Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)
Кафедра экономической математики, информатики и статистики (ЭМИС)

ШАБЛОНЫ КЛАССОВ

Отчет по лабораторной работе по дисциплине «Объектно-ориентированное программирование»

Студент группы 549

 Баулин С.К.

«__» _____ 2021 г.

Кандидат физико-математических наук, доцент кафедры ЭМИС

_____ _____ Шельмина Е. А.

оценка «__» _____ 2021 г.

Томск 2021

Лабораторная работа №9

Шаблоны классов

Цель работы: освоить и применить на практике шаблоны классов

Теоритические сведения

Рассматриваемый далее пример - класс, который хранит пару значений. Функции-элементы этого класса возвращают минимальное и максимальное значения, а также позволяют определить, являются ли два значения одинаковыми. Поскольку перед нами шаблон класса, то тип значения может быть почти любым.

```
template <class T>
class Pair
{
T a, b;
public:
Pair (T t1, T t2);
T Max();
T Min ();
int isEqual ();
};
```

Пока все выглядит также , как и для шаблонов функций. Единственная разница состоит в том, что вместо описания функции используется объявление класса. Шаблоны классов становятся все более сложными, когда вы описываете принадлежащие функции класса. Вот, например, описание принадлежащей функции Min() класса Pair:

```
template <class T>
T Pair <T>::Min()
{return a < b ? a : b;}
```

Чтобы понять эту запись, давайте вернемся немного назад. Если бы Pair был обычным классом (а не шаблоном класса) и T был бы некоторым

конкретным типом, то функция Min класса Pair была бы описана таким образом:

```
T Pair::Min()
{return a < b ? a : b;}
```

Для случая шаблонной версии нам необходимо, во-первых, добавить заголовок шаблона template <class T>.

Затем нужно дать имя классу. Помните, что на самом деле мы описываем множество классов - семейство Pair. Повторяя синтаксис префикса (заголовка) шаблона, экземпляр класса Pair для целых типов, можно назвать Pair<int>, экземпляр для типа double - Pair<double>, для типа Vector - Pair<Vector>. Однако в описании принадлежащей классу функции нам необходимо использовать имя класса Pair<T>. Это имеет смысл, так как заголовок шаблона говорит, что T означает имя любого типа.

Ранее уже отмечалось, что шаблоны функций могут работать только для тех (встроенных) типов данных или классов, которые поддерживают необходимые операции. То же самое справедливо и для шаблонов классов. Чтобы создать экземпляр класса Pair для некоторого классового типа, например для класса X, этот класс должен содержать следующие общедоступные функции:

```
X (X &);// конструктор копирования
int operator == (X)
int operator < (X);
```

Три указанные функции необходимы, так как они реализуют операции, выполняемые над объектами типа T в метода X шаблона класса Pair.

Если вы собираетесь использовать некоторый шаблон класса, как узнать какие операции требуются? Если шаблон класса снабжен документацией, то эти требования должны быть в ней указаны. В противном случае придется читать первичную документацию - исходный текст шаблона. При этом

учитывайте, что встроенные типы данных поддерживают все стандартные операции.

Шаблоны классов не только для типов

Параметризовать некоторый класс так, чтобы он работал для любого типа данных - это только половина того, что шаблоны обеспечивают для классов. Другой аспект состоит в том, чтобы дать возможность задания числовых параметров. Это позволяет Вам, например, создавать объекты типов "Вектор из 20 целых", "Вектор из 1000 целых" или "Вектор из 10 переменных типа double".

Основная идея проста, хотя используемый синтаксис может показаться сложным. Давайте в качестве примера рассмотрим некоторый обобщенный класс Vector. Как и класс Pair, класс Vector содержит функции Min(), Max(), isEqual(). Однако в нем может быть любое количество участников, а не два. В классе Pair число участников фиксировано и задаются они в качестве аргументов конструктора. В шаблоне Vector вместо этого используется второй параметр заголовка шаблона.

