SUNNY KUMAR
C++ PROGRAM
3 JULY 2024

```
/*******************************************************************************
Problem Statement:

Write a generic function template named findMinimum in C++ that takes an array of any data
type T and its size n as arguments. The function should return the minimum element present in
the array.

*******************************************************************************/
#include <iostream>
#include <limits>
using namespace std;
template<typename T>
T findMinimum(T arr[], int n) {
    if (n<=0) {
        cout<<"Array size must be greater than 0";
    }

    T minElement=numeric_limits<T>::max();
    for (int i=0;i<n;i++) {
        if (arr[i]<minElement) {
            minElement=arr[i];
        }
    }
    return minElement;
}

int main() {

    double arr[] = {2.5, 3.7, 1.8, 4.4, 0.9};
    cout<<"Minimum element in double array: "<<findMinimum(arr, 5)<<endl;
    return 0;
}



/*******************************************************************************
```

Problem: Write a function template swap that takes two pointers to variables of any data type T and swaps their values.

Constraints: The function should only modify the values pointed to by the arguments, not the arguments themselves (pass by reference).
**********/

```
#include <iostream>
using namespace std;
template <typename T>
void swap(T* a,T* b) {
   T temp=*a;
   *a=*b;
   *b=temp;
}
int main() {
   double p=3.1, q=2.7;
   swap(&p,&q);
   cout<<"After swap the value is: p is="<<p<<",q is= "<<q<<endl;
   return 0;
}
```

/**********

Problem: Similar to findMinimum, create a function template findMaximum that returns the maximum element in an array of any data type T.
**********/

```
#include <iostream>
using namespace std;
template <typename T>
T findMaximum(T arr[],int n) {
   if (n<=0) {
      cout<<"Array size must be greater than 0";
   }
   T maxElement=arr[0];
   for (int i=1;i<n;i++) {
      if (arr[i]>maxElement) {
         maxElement=arr[i];
      }
   }
   return maxElement;
}
```

```
int main() {
   int arr1[]={5, 3, 8, 6, 2, 7};
   cout<<"Maximum element inarray 1: "<<findMaximum(arr1, 6)<<endl;


   return 0;
}
```

Class template

[1]

```
#include<iostream>
using namespace std;
template<class T1, class T2>
class A{
    T1 a;
    T2 b;
    public:
    A(T1 x, T2 y)
    {
        a=x,b=y;
    }
    void dis()
    {
        cout<<"values of a and b is "<<a<<" "<<b<<endl;
    }
};
int main()
{
    A<int ,float> d(4,4.6);
    d.dis();
    return 0;
}
```

[2]

```
#include <iostream>
using namespace std;
template<typename T>
class A
{
   int num1=5;
   int num2=8;
   void add()
   {
      cout<<"addition of two number is: "<<num2+num1;
   }
};


int main() {

   A<int>d;
   d.add;
   return 0;
}
```

[3]

```
#include <iostream>
using namespace std;
template<typename T>
class A
{
   int num1=5;
   int num2=8;
   void add()
   {
```

```
        cout<<"addition of two number is: "<<num2+num1;
    }
};


int main() {

    A<int>d;
    d.add;
    return 0;
}
```

/*Design a generic data processing library using class and function templates in C++. This library should be able to handle various data types (e.g., integers, floats, strings) without code duplication.

Requirements:

