

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)

ПЕРЕГРУЗКА ОПЕРАЦИЙ

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

Студент группы 549



Баулин С.К.

«__» _____ 2020 г.

Кандидат физико-
математических наук,
доцент кафедры ЭМИС

_____ Шельмина Е. А.

«__» _____ 2020 г.

Томск 2020

Лабораторная работа №6

Перегрузка операций

Цель работы: освоить и применить на практике функцию перегрузки операций.

Теоритические сведения

C++ поддерживает перегрузку операторов (operator overloading). За небольшими исключениями большинство операторов C++ могут быть перегружены, в результате чего они получают специальное значение по отношению к определенным классам. Например, класс, определяющий связанный список, может использовать оператор + для того, чтобы добавлять объект к списку. Другой класс может использовать оператор + совершенно иным способом. Когда оператор перегружен, ни одно из его исходных значений не теряет смысла. Просто для определенного класса объектов определен новый оператор. Поэтому перегрузка оператора + для того, чтобы обрабатывать связанный список, не изменяет его действия по отношению к целым числам.

Операторные функции обычно будут или членами, или друзьями того класса, для которого они используются. Несмотря на большое сходство, имеется определенное различие между способами, которыми перегружаются операторные функции-члены и операторные функции-друзья. В этом разделе мы рассмотрим перегрузку только функций-членов. Позже в этой главе будет показано, каким образом перегружаются операторные функции-друзья.

Для того, чтобы перегрузить оператор, необходимо определить, что именно означает оператор по отношению к тому классу, к которому он применяется. Для этого определяется функция-оператор, задающая действие оператора.

Общая форма записи функции-оператора для случая, когда она является членом класса, имеет вид:

```
тип имя_класса::operator#(список_аргументов)
{
    // действия, определенные применительно к классу
}
```

Здесь перегруженный оператор подставляется вместо символа #, а тип задает тип значений, возвращаемых оператором. Для того, чтобы упростить использование перегруженного оператора в сложных выражениях, в качестве возвращаемого значения часто выбирают тот же самый тип, что и класс, для которого перегружается оператор. Характер списка аргументов определяется не-сколькими факторами.

В ранних версиях C++ было невозможно определить, предшествует или следует за операндом перегруженный оператор ++ или --. Например, для объекта О следующие две инструкции были идентичными:

```
O++;
++O;
```

Однако более поздние версии C++ позволяют различать префиксную и постфиксную форму операторов инкремента и декремента. Для этого программа должна определить две версии функции operator++(). Одна из них должна быть такой же, как показано в предыдущей программе. Другая объявляется следующим образом: loc operator++(int x);

Если ++ предшествует операнду, то вызывается функция operator++(). Если же ++ следует за операндом, то тогда вызывается функция operator++(int x), где x принимает значение 0.

Действие перегруженного оператора по отношению к тому классу, для которого он определен, не обязательно должно соответствовать каким-либо

образом действию этого оператора для встроенных типов C++. Например, операторы `<<` и `>>` применительно к `cout` и `cin` имеют мало общего с их действием на переменные целого типа. Однако, исходя из стремления сделать код более легко читаемым и хорошо структурированным, желательно, чтобы перегруженные операторы соответствовали, там где это возможно, смыслу исходных операторов. Например, оператор `+` по отношению к классу `three_d` концептуально сходен с оператором `+` для переменных целого типа. Мало пользы, например, можно ожидать от такого оператора `+`, действие которого на соответствующий класс будет напоминать действие оператора `||`. Хотя можно придать перегруженному оператору любой смысл по своему выбору, но для ясности его применения желательно, чтобы его новое значение соотносилось с исходным значением.

Имеются некоторые ограничения на перегрузку операторов. Во-первых, нельзя изменить приоритет оператора. Во-вторых, нельзя изменить число операндов оператора. Наконец, за исключением оператора присваивания, перегруженные операторы наследуются любым производным классом. Каждый класс обязан определить явным образом свой собственный перегруженный оператор `=`, если он требуется для каких-либо целей. Разумеется, производные классы могут перегрузить любой оператор, включая и тот, который был перегружен базовым классом. Следующие операторы не могут быть перегружены: `.` `::` `*` `?`

Задание 1.

Для строки символов реализовать операции:

- a. проверка в строке наличия заданного символа (операция `!`);
- b. перевод указанного символа строки в код ASCII (операция `%`).

Членом класса сделать функцию с именем `strlen()` для нахождения суммы кодов ASCII всех символов данной строки.