Имеется несколько вариантов использования шаблонов с параметрами-значениями для динамического размещения массивов различных размеров. Например, можно передать размер массива конструктору. Указание размеров объекта во время конструирования или путем обращения к некоторому методу действительно обеспечивает задание размера, однако такой способ не позволяет создать отдельный тип для каждого отдельного размера. Подход с использованием шаблона гарантирует, что размер становится частью типа. Так, Vector с пятью элементами типа double является типом, отличным от Vector с четырьмя элементами типа double.

Хотите ли Вы, чтобы различные размеры были различными типами, зависит от ваших нужд. Если сравнение двух векторов с четырьмя и пятью координатами не имеет особого смысла, то было бы неплохо сделать их различными типами. Вместе с тем, в случае классов для матриц, Вы,

возможно, не захотите иметь особый тип для каждого размера матриц, так как, например, умножение, работает с матрицами различных размеров. Если Вы столкнетесь с подобной проблемой, то Вам потребуются только разумные проверки времени выполнения, а не контроль типов при компиляции.

Хотя числовые параметры шаблонов часто используются для задания размеров различных элементов, как это было показано для класса Vector, этим их применение не ограничивается. Например, с помощью числовых параметров можно задавать диапазоны значений графических координат в графическом классе.

Наследование в шаблонах классов

Шаблоны классов, как и классы, поддерживают механизм наследования. Все основные идеи наследования при этом остаются неизменными, что позволяет построить иерархическую структуру шаблонов, аналогичную иерархии классов.

Рассмотрим очень простой пример, на котором продемонстрируем, каким образом можно создать шаблон класса, производный из нашего шаблона класса Pair. Пусть это будет класс Trio, в котором к паре элементов a и b из Pair, добавим элемент c.

```
template <class T>
class Trio: public Pair <T>
{
T c;
public:
Trio (T t1, T t2, T t3);
};
template <class T>
Trio<T>::Trio (T t1, T t2, T t3): Pair <T> (t1, t2), c(t3)
{}
// Заметим, что вызов родительского конструктора
```

// также сопровождается передачей типа T в качестве параметра.

Задания

Задание 1. Разработать шаблон класса для работы с однонаправленным некольцевым списком. Реализовать следующие действия: добавление звена в начало списка; удаление звена из начала списка; добавление звена в произвольное место списка, отличное от начала (например, после звена, указатель на которое задан); удаление звена из произвольного места списка, отличного от начала (например, после звена, указатель на которое задан); проверка, пуст ли список; очистка списка; формирование списка; печать списка. Решить задачу: составить программу, которая удаляет из списка L за каждым вхождением элемента E один элемент, если такой есть и он отличен от E.

Скриншоты кода программы и результат представлены на рисунках 1.1 – 1.4

Задание 2. Создать шаблон класса Vector размерности n. Определить несколько конструкторов, в том числе конструктор копирования. Реализовать методы для вычисления модуля вектора, скалярного произведения, сложения, вычитания, умножения на константу. Перегрузить операции сложения, вычитания, умножения, инкремента, декремента, индексирования, присваивания для данного класса. Создать массив объектов.

Скриншоты кода программы и результат представлены на рисунках 2.1 – 2.4.

```cpp
template <typename T>
class Node
{
public:
    Node *pNext;
    T data;
    Node(T data = T(), Node *pNext = nullptr)
    {
        this->data = data;
        this->pNext = pNext;
    }
};

template <typename T>
class List
{
public:
    List();
    ~List();

    void Delete_front();

    List<T> &operator+(T data);

    void Clear();

    int GetSize() { return Size; }

    T &operator[](const int index);

    void New_front(T data);

    void Insert(T data, int index);

    void RemoveAt(int index);

    void Delete_back();

    void Print();

    void DelE(List<T> &l, T item);

private:
    int Size;
    Node<T> *head;
};
```

