2069 Ashad Question Paper

Tribhuvan University
Institute of Engineering
Pulchowk Campus

Submitted by:

Avinav Panta	Raj Sujakhu	Sooraj Khanal
079BEI011	*079BEI028*	*079BEI047*

Submitted to:

Department of Computer and Electronics

2069 Ashad questions:

1. Differentiate between high level and low-level languages. Explain the steps of solving a problem using computer.

Answer:

Low level Language	High Level Language
They are generally faster than high level languages.	They are generally slower than low level language
Low level languages are more difficult to learn as they are closer to machine level languages.	High level languages are comparatively easier to learn.
Requires additional knowledge of the computer architecture	Does not requires additional knowledge of the computer architecture
They are machine dependent and are not portable	Are not machine dependent and therefore more portable
Require any additional knowledge of the computer architecture	Does not require any additional knowledge of the computer architecture
More error prone	Less error prone
Debugging and maintenance is difficult	Debugging and maintenance is comparatively easier
They are generally used for developing system software and embedded applications	They are used to develop a variety of applications like websites and software for mobile and computers.

Following are the ways of solving a problem using a computer:

1. Problem Analysis

This step involves analyzing the problem and brainstorming possible solution. Different people have different preferences on how to analyze a certain problem.

2. Algorithm Development

Algorithms are very helpful to learn how a certain program works and helps in coding. For any given problems, algorithms can be way to visualize the problem analysis done in the previous steps. An algorithm is step by step description of activities or methods to be processed for getting desired output from a given input

3. Flowchart Development

Flowcharts put algorithm into figures which can be especially helpful when coding. A flowchart is a pictorial representation of an algorithm that uses boxes of different shapes to denote different types of instructions. The actual instructions are written within these boxes using clear and concise statements. These boxes are connected by solid lines having arrow marks to indicate the flow of operation, that is, the exact sequence in which the instructions are to be executed.

4. Coding

To make a program, what we have write instructions to the computer which is known as code. The act of writing code in a computer language is known as coding. In other words, code is a set of instruction that a computer can understand.

5. Compilation and Execution

Compilation is the process in which a program translates a code that is written into a machine level language which a computer can understand. Compilation process are done by programs known as compilers. Each programming language has its own different compiler. After the compilation process if the written code is correct then the computer moves into the next step. Execution of a code is the process in which a computer receives instructions from the compiler and performs the required task.

6. Debugging and testing

This is the step in which the code we wrote is analyzed a program to check if there are any errors in it that would prevent the program from running. This step is called debugging. Another method for checking errors in a program is testing. In this method, either a human or a program runs the code to check if any errors show up and amend them if they do.

7. Documentation

This process refers to a brief information about the program we write. It is a usual practice in the industry for programmers to write specific instructions or caveats related to a section of code. This helps other more beginner programmers to understand various sections of codes.

Question no 2.

Consider a statement

$scanf(\%s, str);$

Where str is a variable

In the above statement why "&" symbol us not used? Can we input string with space in this statement? If not, why?

Answer:

The "&" symbol is not used in the statement because str denotes an array of characters and in C when an array is passed in a function, we are passing the address of the first character in an array.

No, we can not input string with spaces in this statement because the function *scanf* is designed to read a collection of non-white space characters like a space, tab and a new line. In order to read a string with space, the above statement can be modified as follows:

$$scanf("\%[^\backslash n]", \mathbf{str})$$

The "%[^\n] " format specifier reads an input of strings until a new line character is encountered.

Question no 3

Write a program in C to find all possible values of the root of a quadratic equation $ax^2 + bx + c$?

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

int main() {
   float a, b, c, d, real, imag, root1, root2;
    printf("Enter coefficients a, b, and c: ");
   scanf("%lf %lf %lf", &a, &b, &c);
   d = b * b - 4 * a * c;
   if (d > 0) {
      root1 = (-b + sqrt(d)) / (2 * a);
      root2 = (-b - sqrt(d)) / (2 * a);
      printf("Roots are real and different: \n");
      printf("Root 1 = %.2f\n", root1);
      printf("Root 2 = %.2f\n", root2);
   } else if (d == 0) {
    root1 = root2 = -b / (2 * a);
      printf("Roots are real and equal:\n");
      printf("Root 1 = Root 2 = %.2f\n", root1);
   } else {
      real = -b / (2 * a);
      imag = sqrt(-d) / (2 * a);
      printf("Roots are complex and different:\n");
      printf("Root 1 = %.2lf + %.2lfi\n", real, imag);
      printf("Root 2 = %.2lf - %.2lfi\n", real, imag);
}
   return 0;
}
```

Question no 4

Write down the significance of main() in C. What are the differences between pass by value and pass by reference arguments. Describe both with meaningful examples. 2+6

In C programming, the main() function represents the entry point of a program. A computer when given a task, starts with the instructions present in the main function. As such, the main() function is required in every C program. The instructions in this function makes the main logic of the programs. The main function serves as the entry point of any program.

In C programming language, generally all arguments are passed by value unless pointers are used. Let us consider the following example for pass by value:

