**Four C++ Questions of 14/08/2024:**

Q1. Implement a custom dynamic array class that supports basic operations like insertion, deletion, resizing, and clearing.

Answer:

#include <bits/stdc++.h>

using namespace std;

template <typename T> class vectorClass {

    // arr is the integer pointer

    // which stores the address of our vector

    T\* arr;

    // capacity is the total storage

    // capacity of the vector

    int capacity;

    // current is the number of elements

    // currently present in the vector

    int current;

public:

    // Default constructor to initialise

    // an initial capacity of 1 element and

    // allocating storage using dynamic allocation

    vectorClass()

    {

        arr = new T[1];

        capacity = 1;

        current = 0;

    }

    // destructor to deallocate storage allocated by dynamic

    // allocation to prevent memory leak

    ~vectorClass() { delete[] arr; }

    // Function to add an element at the last

    void push(T data)

    {

        // if the number of elements is equal to the

        // capacity, that means we don't have space to

        // accommodate more elements. We need to double the

        // capacity

        if (current == capacity) {

            T\* temp = new T[2 \* capacity];

            // copying old array elements to new array

            for (int i = 0; i < capacity; i++) {

                temp[i] = arr[i];

            }

            // deleting previous array

            delete[] arr;

            capacity \*= 2;

            arr = temp;

        }

        // Inserting data

        arr[current] = data;

        current++;

    }

    // function to add element at any index

    void push(T data, int index)

    {

        // if index is equal to capacity then this

        // function is same as push defined above

        if (index == capacity)

            push(data);

        else

            arr[index] = data;

    }

    // function to extract element at any index

    T get(int index)

    {

        // if index is within the range

        if (index < current)

            return arr[index];

        // if index is not within range

        return -1;

    }

    // function to delete last element

    void pop() { current--; }

    // function to get size of the vector

    int size() { return current; }

    // function to get capacity of the vector

    int getcapacity() { return capacity; }

    // function to print array elements

    void print()

    {

        for (int i = 0; i < current; i++) {

            cout << arr[i] << " ";

        }

        cout << endl;

    }

};

// Driver code

int main()

{

    vectorClass<int> v;

    vectorClass<char> v1;

    v.push(10);

    v.push(20);

    v.push(30);

    v.push(40);

    v.push(50);

    v1.push(71);

    v1.push(72);

    v1.push(73);

    v1.push(74);

    cout << "Vector size : " << v.size() << endl;

    cout << "Vector capacity : " << v.getcapacity() << endl;

    cout << "Vector elements : ";

    v.print();

    v.push(100, 1);

    cout << "\nAfter updating 1st index" << endl;

    cout << "Vector elements of type int : " << endl;

    v.print();

    // This was possible because we used templates

    cout << "Vector elements of type char : " << endl;

    v1.print();

    cout << "Element at 1st index of type int: " << v.get(1)

         << endl;

    cout << "Element at 1st index of type char: "

         << v1.get(1) << endl;

    v.pop();

    v1.pop();

    cout << "\nAfter deleting last element" << endl;

    cout << "Vector size of type int: " << v.size() << endl;

    cout << "Vector size of type char: " << v1.size()

         << endl;

    cout << "Vector capacity of type int : "

         << v.getcapacity() << endl;

    cout << "Vector capacity of type char : "

         << v1.getcapacity() << endl;

    cout << "Vector elements of type int: ";

    v.print();

    cout << "Vector elements of type char: ";

    v1.print();

    return 0;

}

Q2. Create a template-based stack class supporting push, pop, and peek operations. Implement it for different data types like int, float, and std::string.

Answer:

#include <iostream>

#include <stack>

using namespace std;

int main()

{

    // Empty stack

    stack<int> mystack;

    mystack.push(0);

    mystack.push(1);

    mystack.push(2);

    // Printing content of stack

    while (!mystack.empty()) {

        cout << ' ' << mystack.top();

        mystack.pop();

    }

}

Q3. Write a program that reads from a file and handles various exceptions such as file not found, read errors, and unexpected data formats.

Answer:

#include <iostream>

#include <stdexcept>

using namespace std;

int main()

{

    // try block

    try {

        int numerator = 10;

        int denominator = 0;

        int res;

        // check if denominator is 0 then throw runtime

        // error.

        if (denominator == 0) {

            throw runtime\_error(

                "Division by zero not allowed!");

        }

        // calculate result if no exception occurs

        res = numerator / denominator;

        //[printing result after division

        cout << "Result after division: " << res << endl;

    }

    // catch block to catch the thrown exception

    catch (const exception& e) {

        // print the exception

        cout << "Exception " << e.what() << endl;

    }

    return 0;

}