1.Create a class template named DataContainer that can hold elements of any data type
specified during instantiation.
2.Implement member functions for DataContainer:
3.DataContainer(size_t size): Constructor to initialize the container with a specific size.
4.T& operator[](size_t index): Overloaded subscript operator to access elements.
5.void printAll(): Prints all elements of the container.
6.Create a function template named swap that takes two DataContainer objects
as arguments and swaps their elements.
7.Ensure proper memory management using appropriate constructors and destructors.*/

```
#include <iostream>
using namespace std;
```

```
template<typename T>
class DataContainer {
    public:
    T* data;
    size_t size;
public:
    DataContainer(size_t size) : size(size) {
        data=new T[size];
    }
    ~DataContainer() {
        delete[] data;
    }
    T& operator[](size_t index) {
        return data[index];
    }

    // Function to print all elements of the container
    void printAll() const {
        for (size_t i=0; i<size; ++i) {
            cout<<data[i]<<" ";
        }
        cout << endl;
    }
};

// Function template to swap two DataContainer objects
template <typename T>
void swap(DataContainer<T>& a, DataContainer<T>& b) {
    if (a.size!=b.size) {
        cout<<"both container size unequal"<<endl;
    }

    for (size_t i=0;i<a.size;++i) {
        T temp=a[i];
        a[i]=b[i];
```

```
        b[i]=temp;
    }
}

int main() {
    // Create instances of DataContainer for different data types
    DataContainer<int> intContainer(5);
    DataContainer<float> floatContainer(5);

    // Populate the containers with sample data
    for (size_t i=0;i<5;i++) {
        intContainer[i] = i * 2;
        floatContainer[i] = i * 1.5f;
    }

    // Print all elements of each container
    cout<<"Integer container: ";
    intContainer.printAll();

    cout<<"Float container: ";
    floatContainer.printAll();

    // Swap elements between containers of the same type
    DataContainer<int> intContainer2(5);
    for (size_t i=0;i<5;i++) {
        intContainer2[i]=i + 10;
    }
    swap(intContainer, intContainer2);
    cout<<"After swap:"<<endl;
    cout<<"Integer container 1: ";
    intContainer.printAll();
    cout<<"Integer container 2: ";
    intContainer2.printAll();

    return 0;
```

```
}

#include <iostream>
#include <vector>
using namespace std;
template<typename T>
class DataContainer {
private:
    vector<T> data;

public:
    DataContainer(size_t size = 0) {
        data.resize(size);
    }

    T& operator[](size_t index) {
        if (index >= data.size()) {
            throw out_of_range("Index out of range");
        }
        return data[index];
    }

    void printAll() const {
        for (const auto& element : data) {
            cout << element << " ";
        }
        cout << endl;
    }

    size_t size() const {
        return data.size();
    }
    void push_back(const T& value) {
        data.push_back(value);
    }
```

```
    friend void swap(DataContainer<T>& a, DataContainer<T>& b) {
        a.data.swap(b.data);
    }
};

int main() {
    DataContainer<int> intContainer1(5);
    DataContainer<int> intContainer2(5);

    for (size_t i = 0; i < 5; ++i) {
        intContainer1[i] = i * 2;
        intContainer2[i] = i * 3;
    }

    cout << "Integer container 1 before swap:" << endl;
    intContainer1.printAll();

    cout << "Integer container 2 before swap:" << endl;
    intContainer2.printAll();

    swap(intContainer1, intContainer2);

    cout << "Integer container 1 after swap:" << endl;
    intContainer1.printAll();

    cout << "Integer container 2 after swap:" << endl;
    intContainer2.printAll();
    DataContainer<double> doubleContainer(3);
    for (size_t i = 0; i < 3; ++i) {
        doubleContainer[i] = i * 1.5;
    }
    cout << "Double container:" << endl;
    doubleContainer.printAll();
```

```
    return 0;
}
```

SMART POINTER

```
#include <iostream>
using namespace std;

// A generic smart pointer class template
template <class T>
class SmartPointer {
    T *p; // Actual pointer
public:
    // Constructor
    SmartPointer(T *ptr = NULL) {
        p = ptr;
    }

    // Destructor
    ~SmartPointer() {
        delete(p);
    }

    // Overloading dereferencing operator
    T &operator*() {
        return *p;
    }

    // Overloading arrow operator so that members of T can be accessed
    T *operator->() {
        return p;
    }
};

int main() {
```

```
    SmartPointer<int> P(new int());
    *P = 26;
    cout << "Value is: " << *P << endl;
    return 0;
}
```

