**Time allowed: 90 Minutes Max. Marks: 40**

**General Instructions:**

* **Follow the instructions given in each section.**
* **Make sure that you attempt the questions in order.**

**SECTION-A (10\*1 mark=10 marks)**

***(All questions are compulsory)***

1. How do you close a file in CPP?
   1. close(file)
   2. fclose(file)
   3. **fclose(file\_ptr)**
   4. close(file\_ptr)
2. How can you check if a pointer is dangling?
   1. By comparing it to NULL.
   2. **By checking if it points to a valid memory address.**
   3. By using a memory debugger tool.
   4. By performing a null check.
3. In a stack implementation using an array, what error occurs when the stack is full and a push operation is performed?
   1. **StackOverflowError**
   2. NullPointerException
   3. ArrayIndexOutOfBoundsException
   4. No error, the push operation overwrites the existing element
4. What is the purpose of using a pointer to a pointer?
   1. To allocate memory dynamically.
   2. **To access elements of a multi-dimensional array.**
   3. To pass a pointer by reference to a function.
   4. To store the address of a pointer.
5. What is the maximum depth of recursion in C++?
   1. There is no limit
   2. 1000
   3. 10000
   4. **It depends on the system's stack size**
6. What is function overloading in C++?
   1. Defining multiple functions with the same name but different return types
   2. **Defining multiple functions with the same name but different parameter types**
   3. Defining multiple functions with the same name and the same parameter types
   4. Defining multiple functions with the same name but different access specifiers
7. Which preprocessor directive is used to undefine a macro in C++?
   1. #include
   2. #define
   3. **#undef**
   4. #ifdef
8. What is the process of selecting the appropriate overloaded function at compile-time called?
   1. Overriding
   2. Overloading
   3. **Resolution**
   4. Compilation
9. Which of the following is true about inline functions?
   1. They have a separate memory space.
   2. They have a fixed return type.
   3. They have their own stack frame.
   4. **They are expanded at compile-time.**
10. What is the base case in a recursive function?
    1. The first step of the recursion
    2. The final step of the recursion
    3. The function that calls the recursive function
    4. **The condition that terminates the recursion**

**SECTION-B (5\*2 mark=10 marks)**

***(All questions are compulsory)***

11) What will be the output of the following code snippet?

int sum = 0;

for (int i = 1; i <= 5; i++)

{

sum += i;

if (sum > 10)

break;

}

cout << sum;

**a) 15**

b) 10

c) 11

d) Compilation error

12) What will be the output of the following C++ code?

#include <iostream>

using namespace std;

int main()

{

char c = 74;

cout << c;

return 0;

}

a) I

**b) J**

c) A

d) N

13) What will be the output of the following C++ code snippet?

#include <iostream>

using namespace std;

int operate (int a, int b)

{

return (a \* b);

}

float operate (float a, float b)

{

return (a / b);

}

int main()

{

int x = 5, y = 2;

float n = 5.0, m = 2.0;

cout << operate(x, y) <<"\t";

cout << operate (n, m);

return 0;

}

a) 10.0 5

**b) 10 2.5**

c) 10.0 5.0

d) 5.0 2.5

14) What will be the output of the following C++ code?

#include <iostream>

using namespace std;

void mani()

void mani()

{

cout<<"hello";

}

int main()

{

mani();

return 0;

}

a) hello

b) hellohello

**c) compile time error**

d) runtime error

15) What will be the output of the following C++ code?

#include <iostream>

using namespace std;

void fun(int x, int y)

{

x = 20;

y = 10;

}

int main()

{

int x = 10;

fun(x, x);

cout << x;

return 0;

}

**a) 10**

b) 20

c) compile time error

d) 30

**SECTION-C(Coding Question) (2x5 marks=5 marks)**

Q16) You have given a balanced expression. Write a program to find if it contains duplicate parenthesis or not.

A set of parenthesis are duplicate if the same subexpression is surrounded by multiple parenthesis.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Test Case 1** | **Test Case 2** | **Test Case 3** |
| **Input** | (((a+(b))+c+d)) | (((a+(b)))+(c+d)) | ((a+(b))+(c+d)) |
| **Output** | Duplicate Found | Duplicate Found | No Duplicates Found |

Solution :

