**Study Guide for Midterm**

**Week 1**

1. **Discuss compile-time error vs runtime error with example:**
   * **Compile-Time Error: Errors detected by the compiler during the compilation of the program, such as syntax errors, undeclared variables, etc.**
     + **Example:**

**int main() {**

**int number = "hello"; // Compile-time error: incompatible type assignment**

**return 0;**

**}**

* + **Runtime Error: Errors that occur while the program is running, such as division by zero, accessing invalid memory, etc.**
    - **Example:**

**int main() {**

**int a = 10, b = 0;**

**int result = a / b; // Runtime error: Division by zero**

**return 0;**

**}**

1. **What are the different levels (high, mid, and low) of programming languages and discuss some differences among those?**
   * **High-Level Language: Closer to human language, easy to write and understand, needs a compiler/interpreter (e.g., Python, Java, C++).**
   * **Mid-Level Language: Acts as a bridge between high and low-level languages, provides hardware access along with high-level constructs (e.g., C language).**
   * **Low-Level Language: Close to machine code, efficient, but difficult to read and write (e.g., Assembly language).**
2. **What is Algorithm, discuss the approach of an algorithm that may help you organize a number of integers in non-decreasing order.**
   * **Algorithm: A set of step-by-step instructions to solve a problem.**
   * **Sorting Algorithm Example (Bubble Sort):**
     + **Start at the first element, compare it with the next element.**
     + **Swap them if they are in the wrong order.**
     + **Repeat the process until no swaps are needed to sort integers in non-decreasing order.**

**Week 2**

1. **Given a decimal number, show me the steps to convert it to the equivalent binary number.**
   * **Example: Convert 13 to binary.**
     1. **13 ÷ 2 = 6, remainder 1**
     2. **6 ÷ 2 = 3, remainder 0**
     3. **3 ÷ 2 = 1, remainder 1**
     4. **1 ÷ 2 = 0, remainder 1**
     5. **Binary equivalent: 1101**
2. **Given a binary number, show the steps to get its decimal equivalent.**
   * **Example: Convert binary 1011 to decimal.**
     1. **(1 × 2^3) + (0 × 2^2) + (1 × 2^1) + (1 × 2^0)**
     2. **8 + 0 + 2 + 1 = 11**
     3. **Decimal equivalent: 11**
3. **Given a binary number, translate it to corresponding hexadecimal equivalent.**
   * **Example: Convert binary 110111 to hexadecimal.**
     1. **Group in sets of four: 0011 0111**
     2. **0011 = 3, 0111 = 7**
     3. **Hexadecimal equivalent: 37**
4. **Given a positive binary number, find the 2’s complement.**
   * **Example: Find 2’s complement of 0101.**
     1. **Invert the bits: 1010**
     2. **Add 1: 1010 + 1 = 1011**
     3. **2’s complement: 1011**

**Week 3**

1. **What is a function prototype and why do you need that? Explain with example(s).**
   * **Function Prototype: Declares the function’s name, return type, and parameters before the actual definition, enabling the compiler to validate function calls.**
     + **Example:**

**void greet(); // Function prototype**

**int main() {**

**greet();**

**return 0;**

**}**

**void greet() {**

**cout << "Hello, World!";**

**}**

1. **What is function overloading, since a function can have the following parts: a function name, a return type, argument(s), function body. What can we change or cannot change in the overloaded function(s)?**
   * **Function Overloading: Defining multiple functions with the same name but different parameter lists.**
   * **Changes allowed: Different parameters (types/number), but not the return type alone.**
2. **Discuss with example default arguments in function.**
   * **Default Arguments: Provide pre-set values in case no argument is passed.**
     + **Example:**

**void display(int x = 10) {**

**cout << x << endl;**

**}**

**int main() {**

**display(); // Outputs: 10**

**display(20); // Outputs: 20**

**return 0;**

**}**

1. **Discuss function arguments passing by value and passing by reference.**
   * **Pass by Value: The function gets a copy of the argument, and changes do not affect the original.**
   * **Pass by Reference: The function gets a reference, and changes reflect on the original variable.**
2. **What is an inline function and what is the benefit/use of inline function.**
   * **Inline Function: Suggests to the compiler to replace the function call with the actual function code, reducing function call overhead.**
     + **Example:**