Рисунок 1.1 – Скриншот кода программы

```
53  template <typename T>
54  List<T>::List()
55  {
56      Size = 0;
57      head = nullptr;
58  }
59
60  template <typename T>
61  List<T>::~List()
62  {
63      Clear();
64  }
65
66  template <typename T>
67  void List<T>::Delete_front()
68  {
69      Node<T> *temp = head;
70      head = head->pNext;
71      delete temp;
72      Size--;
73  }
74
75  template <typename T>
76  void List<T>::Delete_back()
77  {
78      RemoveAt(Size - 1);
79  }
80
81  template <typename T>
82  void List<T>::RemoveAt(int index)
83  {
84      if (index == 0)
85          Delete_front();
86      else
87      {
88          Node<T> *previous = this->head;
89          for (int i = 0; i < index - 1; i++)
90              previous = previous->pNext;
91          Node<T> *toDelete = previous->pNext;
92          previous->pNext = toDelete->pNext;
93          delete toDelete;
94          Size--;
95      }
96  }
```

Рисунок 1.2 – Скриншот кода программы

```
 98  template <typename T>
 99  List<T> &List<T>::operator+(T data)
100  {
101      if (head == nullptr)
102          head = new Node<T>(data);
103      else
104      {
105          Node<T> *current = this->head;
106          while (current->pNext != nullptr)
107              current = current->pNext;
108          current->pNext = new Node<T>(data);
109      }
110      Size++;
111      return *this;
112  }
113
114  template <typename T>
115  void List<T>::New_front(T data)
116  {
117      head = new Node<T>(data, head);
118      Size++;
119  }
120
121  template <typename T>
122  void List<T>::Insert(T data, int index)
123  {
124      if (index == 0)
125          New_front(data);
126      else
127      {
128          Node<T> *previous = this->head;
129
130          for (int i = 0; i < index - 1; i++)
131              previous = previous->pNext;
132          Node<T> *newNode = new Node<T>(data, previous->pNext);
133          previous->pNext = newNode;
134          Size++;
135      }
136  }
137
138  template <typename T>
139  void List<T>::Clear()
140  {
141      while (Size)
142          Delete_front();
143  }
```

Рисунок 1.3 – Скриншот кода программы

```cpp
template <typename T>
T &List<T>::operator[](const int index)
{
    int counter = 0;
    Node<T> *current = this->head;
    while (current != nullptr)
    {
        if (counter == index)
            return current->data;
        current = current->pNext;
        counter++;
    }
}

template <typename T>
void List<T>::Print()
{
    for (int i = 0; i < GetSize(); i++)
        cout << operator[](i) << " ";
    cout << endl;
}

template <typename T>
void List<T>::DelE(List<T> &l, T item)
{
    if (GetSize() == 0)
        return;

    for (int i = 0; i < GetSize(); i++)
    {
        if (l[i] == item && l[i + 1] && l[i + 1] != item)
        {
            RemoveAt(i + 1);
        }
    }
}

int main()
{
    setlocale(LC_ALL, "65001");
    system("chcp 65001");
    system("cls");
    List<int> L;
    L + 2 + 6 + 9 + 1 + 3 + 2 + 6;
    L.Print();
    L.DelE(L, 1);
    L.Print();

    return 0;
}
```

```
ПРОБЛЕМЫ  ВЫХОДНЫЕ ДАННЫЕ  ТЕРМИНАЛ  КОНСОЛЬ ОТЛАДКИ

2 6 9 1 3 2 6
2 6 9 1 2 6
PS C:\Users\seron\Desktop\study\3sem\OOP\laba9>
```