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <stdbool.h>**

**#include <string.h>**

**#define MAX\_LENGTH 100**

**// Function to find duplicate parenthesis in a balanced expression**

**bool findDuplicateParenthesis(char\* str)**

**{**

**// Create a stack of characters**

**char stack[MAX\_LENGTH];**

**int top = -1;**

**// Iterate through the given expression**

**for (int i = 0; i < strlen(str); i++)**

**{**

**// If current character is closing parenthesis ')'**

**if (str[i] == ')')**

**{**

**// Pop character from the stack**

**char topChar = stack[top];**

**top--;**

**// Count the number of characters between a**

**// closing and opening parenthesis**

**// If this count is less than or equal to 1,**

**// then the brackets are redundant, else not**

**int elementsInside = 0;**

**while (topChar != '(')**

**{**

**elementsInside++;**

**topChar = stack[top];**

**top--;**

**}**

**if (elementsInside < 1) {**

**return true;**

**}**

**}**

**// Push open parenthesis '(', operators, and operands to stack**

**else**

**{**

**top++;**

**stack[top] = str[i];**

**}**

**}**

**// No duplicates found**

**return false;**

**}**

**int main()**

**{**

**// Input balanced expression**

**char str[MAX\_LENGTH] = "((a+(b))+(c+d))";**

**//char str[MAX\_LENGTH] ="(((a+(b)))+(c+d))";**

**if (findDuplicateParenthesis(str))**

**printf("Duplicate Found\n");**

**else**

**printf("No Duplicates Found\n");**

**return 0;**

**}**

Q17) Pilot wants to know direction to which he is heading the plane. Write a program that uses an enum to represent the cardinal directions (North, South, East, West).

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Test Case 1** | **Test Case 2** | **Test Case 3** |
| **Input** | W | N | E |
| **Output** | Heading West. | Heading North. | Heading East. |

Solution :

**#include <stdio.h>**

**// Enum representing cardinal directions**

**enum Direction {**

**North,**

**South,**

**East,**

**West**

**};**

**int main() {**

**enum Direction userDirection;**

**printf("Enter a direction (N, S, E, W): ");**

**char direction;**

**scanf(" %c", &direction);**

**switch (direction) {**

**case 'N':**

**userDirection = North;**

**printf("Heading North.\n");**

**break;**

**case 'S':**

**userDirection = South;**

**printf("Heading South.\n");**

**break;**

**case 'E':**

**userDirection = East;**

**printf("Heading East.\n");**

**break;**

**case 'W':**

**userDirection = West;**

**printf("Heading West.\n");**

**break;**

**default:**

**printf("Invalid direction.\n");**

**return 1;**

**}**

**return 0;**

**}**

**SECTION-D (Coding Question)(1x10 mark=10 mark)**

Q18) You are given a list of petrol pumps arranged in a circular path. Each petrol pump has an amount of petrol it can provide and the distance to the next petrol pump. Your task is to find the starting petrol pump from which a truck can complete the circular tour without running out of petrol. If there is no possible solution, print No Solution.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Test Case 1** | **Test Case 2** | **Test Case 3** |
| **Input** | {{6, 4}, {3, 6}, {7, 3}} | {{4, 6}, {6, 5}, {7, 3}, {4, 5}} | {{4, 6}, {3, 5}, {2, 3}, {4, 8}} |
| **Output** | Start= 2 | Start = 1 | No solution |

Solution :

**#include <stdio.h>**

**// A petrol pump has petrol and distance to next petrol pump**

**struct petrolPump**

**{**

**int petrol;**

**int distance;**

**};**

**// The function returns starting point if there is a possible solution,**

**// otherwise returns -1**

**int printTour(struct petrolPump arr[], int n)**

**{**

**// Consider first petrol pump as a starting point**

**int start = 0;**

**int end = 1;**

**int curr\_petrol = arr[start].petrol - arr[start].distance;**

**/\* Run a loop while all petrol pumps are not visited.**

**And we have reached first petrol pump again with 0 or more petrol \*/**

**while (end != start || curr\_petrol < 0)**

**{**

**// If current amount of petrol in truck becomes less than 0, then**

**// remove the starting petrol pump from tour**

**while (curr\_petrol < 0 && start != end)**

**{**

**// Remove starting petrol pump. Change start**

**curr\_petrol -= arr[start].petrol - arr[start].distance;**

**start = (start + 1)%n;**

**// If 0 is being considered as start again, then there is no**

**// possible solution**

**if (start == 0)**

**return -1;**

**}**

**// Add a petrol pump to current tour**

**curr\_petrol += arr[end].petrol - arr[end].distance;**

**end = (end + 1)%n;**

**}**

**// Return starting point**

**return start;**

**}**

**int main()**

**{**

**struct petrolPump arr[] = {{6, 4}, {3, 6}, {7, 3}};**

**int n = sizeof(arr)/sizeof(arr[0]);**

**int start = printTour(arr, n);**

**(start == -1)? printf("No solution"): printf("Start = %d", start);**

**return 0;**

**}**