**inline int add(int a, int b) {**

**return a + b;**

**}**

1. **Review the scope of variables.**
   * **Variable Scope: Can be local (within a function/block) or global (accessible throughout the program).**

**Week 4**

1. **Discuss LValue and RValue with example.**
   * **LValue: Represents a memory location and is modifiable.**
   * **RValue: Represents a data value stored in memory.**
     + **Example:**

**int x = 5; // 'x' is LValue, '5' is RValue**

1. **Discuss correctness and incorrectness of increment & decrement operations.**
   * **Example:**

**int a = 5;**

**int b = ++a; // Correct**

**int c = a++; // Correct**

1. **Discuss operator precedence, associativity, and evaluation order.**
   * **Precedence: Determines which operator is evaluated first.**
   * **Associativity: Determines the order of operations (left-to-right or right-to-left).**
2. **If you pass an array as function argument, is it passed by value or by reference? Discuss the reasoning behind such behavior.**
   * **Passed by Reference: The function receives a pointer to the array’s first element, allowing direct access to the original array elements.**

**Week 5**

1. **Given a code snippet for pointer operations, show the values or states of the memory locations.**
   * **Example:**

**int x = 10;**

**int\* p = &x;**

**\*p = 20;**

**// Now, x = 20.**

1. **Discuss passing an argument by reference using pointer in C++.**
   * **Example:**

**void update(int\* p) {**

**\*p = 5;**

**}**

**int main() {**

**int x = 3;**

**update(&x); // x is now 5**

**return 0;**

**}**

1. **What could be the values of different variables for different operations?**
   * **Example:**

**int x = 5, y = 10;**

**int \*p = &x, \*q = &y;**

**\*p = \*q;**

**cout << x << " " << y << endl;  // Output: 10 10**

1. **Review the code examples.**
   * **Example:**

**int arr[3] = {1, 2, 3};**

**int \*p = arr;**

**for (int i = 0; i < 3; i++) {**

**cout << \*(p + i) << " ";  // Output: 1 2 3**

**}**

**Week 6**

1. **What is structure and why do we need structure?**
   * **Structure: A user-defined data type that groups different data types together, useful for organizing complex data (e.g., student records).**
     + **Example:**

**struct Person {**

**string name;**

**int age;**

**};**

**int main() {**

**Person person;**

**person.name = "John";**

**person.age = 30;**

**cout << person.name << " is " << person.age << " years old." << endl;**

**return 0;**

**}**

1. **What are some features of structures?**
   * **Features:**
     + **Can contain variables of different types.**
     + **Can be nested within other structures.**
     + **Can have functions as members.**
     + **Can be passed to functions by value or by reference.**
     + **Example:**

Programming Problems to Practice:

1. You are given an array of integers arr and the size of the array sz. You need to write a function to reverse the order of the integers in the array.

Input: arr={9, 4, 8, 0, 1, 0, 6}, sz = 7

Output: arr={6, 0, 1, , 8, 4, 9}

#include <iostream>

using namespace std;

void reverseArray(int arr[], int sz) {

    int start = 0;

    int end = sz - 1;

    int temp;

    // Loop to swap elements from the start and end of the array

    while (start < end) {

        // Swap the elements at the start and end positions

        temp = arr[start];

        arr[start] = arr[end];

        arr[end] = temp;

        // Move the start index forward and the end index backward

        start++;

        end--;

    }

}

int main() {

    int arr[] = {9, 4, 8, 0, 1, 0, 6};

    int sz = sizeof(arr) / sizeof(arr[0]);

    // Print the original array

    cout << "Original array: ";

    for (int i = 0; i < sz; i++) {

        cout << arr[i] << " ";

    }

    // Call the function to reverse the array

    reverseArray(arr, sz);

    // Print the reversed array

    cout << "\nReversed array: ";

    for (int i = 0; i < sz; i++) {

        cout << arr[i] << " ";

    }

    return 0;

}

1. You are given an input array (arr\_in) and output array (arr\_out) and the size of both arrays sz, where there are 0s and non-zero integer values. You need to left-align the non-zero elements of the input array and put them in the output array while maintaining the relative order, all the 0s will be on the right in the output array. For instance, if the input array is {5, 3, 0, 0, 9, 1, 2, 0, 4} and sz is 9, your output array will look like this {5, 3, 9, 1, 2, 4, 0, 0, 0}.

