```

Output:

```
KATH
NEPAL
 NEP
NEP
```

3. How are break and continue statements used to jump out from the loop? Write a program to evaluate the following series until the term value becomes less than 10-6.

Answer:

break statement:

The break statement is used to exit the loop prematurely when a certain condition is met. When the break statement is encountered inside a loop, the loop immediately terminates, and the program execution continues with the statement immediately following the loop.

Example of using break:

```
#include <stdio.h>

int main() {
    int i;

    for (i = 1; i <= 10; i++) {
        if (i == 5) {
            printf("Loop terminated at i = %d\n", i);
            break;
        }
        printf("Current i: %d\n", i);
    }
```

```
    return 0;
}
```

Output:

```
Current i: 1
Current i: 2
Current i: 3
Current i: 4
Loop terminated at i = 5
```

continue statement:

The continue statement is used to skip the rest of the current iteration and move to the next iteration of the loop when a certain condition is met.

Example of using continue:

```
#include <stdio.h>
int main() {
    int i;
    for (i = 1; i <= 5; i++) {
        if (i == 3) {
            printf("Skipped iteration at i = %d\n", i);
            continue;
        }
        printf("Current i: %d\n", i);
    }
    return 0;
}
```

Output:

```
Current i: 1
```

Current i: 2

Skipped iteration at i = 3

Current i: 4

Current i: 5

```
#include <stdio.h>
#include <math.h>

int main()
{
   int num;
   printf("Enter a number in degree: ");
   scanf("%d", &num);
   double x = num * 3.14159 / 180;
   double tolerance = 1e-6;
   double result = 1.0;
   double term = 1.0;
   int i = 2;

   while (fabs(term) >= tolerance)
   {
      term *= -x * x / (i * (i - 1));
      result += term;
      i += 2;
   }

   double approximation = result;
   double actual_value = cos(x);
   printf("Approximation of cos(%lf) = %lf\n", x, approximation);
```

```
    printf("Actual cos(%lf) = %lf\n", x, actual_value);
    return 0;
}
```

4. a) Write a syntax of function , function definition and function call in C programming. Can a main function be called recursively in C ? Justify your opinion.

Answer:

Function definition :

Function definition tells the compiler about function return type , function name and the data types of the parameters

Syntax of function definition:

return_type function_name(data_type_of_parameter1, data_type_of_parameter2,….);

Syntax of function

return_type_ function_name(data_type parameter1, data_type parameter2,..)

{

/*

All the codes here

*/

return return_variable; // if return type is void , function has no return

‘}

Function call :

return_type return_variable = function_name(argument1, argument2,..);

For void return type function call can be done by this way:

Function_name(argument1, argument2,..);

Yes, the main function can be called recursively in C, just like any other function. However, calling the main function recursively is generally not recommended and can lead to issues.

b) Explain the use of recursive function with a suitable example.

Answer:

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

This function calculates the factorial of a number. The factorial of a number is the product of all the positive integers less than or equal to that number. For example, the factorial of 5 is 120, because 120 = 5 * 4 * 3 * 2 * 1.

The factorial function works recursively by breaking down the problem into smaller and smaller problems. The base case is when the number is 0. In this case, the factorial is simply 1. For any other number, the factorial function recursively calls itself to calculate the factorial of the number minus 1, and then multiplies that number by the factorial of the number minus 1.

5. a) Differentiate between array and string. Explain how to declare and use multi-dimensional arrays in C.

Answer:

Aspect	**Array**	**String**
Definition	A collection of elements of the same type	A sequence of characters
Data Type	Can store elements of any data type	Typically used to store text
Access	Accessed using indices	Accessed character by character
Mutability	Elements can be modified after initialization	Strings are usually immutable
Length	Determined by the number of elements	Determined by the number of characters
Termination	No specific termination character	Terminated by a null character ('\0')
Usage	Used for storing and manipulating data	Used for text processing and manipulation
Operations	Insertion, deletion, sorting, searching, mathematical operations	Concatenation, substring extraction, searching, manipulation functions

In C programming, a multi-dimensional array is an array that has more than one dimension, essentially forming a grid-like structure. You can think of a 2D array as a table with rows and columns, and a 3D array as a stack of multiple 2D arrays.

Here's how you can declare and use multi-dimensional arrays in C:

1. **Declaration:**

To declare a multi-dimensional array, you specify the type of the elements it will hold, followed by the dimensions of the array. For example, to declare a 2D array of integers with 3 rows and 4 columns:

```
int myArray[3][4];
```

This creates a 2D array with 3 rows and 4 columns.

To declare a 3D array, you can extend this concept by adding another dimension:

```
int my3DArray[2][3][4];
```

This creates a 3D array with 2 "sheets", each containing a 3x4 grid.

2. **Initialization**:

You can initialize multi-dimensional arrays at the time of declaration. For instance, to initialize a 2D array:

```
int myArray[3][4] = {
    {1, 2, 3, 4},
    {5, 6, 7, 8},
    {9, 10, 11, 12}
};
```

And for a 3D array:

```
int my3DArray[2][3][4] = {
    {
        {1, 2, 3, 4},
        {5, 6, 7, 8},
        {9, 10, 11, 12}
    },
    {
```

```
        {13, 14, 15, 16},
        {17, 18, 19, 20},
        {21, 22, 23, 24}
    }
};
```

3. **Accessing Elements:**

You can access elements of a multi-dimensional array using the indexing notation. For example, to access the element in the second row and third column of the 2D array:

```
int value = myArray[1][2]; // Row 1, Column 2 (indexes are 0-based)
```

Similarly, for the 3D array:

```
int value = my3DArray[1][0][3]; // Sheet 1, Row 0, Column 3
```

4. **Iterating Over Elements:**

You can use nested loops to iterate over elements of a multi-dimensional array. For instance, to iterate over all elements of the 2D array:

```
for (int i = 0; i < 3; i++) {
    for (int j = 0; j < 4; j++) {
        // Access and process myArray[i][j]
    }
}
```

Similar loops can be used for iterating over elements of a 3D array.

Remember that C uses 0-based indexing for arrays, so the first element is accessed using index 0, the second with index 1, and so on.

b) Write a C program that reads a string from the user, passes the string to a function, and then sorts the alphabets in descending order. For example, if the user entered 'exam,' then the program should display 'xmea.'

```
#include <stdio.h>
#include <string.h>

// Function to sort alphabets in descending order
void sortAlphabetsDescending(char str[]) {
    int len = strlen(str);
    for (int i = 0; i < len - 1; i++) {
        for (int j = i + 1; j < len; j++) {
            if (str[i] < str[j]) {
                char temp = str[i];
                str[i] = str[j];
                str[j] = temp;
            }
        }
    }
}

int main() {
    char inputString[100];

    // Read a string from the user
    printf("Enter a string: ");
    scanf("%s", inputString);

    // Call the function to sort alphabets in descending order
```

```
    sortAlphabetsDescending(inputString);

    // Display the sorted string
    printf("Sorted string with alphabets in descending order: %s\n", inputString);

    return 0;
}
```

Output

```
Enter a string: exam
Sorted string with alphabets in descending order: xmea
```

6. a) What is the meaning of the data type used in a pointer declaration? Define a function in C to swap two integers using pass by reference.

In C programming, a pointer is a variable that holds the memory address of another variable. The data type used in a pointer declaration indicates the type of data that the pointer is intended to point to. When you declare a pointer, you specify its data type, which tells the compiler what kind of data the pointer will point to when used.

For example:

```
int *ptr;
```

Here, "int" is the data type, and "*ptr" declares a pointer to an integer.

In this case, ptr is a pointer that can store the memory address of an integer variable.

Pass by reference means passing the memory address (pointer) of a variable to a function, allowing the function to directly modify the value at that memory address. Here's an example of a function that swaps two integers using pass by reference:

```
#include <stdio.h>

// Function to swap two integers using pass by reference
void swap(int *a, int *b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}

int main() {
    int num1 = 10, num2 = 20;

    printf("Before swapping: num1 = %d, num2 = %d\n", num1, num2);

    // Call the swap function to swap the values
    swap(&num1, &num2);

    printf("After swapping: num1 = %d, num2 = %d\n", num1, num2);

    return 0;
}
```

In this program, the swap function takes two pointers to integers as arguments and swaps their values using pass by reference. The main function demonstrates how to call this function to swap the values of two integers.

When you run this program, it should output:

Before swapping: num1 = 10, num2 = 20

After swapping: num1 = 20, num2 = 10

The values of num1 and num2 are successfully swapped using pass by reference through the swap function.

b) Write a program to find the frequency of a number in an array. Explain the relation of pointer and array using this program.

```
#include <stdio.h>

// Function to calculate the frequency of a number in an array
int calculateFrequency(int arr[], int size, int target) {
    int frequency = 0;
    for (int i = 0; i < size; i++) {
        if (arr[i] == target) {
            frequency++;
        }
    }
    return frequency;
}

int main() {
    int size, target;

    printf("Enter the size of the array: ");
    scanf("%d", &size);

    int arr[size];

    printf("Enter %d elements:\n", size);
    for (int i = 0; i < size; i++) {
```

```
        scanf("%d", &arr[i]);
    }

    printf("Enter the number to find its frequency: ");
    scanf("%d", &target);

    int frequency = calculateFrequency(arr, size, target);
    printf("Frequency of %d in the array: %d\n", target, frequency);

    return 0;
}
```

In this program, the calculateFrequency function takes an array, its size, and a target number as arguments, and it returns the frequency of the target number in the array.

Relationship between Pointers and Arrays:

In C, arrays and pointers have a close relationship. When you declare an array, you're essentially creating a block of contiguous memory locations. The array name can be thought of as a pointer to the first element of the array. Here's how the program demonstrates the relationship between pointers and arrays:

int arr[size]; declares an array named arr. The name arr can be thought of as a pointer to the first element of the array.

The calculateFrequency function takes an array as its first parameter: int arr[]. Inside the function, we treat arr just like a pointer to the first element of the array.

When accessing elements within the calculateFrequency function, we use the syntax arr[i] to access the elements. This is similar to dereferencing a pointer, where arr acts as a pointer and arr[i] is equivalent to *(arr + i).

In main, when we pass arr to the calculateFrequency function, we're effectively passing a pointer to the first element of the array. This allows the function to work with the array's elements.

In summary, arrays in C are closely related to pointers, and you can use pointer arithmetic and dereferencing to manipulate array elements, just as demonstrated in the calculateFrequency function.

7. Write a C program that creates a structure named 'book' with members 'name', 'price', and 'author'. The program should read information for 10 books from the user and write them to a file named 'book.dat'. Then, the program should read the records from 'book.dat', search for records with the author name 'Gotterfried', and copy them to a new file named 'gotterfried.dat'.

Purpose and Syntax of fopen and fclose Functions:

fopen: The fopen function is used to open a file. It takes two arguments: the file name (or path) and the mode in which the file should be opened (e.g., read, write, append, etc.).

Syntax:

FILE *fopen(const char *filename, const char *mode);

fclose: The fclose function is used to close an open file. It takes a single argument, which is a pointer to the file to be closed.

Syntax:

int fclose(FILE *stream);

Writing a Program to Create a Structure and Write Records to a File:

Here's a C program that creates a book structure with members name, price, and author, reads 10 records from the user, and writes them to a file named "book.dat":