Рисунок 1.4 – Скриншот кода и результата работы программы

```
template <typename T>
class Vector
{
    int n;
    T *arr;

public:
    Vector()
    {
        n = 1;
        arr = new T[n];
    }
    Vector(int sizeN)
    {
        n = sizeN;
        arr = new T[n];
    }
    void SetVector()
    {
        for (int i = 0; i < n; i++)
        {
            cout << "Введите " << i << "-й элемент вектора: ";
            cin >> arr[i];
        }
    }
    T CountABS()
    {
        T Vabs = 0;
        for (int i = 0; i < n; i++)
        {
            Vabs += arr[i] * arr[i];
        }
        return sqrt(Vabs);
    }
    Vector operator=(Vector &v)
    {
        if (this == &v)
            return *this;
        n = v.n;
        for (int i = 0; i < n; i++)
            arr[i] = v.arr[i];
        return *this;
    }
    Vector operator+(Vector &v)
    {
        for (int i = 0; i < n; i++)
            arr[i] += v.arr[i];
        return *this;
    }
```

Рисунок 2.1 – Скриншот кода программы

```
56      Vector operator-(Vector &v)
57      {
58          for (int i = 0; i < n; i++)
59              arr[i] -= v.arr[i];
60          return *this;
61      }
62      Vector operator*(T item)
63      {
64          for (int i = 0; i < n; i++)
65              arr[i] = arr[i] * item;
66          return *this;
67      }
68      Vector operator*(Vector &v)
69      {
70          T scal;
71          for (int i = 0; i < n; i++)
72          {
73              scal += arr[i] * v.arr[i];
74          }
75          return scal;
76      }
77      void Show()
78      {
79          cout << "[";
80          for (int i = 0; i < n; i++)
81          {
82              cout << *(arr + i);
83              if (i < n - 1)
84              {
85                  cout << ", ";
86              }
87          }
88          cout << "]";
89      }
90  };
91
92  int main()
93  {
94      setlocale(LC_ALL, "65001");
95      system("chcp 65001");
96      system("cls");
97      Vector<int> v1(5);
98      v1.SetVector();
99      v1.Show();
100     cout << "\nМодуль вектора: " << v1.CountABS() << endl;
101
```

Рисунок 2.2 – Скриншот кода программы

```
102     Vector<int> v2(3);
103     v2.SetVector();
104     Vector<int> v3;
105     v3 = v2;
106
107     cout << "\nСумма вектора v2 и v3 =  ";
108     v2 + v3;
109     v2.Show();
110
111     cout << endl;
112     v3.Show();
113     cout << " умноженный на 5: ";
114     v3 * 5;
115     v3.Show();
116
117     cout << "\nИз ";
118     v3.Show();
119     cout << " вычесть ";
120     v2.Show();
121
122     v3 - v2;
123     cout << " = ";
124     v3.Show();
125
126     cout << "\nСкалярное произведение ";
127     v2.Show();
128     cout << "и ";
129     v3.Show();
130     v2 *v3;
131     cout << " = ";
132     v2.Show();
133     return 0;
134 }
```

Рисунок 2.3 – Скриншот кода программы

```
ПРОБЛЕМЫ   ВЫХОДНЫЕ ДАННЫЕ   ТЕРМИНАЛ   КОНСОЛЬ ОТЛАДКИ

Введите 0-й элемент вектора: 1
Введите 1-й элемент вектора: 2
Введите 2-й элемент вектора: 3
Введите 3-й элемент вектора: 4
Введите 4-й элемент вектора: 5
[1, 2, 3, 4, 5]
Модуль вектора: 7
Введите 0-й элемент вектора: 10
Введите 1-й элемент вектора: 20
Введите 2-й элемент вектора: 30

Сумма вектора v2 и v3 =  [20, 40, 60]
[10, 20, 30] умноженный на 5: [50, 100, 150]
Из [50, 100, 150] вычесть [20, 40, 60] = [30, 60, 90]
Скалярное произведение [20, 40, 60]и [30, 60, 90] = [20, 40, 60]
```

Рисунок 2.4 – Скриншот результата работы программы

Вывод: освоены и применены на практике шаблоны классов.