#include <iostream>

using namespace std;

void leftAlignNonZero(int arr\_in[], int arr\_out[], int sz) {

    int index = 0;

    // Copy non-zero elements to arr\_out

    for (int i = 0; i < sz; i++) {

        if (arr\_in[i] != 0) {

            arr\_out[index++] = arr\_in[i];

        }

    }

    // Fill the rest of arr\_out with zeros

    while (index < sz) {

        arr\_out[index++] = 0;

    }

}

int main() {

    int arr\_in[] = {5, 3, 0, 0, 9, 1, 2, 0, 4};

    int sz = sizeof(arr\_in) / sizeof(arr\_in[0]);

    int arr\_out[sz];

    leftAlignNonZero(arr\_in, arr\_out, sz);

    cout << "Output array: ";

    for (int i = 0; i < sz; i++) {

        cout << arr\_out[i] << " ";

    }

    return 0;

}

1. **Bonus problem**: can you solve problem 2 without the output array, i.e., modify the input array while maintaining the relative order of the non-zero elements and return it as the output.

#include <iostream>

using namespace std;

void leftAlignNonZeroInPlace(int arr[], int sz) {

    int index = 0;  // Position to place the next non-zero element

    // Move non-zero elements to the front of the array

    for (int i = 0; i < sz; i++) {

        if (arr[i] != 0) {

            arr[index++] = arr[i];

        }

    }

    // Fill the remaining positions with zeros

    while (index < sz) {

        arr[index++] = 0;

    }

}

int main() {

    int arr[] = {5, 3, 0, 0, 9, 1, 2, 0, 4};

    int sz = sizeof(arr) / sizeof(arr[0]);

    // Call the function to modify the array

    leftAlignNonZeroInPlace(arr, sz);

    // Print the modified array

    cout << "Modified input array: ";

    for (int i = 0; i < sz; i++) {

        cout << arr[i] << " ";

    }

    return 0;

}

1. Given an input positive integer, compute the factorial of that number. Can you do the computation using recursive function.

#include <iostream>

using namespace std;

// Iterative function to compute factorial

int factorial(int n) {

    int result = 1;  // Initialize result to 1

    // Multiply result by each number from 2 to n

    for (int i = 2; i <= n; i++) {

        result \*= i;

    }

    return result;  // Return the computed factorial

}

int main() {

    int num;

    cout << "Enter a positive integer: ";

    cin >> num;

    if (num < 0) {

        cout << "Factorial is not defined for negative numbers." << endl;

    } else {

        cout << "Factorial of " << num << " is: " << factorial(num) << endl;

    }

    return 0;

}

1. Compute Fibonacci series recursively.

#include <iostream>

using namespace std;

// Iterative function to compute Fibonacci numbers

void fibonacci(int num) {

    int a = 0, b = 1, next;

    // Print the first two terms

    if (num >= 1) cout << a << " ";

    if (num >= 2) cout << b << " ";

    // Compute and print the remaining terms

    for (int i = 3; i <= num; i++) {

        next = a + b;  // Calculate the next term

        cout << next << " ";  // Print the next term

        a = b;  // Update a to the previous term

        b = next;  // Update b to the current term

    }

}

int main() {

    int num;

    cout << "Enter the number of terms for Fibonacci series: ";

    cin >> num;

    cout << "Fibonacci series: ";

    fibonacci(num);  // Call the function to print the Fibonacci series

    return 0;

}

1. Given a positive integer, write a function to check if that number is prime number.

#include <iostream>

using namespace std;

// Function to check if a number is prime

bool isPrime(int n) {

    if (n <= 1) {  // Numbers less than or equal to 1 are not prime

        return false;