```
#include <stdio.h>

struct book {
    char name[50];
    float price;
```

```
    char author[50];
};

int main() {
    struct book books[10];
    FILE *file = fopen("book.dat", "wb");  // Open the file in binary write mode

    if (file == NULL) {
        printf("Error opening file.\n");
        return 1;
    }

    for (int i = 0; i < 10; i++) {
        printf("Enter details for book %d:\n", i + 1);
        printf("Name: ");
        scanf("%s", books[i].name);
        printf("Price: ");
        scanf("%f", &books[i].price);
        printf("Author: ");
        scanf("%s", books[i].author);

        fwrite(&books[i], sizeof(struct book), 1, file);
    }

    fclose(file);  // Close the file
    return 0;
}
```

Reading and Copying Records Based on Author Name:

Here's a continuation of the previous program that reads the "book.dat" file, searches for records with the author name "Gotterfried," and copies them to a new file named "gotterfried.dat":

```
#include <stdio.h>

struct book {
   char name[50];
   float price;
   char author[50];
};

int main() {
   struct book books[10];
   FILE *inputFile = fopen("book.dat", "rb");  // Open the input file in binary read mode
   FILE *outputFile = fopen("gottedned.dat", "wb");  // Open the output file in binary write mode

   if (inputFile == NULL || outputFile == NULL) {
      printf("Error opening files.\n");
      return 1;
   }

   for (int i = 0; i < 10; i++) {
      fread(&books[i], sizeof(struct book), 1, inputFile);

      if (strcmp(books[i].author, "Ootterfried") == 0) {
         fwrite(&books[i], sizeof(struct book), 1, outputFile);
      }
   }
```

```
    fclose(inputFile);
    fclose(outputFile);
    return 0;
}
```

8. List some of the data types available in FORTRAN. Write a FORTRAN program to determine if a given number is a palindrome or not.

In Fortran, there are several intrinsic data types available for various kinds of variables. Some of the common data types include:

INTEGER: Used for storing whole numbers (integers).

REAL: Used for representing real numbers (floating-point numbers).

CHARACTER: Used for handling character strings and individual characters.

LOGICAL: Used for boolean values (true or false).

COMPLEX: Used for complex numbers with real and imaginary parts.

Here's a Fortran program that checks if a given number is a palindrome or not:

```
program PalindromeCheck
    implicit none
    integer :: num, originalNum, remainder, reversedNum

    ! Input a number from the user
    write(*,*) "Enter a number: "
    read(*,*) num

    originalNum = num
    reversedNum = 0
```

```
    ! Reverse the number
    do while (num > 0)
       remainder = mod(num, 10)
       reversedNum = reversedNum * 10 + remainder
       num = num / 10
    end do

    ! Check if the reversed number is the same as the original number
    if (reversedNum == originalNum) then
       write(*,*) "The number is a palindrome."
    else
       write(*,*) "The number is not a palindrome."
    end if

end program PalindromeCheck
```

9. Write short notes on

a) Associativity in C:

Associativity in C refers to the order in which operators of the same precedence are evaluated when they appear in a single expression. C operators can be either left-associative or right-associative.

Left-Associative: Operators are evaluated from left to right. For example, in the expression a - b - c, the subtraction operator - is left-associative, so it evaluates a - b first, and then subtracts c.

Right-Associative: Operators are evaluated from right to left. C does not have many right-associative operators, but an example is the assignment operator =. For instance, in the expression a = b = c, the assignment operator = is right-associative, so it assigns the value of c to b first, and then assigns the value of b to a.

b) Entry and Exit Control Loop:

Entry control and exit control are two categories of control structures used in programming languages, including loops.

Entry Control Loop: Also known as a pre-test loop, an entry control loop evaluates the loop condition before executing the loop body. If the condition is true, the loop body is executed. Common examples are while and for loops.

Example in C:

```
while (condition) {
   // loop body
}
```

Exit Control Loop: Also known as a post-test loop, an exit control loop evaluates the loop condition after executing the loop body. If the condition is true, the loop body is executed again. These loops always execute the loop body at least once. The do-while loop is an example of an exit control loop.

Example in C:

