**MINISTERUL EDUCAȚIEI ȘI CERCETĂRII AL REPUBLICII MOLDOVA**

**Universitatea Tehnică a Moldovei**

**RAPORT**

Lucrare de laborator nr. 5

la cursul ***„Programarea orientată pe obiecte”***

***Tema: Funcţii şi clase generice***

**Varianta 21**

**A efectuat :**   **St. gr. CR-221FR Serba Cristina**

**A verificat: asist. univ. Mantaluța Marius**

**Chișinău 2024**

# Sarcina lucrării:

а) Creaţi o funcţie generică (şablon), care caută o cheie dată. Funcţia întoarce poziţia primului

element întâlnit. De exemplu: lista - 0 2 3 4 3 6, parametrul - 2, rezultatul - 1. În cazul când

lipseşte elementul necesar să se întoarcă codul erorii. Funcţia trebuie să lucreze cu tablouri

unidimensionale de lungimi diferite.

b) Creaţi clasa generică (parametrizată) Queue - coadă. Clasa trebuie să conţină constructorii,

destructorii şi funcţiile add, in, get, getLength, operatorii [] şi operatorii de intrare/ieşire.

# Mersul programului:

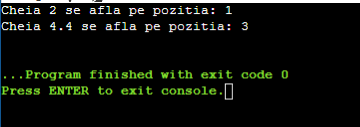
a)

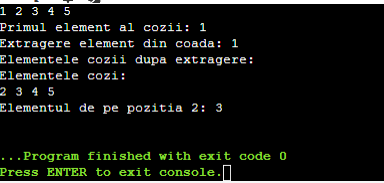
|  |
| --- |
| #include <iostream>  template <typename T>  int cautare(const T\* tablou, int lungime, const T& cheie) {  for (int i = 0; i < lungime; ++i) {  if (tablou[i] == cheie) {  return i; // Returneaza pozitia primei aparitii a cheii in tablou  }  }  return -1; // Returneaza codul de eroare daca cheia lipseste in tablou  }  int main() {  // Test cu tipul de date int  int tablouInt[] = {0, 2, 3, 4, 3, 6};  int pozitieInt = cautare(tablouInt, 6, 2);  if (pozitieInt != -1) {  std::cout << "Cheia 2 se afla pe pozitia: " << pozitieInt << std::endl;  } else {  std::cout << "Cheia nu a fost gasita." << std::endl;  }  // Test cu tipul de date double  double tablouDouble[] = {1.1, 2.2, 3.3, 4.4, 5.5};  int pozitieDouble = cautare(tablouDouble, 5, 4.4);  if (pozitieDouble != -1) {  std::cout << "Cheia 4.4 se afla pe pozitia: " << pozitieDouble << std::endl;  } else {  std::cout << "Cheia nu a fost gasita." << std::endl;  }  return 0;  } |

b)

|  |
| --- |
| #include <iostream>  using namespace std;  template <typename T>  class Queue {  private:  T\* elements; // Pointer catre tabloul de elemente  int capacity; // Capacitatea cozi  int front; // Indicele primului element din coada  int rear; // Indicele ultimului element din coada  int length; // Lungimea actuala a cozi  public:  // Constructor  Queue(int size = 10)  {  capacity = size;  front = 0;  rear = -1;  length = 0;  elements = new T[capacity];  }  // Destructor  ~Queue()  {  delete[] elements;  }  // Functia pentru adaugarea unui element in coada  void add(const T& element)  {  if (length < capacity)  {  rear = (rear + 1) % capacity; // Urmatorul element va fi ultimul din coada  elements[rear] = element;  length++;  } else  {  cout << "Coada plina! Elementul nu a fost adaugat." << endl;  }  }  // Functia pentru extragerea unui element din coada  T get()  {  if (!isEmpty())  {  T element = elements[front];  front = (front + 1) % capacity; // Primul element existent va iesi  length--;  return element;  } else  {  cout << "Coada goala! Nu exista elemente de extras." << endl;  return T();  }  }  // Functia pentru a obtine primul element din coada fara a-l sterge  T peek() const  {  if (!isEmpty())  {  return elements[front];  } else  {  cerr << "Coada goala! Nu exista elemente de extras." << endl;  return T();  }  }  // Functia pentru a verifica daca coada este goala  bool isEmpty() const  {  return (0 == length);  }  // Functia pentru a obtine lungimea curenta a cozii  int getLength() const  {  return length;  }  // Supraincarcarea operatorului []  T& operator[](int index)  {  if (index >= 0 && index < length)  {  return elements[(front + index) % capacity]; // Elementul la pozitia index de la front  } else  {  cout << "Index invalid!" << endl;  exit(1);  }  }  // Supraincarcarea operatorului de afisare  friend ostream& operator<<(ostream& out, const Queue<T>& q)  {  out << "Elementele cozi:" << endl;  if (!q.isEmpty())  {  for (int i = 0; i < q.length; i++)  {  out << q.elements[(q.front + i) % q.capacity] << " "; // Trece prin fiecare pozitie i de la front  }  } else  {  out << "Coada goala!";  }  return out;  }  };  int main()  {  Queue<int> q(5);  q.add(1);  q.add(2);  q.add(3);  q.add(4);  q.add(5);  cout << "Coada initiala:" << endl;  cout << q << endl;  cout << "Primul element al cozii: " << q.peek() << endl;  cout << "Extragere element din coada: " << q.get() << endl;  cout << "Elementele cozii dupa extragere:" << endl;  cout << q << endl;  cout << "Elementul de pe pozitia 2: " << q[1] << endl;  return 0;  } |

# Rezultatele obținute:

a)

b)

# Concluzii:

Funcţiile şi clasele generice, cunoscute şi sub numele de şabloane în limbajul de programare C++, reprezintă un instrument esențial în dezvoltarea de cod flexibil și eficient. Scopul acestei lucrări a fost de a explora în profunzime necesitatea, definirea, utilizarea și specializarea şabloanelor, precum și de a evidenția potenţialele probleme care pot fi rezolvate cu ajutorul lor. Concluzionând, funcţiile şi clasele generice, prin intermediul şabloanelor în C++, oferă o abordare flexibilă și puternică pentru dezvoltarea de software. Utilizarea lor corespunzătoare poate duce la creșterea eficienței și ușurinței în dezvoltarea și întreținerea codului, oferind în același timp posibilitatea de a face față unor probleme diverse într-un mod generalizat și elegant.