COMPUTER PROGRAMMING

Question Paper Solution

TRIBHUVAN UNIVERSITY

INSTITUTE OF ENGINEERING PULCHOWK CAMPUS

Arun Pankaj Bhatta
079BEI008

Bibhav Jha
079BEI015

Roshan Sharma
079BEI030

Sujata Ghimire
079BEI045

1. (a) What is a program? Explain different types of programming languages in brief.

Answer: A program is a set of instructions that a computer can follow to perform specific tasks or achieve desired outcomes. These instructions are written in programming languages, which act as the intermediary between human-readable code and machine-executable code.

Programming languages can be broadly categorized into following types:

(i) Machine Level Language
(ii) Assembly Level Language
(iii) High Level Language
(iv) Very High-Level Language
(v) Artificial Intelligence

(i) **Machine Level Language**: Machine-level language, also known as machine code, is the lowest-level programming language directly understood by a computer's CPU. It uses binary digits (0s and 1s) to represent instructions and data. Each instruction corresponds to a specific CPU operation, like arithmetic calculations or memory access. Machine code is specific to a computer's architecture and processor, making programs non-portable across different CPU types. Writing and understanding machine code is challenging for humans, leading to the development of higher-level languages that are translated into machine code for execution. For example: x86 Machine Code, MIPS Machine Code, etc.

(ii) **Assembly Level Language**: Assembly language is a low-level programming language that represents instructions using mnemonic codes and symbols, making it more human-readable than machine code. Each mnemonic corresponds to a specific machine-level instruction. It provides a direct and more understandable representation of the CPU's operations, closely related to the architecture of the underlying processor. Assembly language allows programmers to write code that is specific to a particular computer system, making it less portable but more optimized for that system's performance. As with machine code, assembly language requires an assembler to convert the code into machine code for execution by the computer's CPU. For example: ARM Assembly, SPARC Assembly, etc.

(iii) **High Level Language**: A high-level programming language is a human-readable and abstract form of coding that provides a natural language-like syntax. It allows programmers to write instructions without being concerned about the computer's underlying hardware. High-level languages come with built-in functions, libraries, and advanced features that simplify complex tasks and enhance code organization. Before execution, high-level code is translated or compiled into machine code or intermediate code to run on the computer. For example: C, C++, etc.

(iv) **Very High-Level Language**: A very high-level programming language is an even more abstract and user-friendly form of coding, providing a simpler and more intuitive syntax. It offers a high level of abstraction, enabling programmers to focus on the problem-solving aspect rather than low-level details. Very high-level languages often

come with extensive libraries and frameworks, making it easier to implement complex functionalities with minimal effort. Due to their user-centric design, very high-level languages sacrifice some performance and control in favor of ease of use and rapid development. For example: Python, JavaScript, etc.

(v) **Artificial Intelligence**: Natural language programming allows programmers to write code using human language, making it accessible to non-technical individuals and reducing the learning curve for software development. Instead of traditional programming syntax, natural language programming relies on everyday words and phrases for expressing instructions and logic. This approach enables people without coding experience to communicate with computers and automate tasks using simple, understandable language constructs. Natural language programming aims to bridge the gap between human communication and computer programming, making it more inclusive and empowering users from diverse backgrounds. For example: ChatGPT, etc.

(b) What is an algorithm? How does algorithm and flowchart help in computer programming?

Answer: An algorithm is a step-by-step set of instructions or a precise method for solving a problem or accomplishing a task in computation or other fields.

Algorithms and flowcharts play crucial roles in computer programming, and here's an elaboration of their significance in points:

Algorithms:

1. **Logic and Step-by-step Approach:** Algorithms provide a clear and systematic logic for solving a problem or performing a task, breaking it down into individual steps.
2. **Efficiency:** Well-designed algorithms optimize resource usage, reducing time and memory requirements for program execution.
3. **Portability:** Algorithms are independent of programming languages, making them transferable across different platforms and environments.
4. **Reusability:** Algorithms can be applied to similar problems, promoting code reuse and modularity.
5. **Abstraction:** Algorithms abstract the solution from implementation details, allowing programmers to focus on the problem-solving aspect rather than specific language syntax.

Flowcharts:

1. **Visualization:** Flowcharts visually represent the algorithm's logical steps using shapes and symbols, making it easier for programmers to understand, plan, and communicate the solution.
2. **Clarity:** Flowcharts enhance the clarity of complex algorithms, helping to identify potential flaws or improvements in the logic.