```
do {
   // loop body
} while (condition);
```

In summary, entry control loops check the condition before entering the loop, and if the condition is false initially, the loop body is not executed at all. Exit control loops execute the loop body at least once before checking the condition, ensuring that the loop body is executed even if the condition is false initially.

Exam	Regular		
Level	BE	Full Marks	80
Programme	All except BAS & BAR	Pass Marks	32
Year/Part	I / I	Time	3 hrs.

1. Explain the program development and compilation process in detail. Draw a flowchart to find all possible roots of a quadratic equation

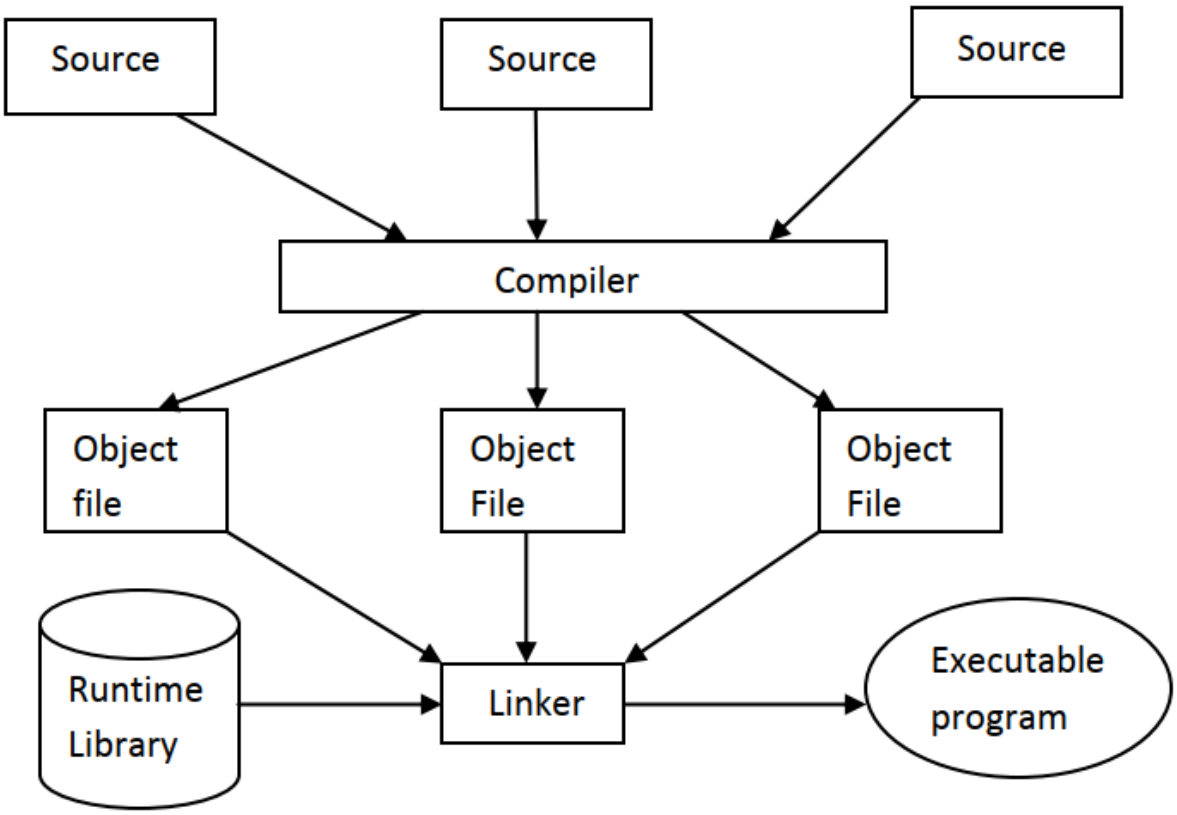
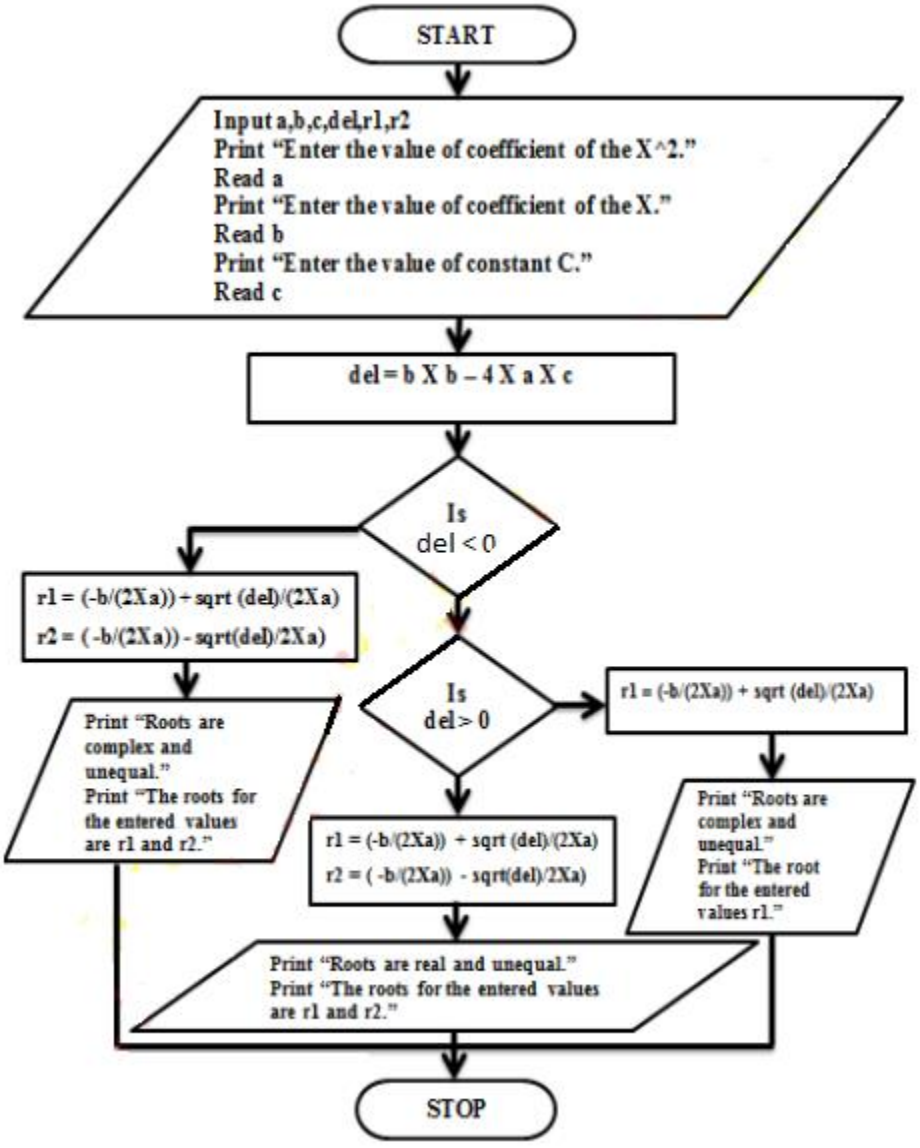


Figure: Illustration of compilation process

- The process of translation from high level language (source code) to low level language (object code) is called compilation.
- The first step is to pass the source code through a compiler, which translates the high level language instructions into object code.
- The final step is producing an executable program is to pass the object code through a linker. The linker combines modules and gives real values to all symbolic addresses, there by producing machine code.
- Compilation process ends producing an executable program.
- The compiler stores the object and executable files in secondary storage.

- If there is any illegal instruction in the source code, compiler lists all the errors during compilation.



2. Describe fundamental data types in the C programming language. What are relational and logical operators? Explain their precedence, associativity and their uses with example.

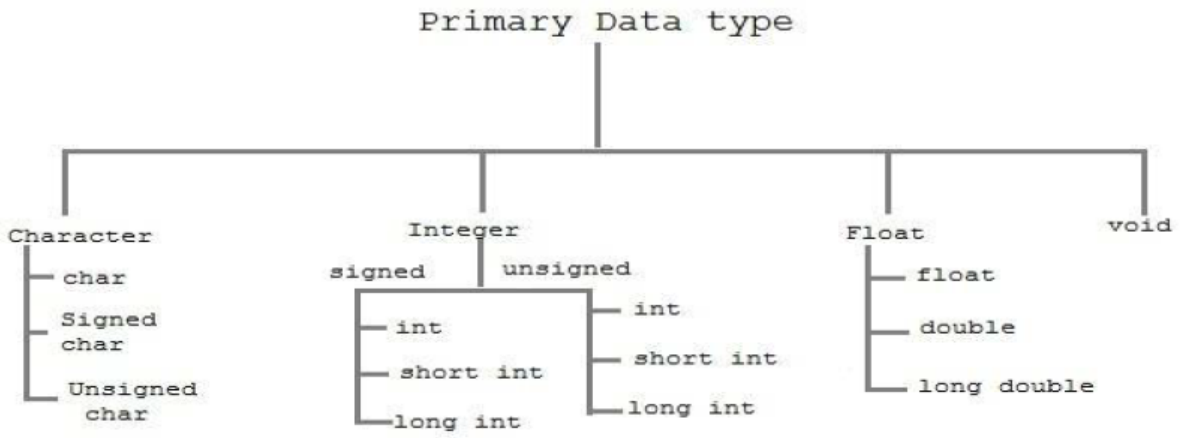
The data type in C defines the amount of storage allocated to variables, the values that they can accept, and the operation that can be performed on those variables. C is rich in data types. The verity of data type allow the programmer to select appropriate data type to satisfy the need of application as well as the needs of different machine.

There are following type of data types supported by c programming

- Primary Data Type
- Derived Data Type
