Лабораторная работа № 7 - "Шаблоны классов"

Цель работы

1. Создание консольного приложения, состоящего из нескольких файлов в системе программирования Visual Studio.
2. Реализация шаблона класса-контейнера.

Постановка задачи

1. Определить шаблон класса-контейнера (см. лабораторную работу №6).
2. Реализовать конструкторы, деструктор, операции ввода-вывода, операцию присваивания.
3. Перегрузить операции, указанные в варианте.
4. Инстанцировать шаблон для стандартных типов данных (int, float, double).
5. Написать тестирующую программу, иллюстрирующую выполнение операций для контейнера, содержащего элементы стандартных типов данных.
6. Реализовать пользовательский класс (см. лабораторную работу №3).
7. Перегрузить для пользовательского класса операции ввода-вывода.
8. Перегрузить операции необходимые для выполнения операций контейнерного класса.
9. Инстанцировать шаблон для пользовательского класса.
10. Написать тестирующую программу, иллюстрирующую выполнение операций для контейнера, содержащего элементы пользовательского класса.

> Класс- контейнер МНОЖЕСТВО с элементами типа int. Реализовать операции: [] – доступа по индексу; != - проверка на неравенство; < число – принадлежность числа множеству; Пользовательский класс Money для работы с денежными суммами. Число должно быть представлено двумя полями: типа long для рублей и типа int для копеек. Дробная часть числа при выводе на экран должна быть отделена от целой части запятой.

Описание параметризированного класса-контейнера.

```
// Множество
template<class T>
class Array {
public:
Array(int s, T k);

Array(const Array<T> &a);

~Array();

Array &operator=(const Array<T> &a);     //оператор присваивания
T &operator[](int index); //операция доступа по индексу
Array operator+(T k);

int operator()(); //операция, возвращающая длину множества
friend ostream &operator
```

```
<<<>(
ostream &out,
const Array<T> &a
);

friend istream &operator>><>(istream &in, Array<T> &a);

private:
int size;//размер множества
T *data;//укзатель на динамический массив значений множества
};
```

Определение компонентных функций

```
template<class T>
Array<T>::Array(int s, T k) {
    size = s;
    data = new T[size];
    for (int i = 0; i < size; i++)
        data[i] = k;
}

template<class T>
Array<T>::Array(const Array<T> &a) {
    size = a.size;
    data = new T[size];
    for (int i = 0; i < size; i++)
        data[i] = a.data[i];
}

template<class T>
Array<T>::~Array() {
    delete[]data;
    data = 0;
}

template<class T>
Array<T> &Array<T>::operator=(const Array<T> &a) {
    if (this == &a)return *this;
    size = a.size;
    if (data != 0) delete[]data;
    data = new T[size];
    for (int i = 0; i < size; i++)
        data[i] = a.data[i];
    return *this;
}

template<class T>
T &Array<T>::operator[](int index) {
    if (index < size) return data[index];
    else cout << "\nError! Index>size";
```

```
}

template<class T>
int Array<T>::operator()() {
    return size;
}

template<class T>
ostream &operator<<(ostream &out, const Array<T> &a) {
    for (int i = 0; i < a.size; ++i)
        out << a.data[i] << " ";
    return out;
}

template<class T>
istream &operator>>(istream &in, Array<T> &a) {
    for (int i = 0; i < a.size; ++i)
        in >> a.data[i];
    return in;
}

template<class T>
Array<T> Array<T>::operator+(const T k) {
    Array<T> temp(size, k);
    for (int i = 0; i < size; ++i)
        temp.data[i] = data[i] + k;
    return temp;
}
```

Описание пользовательского класса и его компонентных функций

```
Money::Money() { rub = 0; cheers = 0; }
Money::Money(long r, int c) {
    rub = (r * 100 + c) / 100;
    cheers = c;
    cheers = (r * 100 + c) % 100;
}
Money::Money(const Money &m) { rub = m.rub; cheers = m.cheers; }
Money::~Money() {}

long Money::getRub() { return rub; }
int Money::getCheers() { return cheers; }
void Money::setRub(long r) { rub = r; }
void Money::setCheers(long c) { cheers = c; }

// перегрузка операции присваивания
Money &Money::operator=(const Money &m) {
    if (&m == this) return *this;
    rub = m.rub;
    cheers = m.cheers;
    return *this;
```

```
}

// перегрузка операций сравнения
bool Money::operator==(const Money &other) {
    return (this->rub == other.rub && this->cheers == other.cheers);
}

bool Money::operator!=(const Money &other) { return !(*this == other); }

// перегрузка операции вычитания
Money operator-(const Money &lhs, const Money &rhs) {
    Money obj(lhs.rub - rhs.rub, lhs.cheers - rhs.cheers);
    return obj;
}

Money Money::operator+(const Money& k) const {
    int t = rub * 100 + cheers;
    int kt = k.rub * 100 + k.cheers;
    t += kt;
    Money temp(t / 100, t % 100);
    return temp;
}

// перегрузка глобальной функции-операции ввода
istream &operator>>(istream &in, Money &m) {
    cout << "rub?";
    in >> m.rub;
    cout << "cheers?";
    in >> m.cheers;
    return in;
}

// перегрузка глобальной функции-операции вывода
ostream &operator<<(ostream &out, const Money &m) {
    return (out << m.rub << "," << setfill('0') << setw(2) << m.cheers) << "rub";
}
```