```
#include <stdio.h>

void modify(int x) {
   x = x * 2;
}

int main() {
   int num = 5;
   modifyValue(num);
   printf("Num: %d\n", num);
   return 0;
}
```

The above program will give the following output:

Num : 5

This is an example of pass by value. This means that the function receives a copy of the actual value of the argument, not a reference or the memory address to the original variable. Any changes made to the parameter within the function do not affect the original value outside the function. In the above example, we can see that the function

$x = x * 2$; had no effect on the output of the program. In pass by value, the function only works with its own local copy of the value.

In C pointers can be used for pass by reference. Consider the following example:

```
#include <stdio.h>

void modify (int *x) {
   *x = *x * 2;
}
int main() {
   int num = 5;
   modify (&num);
   printf("Num: %d\n", num);
   return 0;
}
```

The output of the given program is:

$$Num: 10$$

When a pointer is passed in a function, the function receives the memory address of the value and not the actual value itself like in pass by value. This allows the function to indirectly modify the original value via pointers.

Question no 5

Explain how arrays can be passed to functions. WAP that passes an array to a function and prints the largest and the smallest element in it 2+6

Arrays are the collection of elements of the same data type. Consider the following example for passing arrays to functions:

```
#include <stdio.h>

void sum (int x[69]) {
   x[0] = 0;
}
int main() {
   int num[5]
   sum (num);
   printf("Element number zero is : %d\n", num[0]);
   return 0;
}
```

In this way arrays can be passed onto functions.

```
#include <stdio.h>

void  printLargeSmall(int x[5]) {
        int temp;
   for(int i = 0; i<5;i++)
        {
           for(int j=1; j<5;j++)
                {
                        if(x[j] > x[i])
                        { x[i] = temp;
                        x[i] = x[j];
                        x[j] = temp;}
```

```
                }
        }
        printf("The largest element is %d and the smallest is %d" , x[0], x[5]);
}
int main() {
   int num[5]
for(int i = 0; i<5;i++)
{
        printf("Please enter a number: ");
        scanf("%d", num[i]);
}
   printLargeSmall(num);
   return 0;
}
```

Question no 6

How are structures different from arrays. Create a structure in C to store the name of a batsman, runs scored and the no of times the batsman is dismissed. In the program read the data of five players and display the batting average of the player whose name is entered by the user. Batting average is given by total runs / total dismissals 2+6

In C, an array is a collection of elements the same data type while a structure is a collection of elements of different data types. Consider the following example:

```
#include <stdio.h>
int main() {
   int num[5];
   struct num
{
        char name[6];
        float wages;
        int age;
}same;
   return 0;
}
```

In the above example *num* is an example of an array and it contains five elements of the int data type only while *same* is a structure which contains three elements, each of a different data type.

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

struct cricket
{
   char name[12];
   int run;
   int dismissals;
```

```
};


int main()
{
   struct cricket batsman[2];
   for(int i = 0; i<2;i++)
   {
      printf("Enter the name of a batsman: ");
     scanf("%s", batsman[i].name);

      printf("\nEnter the no of runs: ");
      scanf("%d", &batsman[i].run);

      printf("\nEnter the no of dismissals: ");
      scanf("%d", &batsman[i].dismissals);
   }
   char choice;
   char tempName[12];
   float avg;
   int g = 0;
   do
   {
      printf("Which batsman do you want to see: ");
      scanf("%s", tempName);
      for(int i = 0; i<2;i++)
      {
         if(strcmp(batsman[i].name, tempName) == 0)
         {
            avg = batsman[i]. run / batsman[i].dismissals;
```

```
            printf("The batting average of %s is %.2f\n", batsman[i].name, avg);
            g++;
          }
        if(g == 0)
        {
          printf("Player not found.\n");
        }

        }

        printf("Do you want to continue(Y\N): ");
        scanf("%c", &choice);

    }while(choice == 'Y' || 'y');
    return 0;
}
```

Question no 7

Write down the advantages of pointer. What type of arithmetic operations can be implemented in pointers. Also describe the relationship between array and pointer with approximate syntax and examples

Following are the advantages of pointers:

- Used in pass by reference
- More memory efficient
- Allows for dynamic memory allocation
- Helps while working with arrays
- Efficient string handling
- Helps to create dynamic data structures

The following arithmetic operations can be implemented in pointers:

- Addition or subtraction:

```
int arr[5] = {10, 20, 30, 40, 50};
int *ptr = arr;  // Pointer to the first element

// Move the pointer to the third element
ptr = ptr + 2;

// Access the value at the third element
int value = *ptr;  // value = 30
```

- Pointer comparison:

  Pointers can be compared by using <,> and = symbols. For eg:

```
int arr[5] = {10, 20, 30, 40, 50};
int *ptr1 = arr;        // Pointer to the first element
int *ptr2 = arr + 2;    // Pointer to the third element
```

```
if (ptr1 < ptr2) {
  // This code block will execute
}
```

- Dereferencing

Dereferencing a pointer allows a user to access the data stored in a particular memory address. For eg:

```
int value = 42;
int *ptr = &value; // Pointer to the integer variable

// Access the value using the pointer
int retrievedValue = *ptr; // retrievedValue = 42
```

Following is the relationship between pointers and arrays

- An array can be thought of as a constant pointer that points to its first element of an array. That is if we use an array without giving the index, it points to the first element. For example:

```
int numbers[5] = {10, 20, 30, 40, 50};
int *ptr = numbers; // Equivalent to int *ptr = &numbers[0];
int y = *ptr; // Access the first element (10)
```

In the above example, the array points to its first element.