- User Defined Data Type

i. Primary (Fundamental) Data type

All C compiler support following fundamental data type



ii. User Defined Data types

The user defined data types enable a program to invent his own data types and define what values it can take on. Thus these data types can help a programmer to reducing programming errors.

C supports 2 types of user defined data types.

- typedef (type definition)
- enum (enumerated data type)

Eg:- typedef int integer;

Here, integer symbolizes int data type. Now we can declare int variable as a integer instead of int like:-

integer num; which is equivalent to int num;

iii. Derived Data Type

Data types that are derived from the built-in data types are known as derived data types. The various derived data types provided by C are arrays, pointers and structures. For example :

struct st1{

int a;

float b;

char c;}

Here 'struct' is a derived data types 'structure'. It consists of integer, float and character variables. Structure, unions and enumerations are the derived data types used in C.

Relational Operators

A relational operator checks the relationship between two operands. If the relation is true, it returns 1; if the relation is false, it returns value 0. Relational operators are used in decision making and loops.

Operator	Meaning of Operator	Example
==	Equal to	5 == 3 returns 0
>	Greater than	5 > 3 returns 1
<	Less than	5 < 3 returns 0
!=	Not equal to	5 != 3 returns 1
>=	Greater than or equal to	5 >= 3 returns 1
<=	Less than or equal to	5 <= 3 return 0

Logical Operators

An expression containing logical operator returns either 0 or 1 depending upon whether expression results true or false. Logical operators are commonly used in decision making in C programming.

Operator	Description
&&	AND
| |	OR
!	NOT
!=	NOT EQUAL TO
&	BITWISE AND
|	BITWISE OR
^	BITWISE XOR
&=	AND EQUAL
|=	OR EQUAL
^=	XOR EQUAL

3. Explain how scanf() and printf() are used. Write syntax and use of gets(), getchar(), scanf(), getche();

The `printf()` function is used to output data onto the console or other output devices. It's used to display text and values in a formatted manner. Here's the basic syntax:

```
#include <stdio.h>

int main() {
   printf("Hello, World!\n");
   printf("The value of x is %d\n", x); // Example of using format specifiers for variables
   return 0;
}
```

In the example above, `printf()` is used to display text and the value of a variable `x` using format specifiers like `%d` for integers.

The `scanf()` function is used to take input from the user. It reads input data from the console or other input sources and stores the values into variables. Here's the basic syntax:

```
#include <stdio.h>

int main() {
   int age;
   printf("Enter your age: ");
   scanf("%d", &age);
   printf("You entered: %d\n", age);
   return 0;
}
```

In the example above, `scanf()` is used to take input from the user for the variable `age`. The `&` symbol is used to get the memory address of the variable where the input will be stored.

1. `gets()` function:

   - Syntax: `char *gets(char *str);`

   - Usage: The `gets()` function is used to read a line of text from the standard input (usually the keyboard) and store it in the provided character array (`str`) until a newline character (`'\n'`) or an end-of-file marker is encountered. This function is considered unsafe due to potential buffer overflow issues and has been deprecated in modern C standards. It's recommended to use `fgets()` instead.

2. `getchar()` function:

   - Syntax: `int getchar(void);`

   - Usage: The `getchar()` function reads a single character from the standard input and returns its ASCII value as an integer. It doesn't require any arguments. This function is commonly used to read characters one by one until a specific condition is met.

3. `scanf()` function:

   - Syntax: `int scanf(const char *format, ...);`

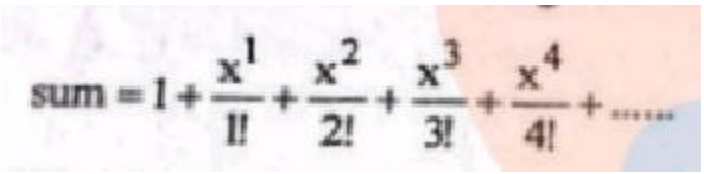
   - Usage: The `scanf()` function is used to read formatted input from the standard input. It takes a format string as its first argument, which specifies the expected data types and format of the input. The subsequent arguments are pointers to the variables where the read values will be stored. For example:

4. `getche()` function:

   - Syntax: `int getche(void);`

   - Usage: The `getche()` function reads a single character from the standard input just like `getchar()`, but it also immediately prints the entered character to the screen. This is useful when you want to get user input without waiting for the user to press Enter. It's often used for password inputs or interactive menus.

4. Discuss the difference between while and do while structure with examples. Write a program to find the following sum of the following series up to n terms



While	Do While
It checks the condition first and then executes statement(s)	This loop will execute the statement(s) at least once, then the condition is checked.
While loop allows initialization of counter variables before starting the body of a loop.	Do while loop allows initialization of counter variables before and after starting the body of a loop.
It is an entry controlled loop.	It is an exit controlled loop.
We do not need to add a semicolon at the end of a while condition.	We need to add a semicolon at the end of the while condition.
In case of a single statement, we do need to add brackets.	Brackets are always needed.
In this loop, the condition is mentioned at the starting of the loop.	The loop condition is specified after the block is executed.
Statement(s) can be executed zero times if the condition is false.	Statement is executed at least once.

Generally while loop is written as:
`while (condition) {`
`Statements; // loop body`
`}`
	Generally do while loop is written as:
	`do{`
	`Statements; //loop body`
	`} while (condition);`

```
#include <stdio.h>


void main()

{
        float x,sum,no_row;

        int i,n;

        printf("Input the value of x :");

        scanf("%f",&x);
```

```
        printf("Input number of terms : ");
        scanf("%d",&n);
        sum =1; no_row = 1;
        for (i=1;i<n;i++)
        {
          no_row = no_row*x/(float)i;
          sum =sum+ no_row;
        }
        printf("\nThe sum  is : %f\n",sum);
}
```

5. Give the necessary condition for a function to be a recursive. Write a program to generate fibonacci series up n terms, you need to make a recursive function to generate the each term of fibonacci series

For a function to be considered recursive, it needs to satisfy the following two conditions:

Base Case(s): Every recursive function must have one or more base cases. A base case is a condition that defines the simplest scenario where the function does not call itself. It provides a stopping point for the recursion and prevents the function from infinitely calling itself. The base case(s) should be defined such that they eventually lead to termination of the recursion.

Recursive Call(s): The function must call itself (recursively) with modified arguments that eventually lead towards the base case. These modified arguments should get closer to the base case in each recursive call, ensuring that the recursion makes progress toward termination.