Функция main()

```
int main() {
    //инициализация, ввод и вывод значений вектора
    Array<int> A(5, 0);
    cin >> A;
    cout << A << endl;
    //инициализация и вывод значений вектора
    Array<int> B(10, 1);
    cout << B << endl;
    //операция присваивания
    B = A;
    cout << B << endl;
    //доступ по индексу
```

```
    cout << A[2] << endl;
    //получение длины вектора
    cout << "size=" << A() << endl;
    //операция сложения с константой
    B = A + 10;
    cout << B << endl;

    Money m;
    cin >> m;
    cout << m;

    int k;
    cout << "k?";
    cin >> k;
    Money p;
    //p = m + k; TODO Fix this
    cout << p;

    Money t;
    cin >> t;
    cout << t;
    Array<Money> A2(5, t);
    cin >> A2;
    cout << A2 << endl;
    Array<Money> B2(10, t);
    cout << B2 << endl;
    B2 = A2;
    cout << B2 << endl;
    cout << A2[2] << endl;
    cout << "size=" << A2() << endl;
    B2 = A2 + t;
    cout << B << endl;

    return 0;
}
```

Объяснение результатов работы программы

Программа демонстрирует работу шаблонного класса Array с двумя типами данных: int и пользовательским классом Money.

```
Для типа int:

    Создается массив из 5 элементов (инициализированных 0)

    Пользователь вводит значения элементов

    Выполняются операции: присваивание, доступ по индексу, получение размера,
сложение с константой

Для класса Money:
```

```
Создаются объекты Money с вводом значений рублей и копеек

Демонстрируется работа операторов ввода/вывода

Создаются массивы Array<Money> с операциями аналогичными int
```

Особенности работы с Money:

```
Копейки автоматически нормализуются (например, 125 копеек → 1 рубль 25 копеек)

Вывод форматируется с ведущим нулем для копеек (например, "5,07rub")

Реализованы арифметические операции с учетом перевода рублей в копейки
```

Ответы на контрольные вопросы

1. В чем смысл использования шаблонов?

Ответ: Шаблоны позволяют создавать обобщенные классы и функции, которые могут работать с разными типами данных без необходимости переписывать код для каждого типа. Это повышает повторное использование кода и уменьшает дублирование.

2. Каковы синтаксис/семантика шаблонов функций?

Ответ: Синтаксис:

```
template <typename T>
T functionName(T param) { ... }
```

Семантика: компилятор генерирует конкретные реализации функции для каждого используемого типа.

3. Каковы синтаксис/семантика шаблонов классов?

Ответ: Синтаксис:

```
template <class T>
class ClassName { ... };
```

Семантика: класс становится "шаблоном" для создания семейства классов с одинаковой структурой, но разными типами данных.

4. Что такое параметры шаблона функции?

Ответ: Параметры шаблона функции - это типы или значения, которые передаются в шаблон при его использовании. Они указываются в угловых скобках после ключевого слова template.

5. Перечислите основные свойства параметров шаблона функции.

Ответ:

1. Могут быть типами (class/typename) или конкретными значениями
2. Может быть несколько параметров
3. Параметры могут иметь значения по умолчанию
4. Используются для обобщения алгоритма функции

6. Как записывать параметр шаблона?

Ответ: Параметры шаблона записываются в угловых скобках после ключевого слова template:

```
template <class T, int size = 10>
```

7. Можно ли перегружать параметризованные функции?

Ответ: Да, параметризованные функции можно перегружать как обычные функции, а также можно создавать специализации шаблонов для конкретных типов.

8. Перечислите основные свойства параметризованных классов.

Ответ:

1. Позволяют создавать обобщенные структуры данных
2. Все методы класса автоматически становятся параметризованными
3. Могут иметь несколько параметров
4. Поддерживают специализацию
5. Могут иметь статические члены (отдельные для каждого инстанцирования)

9. Все ли компонентные функции параметризованного класса являются параметризованными?

Ответ: Да, все методы шаблонного класса неявно получают параметры шаблона своего класса и могут использовать их в своей реализации.

10. Являются ли дружественные функции, описанные в параметризованном классе, параметризованными?

Ответ: Да, если они объявлены внутри шаблонного класса без явного указания template, они становятся шаблонными функциями с теми же параметрами, что и класс.

11. Могут ли шаблоны классов содержать виртуальные компонентные функции?

Ответ: Да, шаблоны классов могут содержать виртуальные функции, но сама виртуальная функция не может быть шаблонной (не может иметь свои собственные параметры шаблона).

12. Как определяются компонентные функции параметризованных классов вне определения шаблона класса?

Ответ: Каждая функция должна быть объявлена как шаблон, с теми же параметрами, что и класс, и использовать полное имя класса с параметрами:

```
template <class T>
void ClassName<T>::function() { ... }
```

13. Что такое инстанцирование шаблона?

Ответ: Инстанцирование шаблона - это процесс создания конкретного класса или функции из шаблона путем подстановки конкретных типов или значений вместо параметров шаблона.

14. На каком этапе происходит генерирование определения класса по шаблону?

Ответ: Генерация происходит на этапе компиляции, когда компилятор встречает использование шаблона с конкретными параметрами. Это называется "неявным инстанцированием".

Лабораторная работа доступна в GitHub репозитории hanriel/PSTU-CPP