    }

    // Check divisibility from 2 to sqrt(n)

    for (int i = 2; i \* i <= n; i++) {

        if (n % i == 0) {

            return false;  // If divisible by any number other than 1 and itself, not prime

        }

    }

    return true;  // If no divisors are found, it is prime

}

int main() {

    int num;

    cout << "Enter a positive integer: ";

    cin >> num;

    if (isPrime(num)) {

        cout << num << " is a prime number." << endl;

    } else {

        cout << num << " is not a prime number." << endl;

    }

    return 0;

}

1. Given a string, write a function to check whether the string is a palindrome.

#include <iostream>

#include <string>

using namespace std;

// Function to check if a string is a palindrome

bool isPalindrome(string str) {

    int start = 0;  // Start index

    int end = str.length() - 1;  // End index

    // Compare characters from the beginning and end of the string

    while (start < end) {

        if (str[start] != str[end]) {

            return false;  // If characters don't match, it's not a palindrome

        }

        start++;

        end--;

    }

    return true;  // If all characters match, it's a palindrome

}

int main() {

    string str;

    cout << "Enter a string: ";

    cin >> str;

    // Check if the string is a palindrome

    if (isPalindrome(str)) {

        cout << str << " is a palindrome." << endl;

    } else {

        cout << str << " is not a palindrome." << endl;

    }

    return 0;

}

1. Given a positive number, write a function to verify whether the number is a perfect number.

#include <iostream>

using namespace std;

// Function to check if a number is a perfect number

bool isPerfectNumber(int n) {

    if (n <= 1) {  // No perfect numbers less than or equal to 1

        return false;

    }

    int sum = 0;

    // Find divisors and sum them (excluding the number itself)

    for (int i = 1; i <= n / 2; i++) {

        if (n % i == 0) {

            sum += i;

        }

    }

    // If the sum of divisors equals the number, it is perfect

    return sum == n;

}

int main() {

    int num;

    cout << "Enter a positive integer: ";

    cin >> num;

    if (isPerfectNumber(num)) {

        cout << num << " is a perfect number." << endl;

    } else {

        cout << num << " is not a perfect number." << endl;

    }

    return 0;

}

1. Given a string, return the reverse of the string.

#include <iostream>

#include <string>

using namespace std;

// Function to reverse a string

string reverseString(string str) {

    int start = 0;

    int end = str.length() - 1;

    // Swap characters from both ends of the string

    while (start < end) {

        swap(str[start], str[end]);

        start++;

        end--;

    }

    return str;

}

int main() {

    string str;

    cout << "Enter a string: ";

    cin >> str;

    string reversedStr = reverseString(str);

    cout << "Reversed string: " << reversedStr << endl;

    return 0;

}

1. Practice and understand the Problem given in you Programming Lab 2 and 3.
2. Practice the programming problems on slide 44 (pop quiz) of Week 3.

1)

1. #include <iostream>

using namespace std;

1. int main() {
2. int n;
3. cout << "Enter a positive integer (1 to 100): ";
4. cin >> n;
5. cout << "Even numbers up to " << n << " are: ";
6. for (int i = 2; i <= n; i += 2) {
7. cout << i;
8. if (i + 2 <= n) {
9. cout << ", ";
10. }
11. }
12. return 0;
13. }

2)

2. #include <iostream>

using namespace std;

int main() {

    int n, square = 0;  // Declare variables

    cout << "Enter a positive integer: ";

    cin >> n;

    // Calculate the square of n by adding n to itself n times

    for (int i = 1; i <= n; ++i) {

        square += n;

    }

    // Print the result

    cout << "The square of " << n << " is: " << square << endl;

    return 0;

}

3)

#include <iostream>

using namespace std;

int main() {

    int dividend, divisor, quotient = 0, remainder;  // Declare variables

    cout << "Enter two positive integers (dividend and divisor): ";

    cin >> dividend >> divisor;

    remainder = dividend;  // Initialize remainder to dividend

    // Calculate quotient and remainder using repeated subtraction

    while (remainder >= divisor) {

        remainder -= divisor;  // Subtract divisor from remainder

        ++quotient;  // Increment quotient

    }

    // Print the result

    cout << "Quotient: " << quotient << ", Remainder: " << remainder << endl;

    return 0;

}