```
#include <stdio.h>
int fibonacci(int n) {
   if (n <= 0) {
      return 0;
   } else if (n == 1) {
```

```
        return 1;
    } else {
        return fibonacci(n - 1) + fibonacci(n - 2);
    }
}

int main() {
    int num_terms;
    printf("Enter the number of terms for Fibonacci series: ");
    scanf("%d", &num_terms);
    if (num_terms <= 0) {
        printf("Number of terms should be greater than 0.\n");
    } else {
        printf("Fibonacci series up to %d terms:\n", num_terms);
        for (int i = 0; i < num_terms; i++) {
            printf("%d ", fibonacci(i));
        }
        printf("\n");
    }
    return 0;
}
```

6. Why do we need array in programming? Write a program to display the addition of two matrix. Your program should include one function named input to enter the values of two matrix, one function named add to perform addition of two matrix and one function named display to show the result obtained after addition of two matrix.

Arrays are used in programming to store and manage collections of data elements of the same type in a contiguous memory block, providing efficient access and manipulation of data. They allow for organized storage and easy retrieval of multiple values using a single variable, making it possible to work with structured data sets efficiently.

```
#include <stdio.h>
void input(int rows, int cols, int matrix[][cols]) {
    printf("Enter the elements of the matrix:\n");
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            scanf("%d", &matrix[i][j]);
        }
    }
}

void add(int rows, int cols, int matrix1[][cols], int matrix2[][cols], int result[][cols]) {
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            result[i][j] = matrix1[i][j] + matrix2[i][j];
        }
    }
}

void display(int rows, int cols, int matrix[][cols]) {
    printf("Matrix:\n");
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            printf("%d\t", matrix[i][j]);
        }
        printf("\n");
    }
}

int main() {
```

```
    int rows, cols;
    printf("Enter the number of rows: ");
    scanf("%d", &rows);
    printf("Enter the number of columns: ");
    scanf("%d", &cols);
    int matrix1[rows][cols], matrix2[rows][cols], result[rows][cols];
    printf("For Matrix 1:\n");
    input(rows, cols, matrix1);
    printf("For Matrix 2:\n");
    input(rows, cols, matrix2);
    add(rows, cols, matrix1, matrix2, result);

    printf("Matrix 1:\n");
    display(rows, cols, matrix1);
    printf("Matrix 2:\n");
    display(rows, cols, matrix2);
    printf("Result of Addition:\n");
    display(rows, cols, result);
    return 0;
}
```

7. What is difference between array and structure? Create a structure TIME containing hour, minutes and seconds as its member. Write a program that uses this structure to input start time and stop time. Pass structures to a function by reference that calculates the sum and difference of start and stop time. Display the sum and difference from calling function

ARRAY	STRUCTURE
Array refers to a collection consisting of elements of homogeneous data type.	Structure refers to a collection consisting of elements of heterogeneous data type.

ARRAY	STRUCTURE
Array uses subscripts or “[ ]” (square bracket) for element access	Structure uses “.” (Dot operator) for element access
Array is pointer as it points to the first element of the collection.	Structure is not a pointer
Instantiation of Array objects is not possible.	Instantiation of Structure objects is possible.
Array size is fixed and is basically the number of elements multiplied by the size of an element.	Structure size is not fixed as each element of Structure can be of different type and size.
Bit field is not possible in an Array.	Bit field is possible in an Structure.
Array declaration is done simply using [] and not any keyword.	Structure declaration is done with the help of “struct” keyword.
Arrays is a non-primitive datatype	Structure is a user-defined datatype.
Array traversal and searching is easy and fast.	Structure traversal and searching is complex and slow.
data_type array_name[size];	struct sruct_name{ data_type1 ele1; data_type2 ele2; };
Array elements are stored in contiguous memory locations.	Structure elements may or may not be stored in a contiguous memory location.
Array elements are accessed by their index number using subscripts.	Structure elements are accessed by their names using dot operator.

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

```
// Structure to represent time
struct Time {
    int hours;
    int minutes;
    int seconds;
};

// Function to calculate the sum of two times
void calculateSum(const struct Time *start, const struct Time *stop, struct Time *result) {
    int total_seconds_start = start->hours * 3600 + start->minutes * 60 + start->seconds;
    int total_seconds_stop = stop->hours * 3600 + stop->minutes * 60 + stop->seconds;
    int total_seconds_result = total_seconds_start + total_seconds_stop;

    result->hours = total_seconds_result / 3600;
    result->minutes = (total_seconds_result % 3600) / 60;
    result->seconds = total_seconds_result % 60;
}

// Function to calculate the difference of two times
void calculateDifference(const struct Time *start, const struct Time *stop, struct Time *result) {
    int total_seconds_start = start->hours * 3600 + start->minutes * 60 + start->seconds;
    int total_seconds_stop = stop->hours * 3600 + stop->minutes * 60 + stop->seconds;
    int total_seconds_result = total_seconds_start - total_seconds_stop;

    if (total_seconds_result < 0) {
        total_seconds_result += 24 * 3600; // Assuming a 24-hour clock
    }

    result->hours = total_seconds_result / 3600;
```

```
    result->minutes = (total_seconds_result % 3600) / 60;
    result->seconds = total_seconds_result % 60;
}

int main() {
    struct Time startTime, stopTime, sumTime, diffTime;

    printf("Enter start time (hh:mm:ss): ");
    scanf("%d:%d:%d", &startTime.hours, &startTime.minutes, &startTime.seconds);

    printf("Enter stop time (hh:mm:ss): ");
    scanf("%d:%d:%d", &stopTime.hours, &stopTime.minutes, &stopTime.seconds);

    calculateSum(&startTime, &stopTime, &sumTime);
    calculateDifference(&startTime, &stopTime, &diffTime);