/*use abstract classes and polymorphism in C++ for calculating the areas of various shapes*/

```
#include <iostream>
#include <cmath>

class Shape {
public:
    virtual double area() const = 0;
    virtual double perimeter() const = 0;
    virtual ~Shape() = default;
};

class Circle : public Shape {
private:
    double radius;

public:
    Circle(double r) : radius(r) {}

    double area() const override {
        return M_PI * radius * radius;
    }

    double perimeter() const override {
        return 2 * M_PI * radius;
    }
};
```

```
class Rectangle : public Shape {
private:
    double width, height;

public:
    Rectangle(double w, double h) : width(w), height(h) {}

    double area() const override {
        return width * height;
    }

    double perimeter() const override {
        return 2 * (width + height);
    }
};

void printShapeDetails(const Shape& shape) {
    std::cout << "Area: " << shape.area() << std::endl;
    std::cout << "Perimeter: " << shape.perimeter() << std::endl;
}

int main() {
    Circle circle(5.0);
    Rectangle rectangle(4.0, 6.0);

    std::cout << "Circle details:" << std::endl;
    printShapeDetails(circle);

    std::cout << "Rectangle details:" << std::endl;
    printShapeDetails(rectangle);

    return 0;
}
```

/*
Inventory Management System:
Problem: Design a system to manage inventory for various products.
Each product might have different attributes (name, price, quantity) and potentially
unique functionalities (e.g., perishable items with an expiry date).

what we have to do in step wise

1.define the product class and attribute is name ,price,quantity and add a function where we can display
and addd a method to update the quantity and get the name of that Product

2.define the perishable class which is inherit from base class and attribute is expiry date and override
the display fn to overridethe expiryDate

3.define the inventory calss

*/

```
#include <iostream>
#include <vector>
#include <string>
#include <algorithm> // For std::remove_if

class Product {
protected:
    std::string name;
    double price;
    int quantity;
```

```
public:
    Product(const std::string& name, double price, int quantity)
        : name(name), price(price), quantity(quantity) {}

    virtual ~Product() = default;

    virtual void display() const {
        std::cout << "Name: " << name << "\nPrice: " << price << "\nQuantity: "
<< quantity << std::endl;
    }

    virtual void updateQuantity(int amount) {
        quantity += amount;
    }

    virtual std::string getName() const {
        return name;
    }
};

class PerishableProduct : public Product {
private:
    std::string expiryDate;

public:
    PerishableProduct(const std::string& name, double price, int quantity,
const std::string& expiryDate)
        : Product(name, price, quantity), expiryDate(expiryDate) {}

    void display() const override {
        Product::display();
        std::cout << "Expiry Date: " << expiryDate << std::endl;
    }

    std::string getExpiryDate() const {
```

```
        return expiryDate;
    }
};

class Inventory {
private:
    std::vector<Product*> products;

public:
    ~Inventory() {
        for (auto product : products) {
            delete product;
        }
    }

    void addProduct(Product* product) {
        products.push_back(product);
    }

    void removeProduct(const std::string& name) {
        auto it = std::remove_if(products.begin(), products.end(),
                                 [&name](Product* product) { return
product->getName() == name; });
        if (it != products.end()) {
            delete *it;
            products.erase(it, products.end());
        }
    }

    void displayInventory() const {
        for (const auto& product : products) {
            product->display();
            std::cout << "------------------------" << std::endl;
        }
    }
```

```
    void updateProductQuantity(const std::string& name, int amount) {
        for (auto& product : products) {
            if (product->getName() == name) {
                product->updateQuantity(amount);
                return;
            }
        }
    }
};

int main() {
    Inventory inventory;

    // Add some products
    inventory.addProduct(new Product("Laptop", 1200.99, 10));
    inventory.addProduct(new PerishableProduct("Milk", 2.99, 20,
"2024-07-10"));

    // Display inventory
    std::cout << "Initial Inventory:" << std::endl;
    inventory.displayInventory();

    // Update quantity of a product
    inventory.updateProductQuantity("Laptop", -2);

    // Display inventory after update
    std::cout << "Inventory after updating quantity:" << std::endl;
    inventory.displayInventory();

    // Remove a product
    inventory.removeProduct("Milk");

    // Display inventory after removal
    std::cout << "Inventory after removing a product:" << std::endl;
```

```
    inventory.displayInventory();

    return 0;
}
```

SMART POINTER

```
#include <iostream>
#include <memory>
using namespace std;
class MyClass {
public:
    void display() {
        cout << "Hi everyone my name is sunny kumar sharma" << endl;
    }
};
int main() {
    unique_ptr<MyClass> ptr(new MyClass());

    ptr->display();

    return 0;
}
```