3. **Debugging:** When errors occur, flowcharts help in pinpointing the specific step or decision that caused the problem, simplifying the debugging process.
4. **Collaboration:** Flowcharts serve as a common language for developers, designers, and stakeholders to discuss and agree upon the program's structure and logic.
5. **Documentation:** Flowcharts act as documentation of the program's design, making it easier for future developers to maintain and modify the code.

2. (a) Explain Ternary operator in C with an example. Define following terms.
(i) Preprocessor Devices
(ii) Keywords

Answer: Ternary operator in C is a compact way to express simple conditionals. Its syntax is condition? expression1: expression2. If the condition is true, it returns expression1, otherwise expression2. For example;

```
int number = 10;
char* result = (number % 2 == 0)? "even": "odd";
```

Here, the 'result' holds the value 'even' as the specified condition results true which the ternary operator operates by assigning the first string to the "result"

(i) **Preprocessor directives:** Preprocessor directives in C are special instructions that begin with a '#' symbol and are processed before the actual compilation of the code, controlling conditional compilation, including header files, and performing text substitution. For example; #include <stdio.h>, #include<string.h>, etc.
(ii) **Keywords:** Keywords in programming are reserved words with predefined meanings that form the foundation of a language's syntax and structure. For Example; int, printf, etc.

(b) Write the output of the following C program?

```
#include<stdio.h>
void main()
{

   int a, b;
   double c = 123.55667788;
   char str[]= "I enjoy programming";

   scanf("%3d%2d",&a, &b);
   printf("a=%5d\n b=%-7d", a, b);
   printf("\n%10.7s", str);
   printf("\n%8.3f",c);
   printf("\n%-10.6f",c);
}
```

Input: 123456 789

Output:

```
123456  789
a=   123
b=  45
I enjoy
  123.557
123.556678
```

3. **Write the difference between formatted I/O and unformatted I/O functions in C-programming. Write the syntax for the following functions.**
   (i) getche()
   (ii) getchar()
   (iii)scanf()

Answer: The difference between formatted I/O and unformatted I/O are:

S.N.	Formatted I/O	Unformatted I/O
1.	Formatted I/O uses human-readable representations for data, such as strings, numbers, or formatted dates, making it easier for users to understand and interpret the data.	Unformatted I/O deals with binary representations of data, which are machine-readable and not directly human-readable.
2.	Formatted output is commonly used for displaying results on the console or generating formatted reports.	Unformatted output is commonly used for saving and loading data from binary files and for data transmission between different systems or platforms.
3.	For example: printf(), scanf(), etc.	For example: getc(), getch(), etc.

(i) getche()

Syntax:

```
Var1 = getche(void);
```

(ii) getchar()

Syntax:

```
Var2= getchar(void);
```

(iii) scanf()

Syntax:

```
scanf("%format1, %format2…", argument1, argument2);
```

4. **(a) What do you mean by iteration? Explain the operation of break and continue statement with a suitable example.**

Answer: Iteration refers to the process of repeatedly executing a sequence of statements or a block of code until a specific condition is met or for a defined number of times.

In C, the break and continue statements are used to control the flow of loops (like for, while, and do-while).

Break: The break statement is used to prematurely exit a loop. When encountered, it immediately terminates the loop's execution, and the program continues with the next statement after the loop.

Continue: The continue statement is used to skip the current iteration of the loop and proceed to the next iteration, effectively skipping the remaining statements inside the loop for that specific iteration.

Example of break:

```
#include <stdio.h>

int main() {
    int i;

    for (i = 1; i <= 10; i++) {
        if (i == 6) {
            break;
        }
        printf("%d ", i);
    }

    return 0;
}
Output: 1 2 3 4 5
```

Example of continue:

```
#include <stdio.h>

int main() {
    int i;

    for (i = 1; i <= 5; i++) {
        if (i == 3) {
            continue; // Skip the iteration
when i equals 3
        }
        printf("%d ", i);
    }

    return 0;
}
Output: 1 2 4 5
```

(b) Write a program to check whether a word is palindrome or not without using library function.

Answer:

```
#include <stdio.h>

int isPalindrome(int num) {
    int originalNum = num;
    int reversedNum = 0;

    while (num > 0) {
        int remainder = num % 10;
        reversedNum = reversedNum * 10 + remainder;
        num /= 10;
    }

    return (originalNum == reversedNum);
}

int main() {
    int num;

    printf("Enter a number: ");
    scanf("%d", &num);

    if (isPalindrome(num)) {
        printf("%d is a palindrome.\n", num);
    } else {
        printf("%d is not a palindrome.\n", num);
    }

    return 0;
}
```

5. (a) What do you mean by function header? Explain the function parameters and its types.