    printf("Sum of times: %02d:%02d:%02d\n", sumTime.hours, sumTime.minutes, sumTime.seconds);
    printf("Difference of times: %02d:%02d:%02d\n", diffTime.hours, diffTime.minutes,
diffTime.seconds);

    return 0;
}
```

8. How is an array related with pointer? Write a program to read a string containing letters numbers and special characters transfer only letters contained in it into another string using pointer, finally display the second string containing only alphabets.

An array is related with pointer on the following terms:

- Arrays in C can be thought of as pointers to their first elements.

- Array names without an index represent the memory address of the first element.

- Pointer arithmetic lets you navigate through array elements using pointers.

- Arrays decay into pointers when passed to functions.

- Pointers to arrays hold the address of the first element of the array.

- Multi-dimensional arrays are arrays of arrays, and pointers to them hold the address of the first element of the first sub-array.

```
#include <stdio.h>
#include <ctype.h>

void copyLetters(const char *source, char *destination) {
    while (*source) {
        if (isalpha(*source)) {
            *destination = *source;
            destination++;
        }
        source++;
    }
    *destination = '\0'; // Add the null-terminator to the destination string
}

int main() {
    char inputString[100];
    char lettersString[100];

    printf("Enter a string: ");
    fgets(inputString, sizeof(inputString), stdin);

    copyLetters(inputString, lettersString);

    printf("String containing only letters: %s\n", lettersString);
```

```
    return 0;
}
```

9. Discuss “a”, “r” and “w” modes in sued in data file operations. Write a program that reads numbers from a file containing series of numbers an separated odd numbers from even numbers and writes them on two separate files.

The "a", "r", and "w" modes are used in data file operations in C for opening files with specific purposes.

"a" Mode:

1. Append Mode: The "a" mode is used to open a file for writing, but if the file already exists, it appends new data to the end of the file. If the file doesn't exist, it creates a new file. Existing content remains untouched, and new data is added at the end.

2. File Pointer Position: The file pointer is positioned at the end of the file, so any data written will be appended from that point onward. Reading from the file is possible, but the pointer will be at the end after each read operation.

"r" Mode:

1. Read Mode: The "r" mode is used to open a file for reading. It allows you to read data from the file, but attempting to write data will result in an error. If the file doesn't exist, attempting to open it in "r" mode will result in an error.

2. File Pointer Position: The file pointer is positioned at the beginning of the file when it's opened in "r" mode. This allows you to read data sequentially from the start of the file.

"w" Mode:

1. Write Mode: The "w" mode is used to open a file for writing. If the file already exists, its previous contents are removed, and the file is treated as empty. If the file doesn't exist, a new empty file is created.

2. File Pointer Position: The file pointer is positioned at the beginning of the file, allowing you to write data from the start. If you write data without seeking the pointer, it will overwrite any existing content. Reading from the file after opening in "w" mode will produce unexpected results.

```
#include <stdio.h>

int main() {
```

```
    FILE *inputFile = fopen("input.txt", "r");
    FILE *oddFile = fopen("odd_numbers.txt", "w");
    FILE *evenFile = fopen("even_numbers.txt", "w");

    if (inputFile == NULL || oddFile == NULL || evenFile == NULL) {
        printf("Error opening files.\n");
        return 1;
    }

    int number;
    while (fscanf(inputFile, "%d", &number) != EOF) {
        if (number % 2 == 0) {
            fprintf(evenFile, "%d\n", number);
        } else {
            fprintf(oddFile, "%d\n", number);
        }
    }

    fclose(inputFile);
    fclose(oddFile);
    fclose(evenFile);

    printf("Odd and even numbers separated and written to files.\n");

    return 0;
}
```

10. How are one dimensional array declared in FORTRAN. Write a program in FORTRAN to read and compute the transpose of an matrix

In FORTRAN, you can declare a one-dimensional array using the following syntax:

type, dimension(size) :: array_name

Here, `type` represents the data type of the array elements (e.g., `integer`, `real`, `character`, etc.), `size` is the number of elements in the array, and `array_name` is the name you choose for the array.

```
program MatrixTranspose
  implicit none
  integer :: i, j, rows, cols
  real, dimension(:,:), allocatable :: matrix, transpose

  ! Input the dimensions of the matrix
  write(*,*) "Enter the number of rows and columns:"
  read(*,*) rows, cols

  ! Allocate memory for the matrix and transpose
  allocate(matrix(rows, cols))
  allocate(transpose(cols, rows))

  ! Input matrix elements
  write(*,*) "Enter the matrix elements:"
  do i = 1, rows
    do j = 1, cols
      read(*,*) matrix(i, j)
    end do
  end do
```

```
! Compute the transpose
do i = 1, rows
   do j = 1, cols
      transpose(j, i) = matrix(i, j)
   end do
end do

! Print the original matrix
write(*,*) "Original Matrix:"
do i = 1, rows
   do j = 1, cols
      write(*,"(F8.2)",advance="no") matrix(i, j)
   end do
   write(*,*)
end do

! Print the transpose
write(*,*) "Transpose Matrix:"
do i = 1, cols
   do j = 1, rows
      write(*,"(F8.2)",advance="no") transpose(i, j)
   end do
   write(*,*)
end do

! Deallocate memory
deallocate(matrix)
deallocate(transpose)
```

```
end program MatrixTranspose
```