Question no 8

WAP in C to read the following information for 96 students

Student Name, Student Roll number, Marks obtained in 100

Record all data in ioe.txt file and program should print roll number and real name of student who have obtained greater than or equal to 40 marks.

```
#include <stdio.h>

struct Student {
   char name[50];
   int rollNumber;
   int marks;
};

int main() {
   FILE *file;
   file = fopen("ioe.txt", "w+");
   if (file == NULL) {
      printf("FILE NOT FOUND.\n");
      return 1;
   }

   struct Student students[96];

   // Read student information
   for (int i = 0; i < 96; i++) {
      printf("Enter details for Student %d:\n", i + 1);
      printf("Name: ");
      scanf("%s", students[i].name);
      printf("Roll Number: ");
```

```
        scanf("%d", &students[i].rollNumber);
        printf("Marks obtained (out of 100): ");
        scanf("%d", &students[i].marks);
        printf("\n");

        // Write the student information to the file
        fprintf(file, "%s %d %d\n", students[i].name, students[i].rollNumber, students[i].marks);
    }

    // Close the file
    fclose(file);

    // Print the roll number and name of students with 40 marks or more
    printf("Students with 40 marks or more: \n");
    for (int i = 0; i < 96; i++) {
        if (students[i].marks >= 40) {
            printf("Roll Number: %d, Name: %s\n", students[i].rollNumber, students[i].name);
        }
    }

    return 0;
}
```

Question no 9

Rewrite the following source code correcting any error present in it. Also present the error corrected as a comment and then write the output of the program

Answer:

```
//program to convert a list of temperatures
//in centigrade to Fahrenheit
#include <stdio.h>
void convert(float x[n], float y[n], int n);
void cel2far(float f, float c);
// function should be declared before the main function
int main()
{
   int i;
   int n = 3;
   float celc[n], faren[n];

   for(i = 0; i<n; i++)
   {
      printf("Celci[%d] = ", i+1); // semi colon not included and %d should've had a int value after the quotes
      scanf("%d", celc[i]);
   }
   convert(cecl, faren, n);
   for(i = 0; i<n;i++)
   {
      printf("%d", faren[i]\n); // printf statment should be used like this
   }
   return 0;
}

void convert
```

```
{
   for(int i = 0; i<n; i++)
   {
      cel2far(far[i], cel[i]); // for loop body parts should be inside curly brackets
   }
}
void cel2far
{
   f = 9/5 *c + 32;
}
```

Question no 10

In Fortan, do loops are done in the following way:

```
do index_variable = start_value, end_value, step_size
  ! Code to be executed inside the loop
end do
```

Program to store and sort integers in Fortan:

```
program sort_integers
  implicit none
  integer, parameter :: size = 10
  integer :: i, j, temp
  integer :: arr(size)

  ! Read ten integers from the user
  print *, "Enter ten integers:"
  do i = 1, size
    read *, arr(i)
  end do

  ! Bubble sort: arrange the integers in ascending order
  do i = 1, size - 1
    do j = 1, size - i
      if (arr(j) > arr(j + 1)) then
        ! Swap the elements
        temp = arr(j)
        arr(j) = arr(j + 1)
        arr(j + 1) = temp
      end if
```

```
    end do
  end do

  ! Display the sorted integers
  print *, "Sorted integers in ascending order:"
  do i = 1, size
    print *, arr(i)
  end do

end program sort_integers
```

Question no 11

Syntax of two dimensional array in Fortan:

```
program two_dimensional_array_syntax
  implicit none
  integer, parameter :: rows = 3
  integer, parameter :: cols = 4
  integer :: i, j
  integer :: my_array(rows, cols)

  ! Initialize the array
  do i = 1, rows
    do j = 1, cols
      my_array(i, j) = i * 10 + j
    end do
  end do

  ! Access and use elements of the array
  do i = 1, rows
    do j = 1, cols
      ! Access element at row i and column j
      print *, "my_array(", i, ",", j, ") = ", my_array(i, j)
    end do
  end do

end program two_dimensional_array_syntax
```

Examples of this is:

```
program example
  implicit none
  integer, parameter :: rows = 3
```

```
  integer, parameter :: cols = 4
  integer :: i, j
  integer :: my_array(rows, cols)

  ! Initialize the array
  do i = 1, rows
    do j = 1, cols
      my_array(i, j) = i * 10 + j
    end do
  end do

  ! Display the array
  print *, "Contents of the two-dimensional array:"
  do i = 1, rows
    do j = 1, cols
      print *, my_array(i, j)
    end do
  end do

end program example
```