Answer: A function header is the first line of a function that includes its return type, name, and parameters, defining the function's signature.

A function can take parameters which are just values you supply to the function so that the function can do something utilizing those values. These parameters are just like variables except that the values of these variables are defined when we call the function and are not assigned values within the function itself. Types of function parameters are:

a) Formal Parameters: Formal parameters are the values referenced in the parameter index of a subprogram.
b) Actual Parameters: The Actual parameters are the variables that are transferred to the function when it is requested.

For example:

```
int sum(int a,int b)
{
  return(a+b);
}

void main()
{
    int sum(int,int);
    int x=5,y=6;
    total = sum(x,y);
}
```

When we call a function in main() or anywhere else in the program, and the function we created needs parameters, we would pass parameters to it while calling the function. In the example above, we passed variables x and y to obtain the sum of x and y.

According to the example above, the formal parameters are a and b, and the actual parameters are x and y.

(b) Write a C program to calculate the sum of digits until the sum becomes a single digit number using recursion.

Answer:

```
#include<stdio.h>
void main()
{
   int n, i, j, r, sum =0, temp;
   printf("Enter a number:\t");
   scanf("%d",&n);
   temp = n;

   do{
      sum=0;
      while(n!=0){
         r = n%10;
         sum += r;
         n /= 10;
      }
      n = sum;
   }while(sum%10 == 0);
   printf("\n%d", sum);
}
```

6. (a) What is an array? Why is it necessary in C programming?

Answer: Array can be defined as a finite ordered set of homogeneous elements.

For example; int a[5] is the set finite set of integers and is defined as a[5].

Arrays are used when processing of large number of similar data is required. Some major reasons for array to be important in C programming are:

a. **Data Grouping:** Arrays allow you to group similar data items together, making it easier to manage and manipulate related values.
b. **Efficient Access:** Elements in an array are stored sequentially in memory, enabling efficient sequential access and quick random access using index-based retrieval.
c. **Algorithm Support:** Arrays are essential for various algorithms like sorting and searching, and they serve as a foundation for implementing more complex data structures.
d. **Memory Efficiency:** Arrays provide efficient memory usage by storing data in contiguous memory blocks, minimizing wastage and supporting predictable memory layouts.

(b) Write a program to display following pattern.

Pattern:

H
HE
HEL
HELL
HELLO
HELL
HEL
HE
H

Answer:

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

void main(){
   char str[]="HELLO";
   int i, j, l;
   l=strlen(str);

   for(i = 0; i < l; i++){
      for(j = 0; j <= i; j++){
         printf("%c",str[j]);
      }
      printf("\n");
   }
   for(i = 0; i < l; i++){
      for(j = 0; j < l-i; j++){
         printf("%c",str[j]);
      }
      printf("\n");
   }
}
```

7. (a) Is there any relation between an array and a pointer? If yes, then show the relation between an array and a pointer using suitable example.

Answer: There is a close relationship between arrays and pointers in C programming. In fact, arrays and pointers are somewhat interchangeable in certain contexts due to the way C handles array variables. When you use the name of an array in an expression, it "decays" into a pointer to its first element. This means that in many cases, an array's name can be treated like a pointer pointing to the first element of the array.

For example;

```
#include <stdio.h>

int main() {
    int numbers[3] = {10, 20, 30};
    int *ptr = numbers;

    printf("Value using array indexing: %d\n", numbers[1]);  // Prints 20
    printf("Value using pointer: %d\n", ptr[1]);  // Also prints 20

    return 0;
}
```

In this example, we have an array named numbers containing three integers. We create a pointer ptr and assign it the address of the first element of the array. Then, we use both array indexing (numbers[1]) and pointer indexing (ptr[1]) to access the second element of the array, which has the value 20.

8. What is a structure? Write a program to read a structure named "Faculty" having StaffID, Name, Address, and ServiceYear as members. Where ServiceYear is another structure having DurationYear as a member. Now display the details of those faculties whose service duration is more than 10 and less than 30 years.

Answer: Structure can be defined as a new named data type, thus extending the number of available data types. For example;

```
struct student
{
        char name[50];
        int roll;
        char sec;
        float marks;
} ;
```

Here, student is a new data type that has name, roll, sec and marks as its members.

```
#include <stdio.h>
void main(){
   struct ServiceYear{
      int DurationInYear;
   };

   struct Faculty{
      int StaffID;
      char Name[50];
      char Address[50];
      struct ServiceYear service;
   }f[5];

   int i;

   for(i = 0; i++; i<5){
      printf("Member-%d\n",i+1);
      printf("Staff ID\t");
      scanf("%d",&f[i].StaffID);
      printf("Staff Name\t");
      scanf("%s",f[i].Name);
      printf("Staff Address\t");
      scanf("%s",f[i].Address);
      printf("Staff service year\t");
      scanf("%d",&f[i].service.DurationInYear);
   }

   printf("\n\nFaculty Memeber who meet required criteria:\n");

   for(i = 0; i < 5; i++){
      if(f[i].service.DurationInYear> 10 && f[i].service.DurationInYear < 30){
         printf("Staff ID\t%d",f[i].StaffID);
         printf("Staff Name\t%s",f[i].Name);
         printf("Staff Address\t%s",f[i].Address);
         printf("Staff service year\t%d",f[i].service.DurationInYear);
      }
   }
}
```

9. (a) Why do we need file handling? Write different modes of file opening.

Answer: File handling is crucial in programming for tasks like storing data persistently, input and output operations, data sharing between programs, creating backups, logging, debugging, configuration management, and resource sharing.

In C programming, when you open a file using the fopen function, you specify a "mode" that indicates how you intend to use the file. Here are the different modes of file opening:

(a) "r" (Read): Opens the file for reading. The file must exist; if it doesn't, the opening fails.
(b) "w" (Write): Opens the file for writing. If the file already exists, its contents are truncated (erased). If the file doesn't exist, a new file is created.

(c) "a" (Append): Opens the file for writing, but the new data is added to the end of the file. If the file doesn't exist, a new file is created.
(d) "r+" (Read and Write): Opens the file for both reading and writing. The file must exist.
(e) "w+" (Read and Write): Opens the file for both reading and writing. If the file exists, its contents are truncated. If the file doesn't exist, a new file is created.
(f) "a+" (Append and Read): Opens the file for both reading and writing. New data can be added to the end of the file. If the file doesn't exist, a new file is created.

(b) What is the purpose of fseek and write a program to write the name, roll no, and age of five students into a disk file name "STUDENT.DAT".

Answer: The `fseek` function in C is used to reposition the file position indicator within a file, enabling random access and facilitating operations at specific byte offsets

```
#include <stdio.h>

void main()
{
   FILE *fp;
   int roll, age, i;
   char name[20];

   fp = fopen("STUDENT.dat", "w+");
   if (fp == NULL)
   {
      printf("Cannot open file");
   }

   for(i=0; i<5; i++)
   {
      printf("Student-%d\n",i+1);
      printf("Name:\t");
      scanf("%s", name);
      printf("Roll no.:\t");
      scanf("%d", &roll);
      printf("Age:\t");
      scanf("%d", &age);

      fprintf(fp, "%s\t%d\t%d\n", name, roll, age);
   }

   fclose(fp);
}
```

10. Describe X format and T format in FORTRAN. Differentiate between unconditional goto and completed goto in FORTRAN. Write a program in FORTRAN to sort elements of a ID array in ascending ass well as descending order.

Answer: In FORTRAN, "X format" adds spacing or characters in formatted input/output to control alignment, while "T format" sets tab positions for subsequent output fields, helping achieve proper data layout and alignment in I/O operations.

In FORTRAN, an "unconditional goto" directly transfers control to a labeled statement without conditions, while a "computed goto" dynamically selects a target label based on an expression's value. Both can make code less structured and harder to read, encouraging the use of more organized control structures instead.

```
PROGRAM ArraySort
  IMPLICIT NONE

  INTEGER, PARAMETER :: N = 10
  INTEGER :: ID(N)
  INTEGER :: Temp
  INTEGER :: i, j

  ! Input
  WRITE(*, *) "Enter", N, "integer elements:"
  DO i = 1, N
    READ(*, *) ID(i)
  END DO

  ! Ascending order
  DO i = 1, N - 1
    DO j = 1, N - i
      IF (ID(j) > ID(j + 1)) THEN
        Temp = ID(j)
        ID(j) = ID(j + 1)
        ID(j + 1) = Temp
      END IF
    END DO
  END DO

  WRITE(*, *) "Ascending order:"
  DO i = 1, N
    WRITE(*, *) ID(i)
  END DO

  ! Descending order
  DO i = 1, N - 1
    DO j = 1, N - i
      IF (ID(j) < ID(j + 1)) THEN
        Temp = ID(j)
        ID(j) = ID(j + 1)
        ID(j + 1) = Temp
      END IF
    END DO
  END DO

  WRITE(*, *) "Descending order:"
  DO i = 1, N
    WRITE(*, *) ID(i)
  END DO

END PROGRAM ArraySort
```