Скриншоты выполненного задания представлены на рисунках 1 - 2.

```
class Per
{
public:
    string a;
    char b;
    Per operator!=(Per t);
    Per operator%(Per t);

    void strlen();
};
Per Per::operator!=(Per t)
{
    Per t1, t2;
    bool check = false;
    t1.a = a;
    t2.b = t.b;
    for (int i = 0; i < t1.a.length() + 1; i++)
    {
        if (t1.a[i] == t2.b)
        {
            check = true;
        }
    }
    if (check)
    {
        cout << "Данный символ есть в строке." << endl;
    }
    else
    {
        cout << "Данного символа нет в строке." << endl;
    }
};
Per Per::operator%(Per t)
{
    cout << "Код символа " << t.b << " в ASCII: " << (int)t.b << endl;
};
```

Рисунок 1 – Скриншот начала кода задания 1

```

44 void Per::strlen()
45 {
46     int count = 0, i = 0;
47     for (i = 0; i < a.length() + 1; i++)
48     {
49         count += (int)a[i];
50     }
51     cout << "Сумма кодов ASCII всех символов первой строки = " << count << endl;
52 };
53 int main()
54 {
55     SetConsoleCP(1251);
56     SetConsoleOutputCP(1251);
57     setlocale(LC_ALL, "Russian");
58     Per str1, str2;
59     cout << "Введите первую строку: ";
60     getline(cin, str1.a);
61     cout << "Введите символ для поиска в первой строке и для перевода в код ASCII: ";
62     cin >> str2.b;
63     cout << endl;
64     str1 != str2;
65     str1 % str2;
66     str1.strlen();
67     return 0;
68 }

```

ПРОБЛЕМЫ ВЫХОДНЫЕ ДАННЫЕ ТЕРМИНАЛ КОНСОЛЬ ОТЛАДКИ

```

PS C:\Users\seron\Desktop\study\3sem\OOP\laba6> .\laba6_1.exe
Введите первую строку: Hello, world!
Введите символ для поиска в первой строке и для перевода в код ASCII: !

Данный символ есть в строке.
Код символа ! в ASCII: 33
Сумма кодов ASCII всех символов первой строки = 1161

```

Рисунок 1.2 – Скриншот конца кода и результат задания 1

Задание 2.

Определить класс-строку. В класс включить два конструктора: для определения класса строки символов и путем копирования другой строки (объекта класса строки).

Определить операции над строками:

- ++ преобразование символов строки в прописные (заглавные) символы;
- -- нахождение самого короткого слова в строке.

Скриншоты выполненного задания представлены на рисунке 3.

```

8  class Str // you, 2 months ago * added
9  {
10 public:
11     string a;
12     Str operator++(); // в заглавные
13     Str operator--(); // самое короткое слово в строке
14     void show()
15     {
16         cout << a << endl;
17     };
18 };
19 Str Str::operator++()
20 {
21     Str toUp;
22     toUp.a = a;
23     for (int i = 0; i < a.length() + 1; i++)
24     {
25         toUp.a[i] = toupper(a[i]);
26     }
27     {
28         return toUp;
29     }
30 }
31 Str Str::operator--()
32 {
33     Str k;
34     string word, min_word;
35     istringstream r(a);
36     r >> min_word;
37     while (r >> word)
38     {
39         if (min_word.size() > word.size())
40             min_word = word;
41     }
42     k.a = min_word;
43     {
44         return k;
45     }
46 }
47 int main()
48 {
49     setlocale(LC_ALL, "65001");
50     system("chcp 65001");
51     system("cls");
52
53     Str A, B, C;
54     A.a = "Hello, my world!";
55     B = A;
56     B.show();
57     C = ++B;
58     cout << "Все строки в заглавные: ";
59     C.show();
60     C = --B;
61     cout << "Самое короткое слово в строке: ";
62     C.show();
63     return 0;
64 }

```

ПРОБЛЕМЫ ВЫХОДНЫЕ ДАННЫЕ ТЕРМИНАЛ КОНСОЛЬ ОТЛАДКИ

```

Hello, my world!
Все строки в заглавные: HELLO, MY WORLD!
Самое короткое слово в строке: my

```

Рисунок 3 – Скриншот кода и результата задания 2

Задание 3.

При решении задач необходимо описать класс, который используется для представления элементов динамической структуры данных. Затем разрабатывается класс для работы с используемой динамической структурой данных, которая при тестировании класса может быть построена путем ввода данных: а) с клавиатуры; б) из файла.

Возможны два варианта решения:

- а) динамическая структура данных постоянно хранится в памяти;
- б) динамическая структура данных хранится в файле.

2. Построить класс для работы со стеком. Элемент стека – целое число. Ввести две неубывающие последовательности чисел в два стека. Использовать третий стек для слияния двух последовательностей в одну неубывающую.

Скриншоты выполненного задания представлены на рисунках 4 - 6.

Вывод: освоены и применены на практике функции перегрузки операций.


```

9
10 private:
11     int data[MAX];
12
13 public:
14     int n;
15     Stc()
16     {
17         n = 0;
18     }
19     ~Stc() {}
20     int get_top()
21     {
22         if (n == 0)
23         {
24             cerr << "Стек пуст!" << endl;
25             return 0;
26         }
27         return data[n - 1];
28     }
29     void push(int item)
30     {
31         if (n < MAX)
32         {
33             data[n++] = item;
34         }
35         else
36         {
37             cerr << "Стек переполнен!" << endl;
38         }
39     }
40     void del()
41     {
42         if (n == 0)
43         {
44             cerr << "Стек пуст!" << endl;
45         }
46         else
47         {
48             n--;
49         }
50     }
51     int size()
52     {
53         return n;
54     }
55     void show()
56     {
57         for (int i = 0; i < n; i++)
58         {
59             cout << data[i] << " ";
60         }
61         cout << endl;
62     }
63 };
64

```

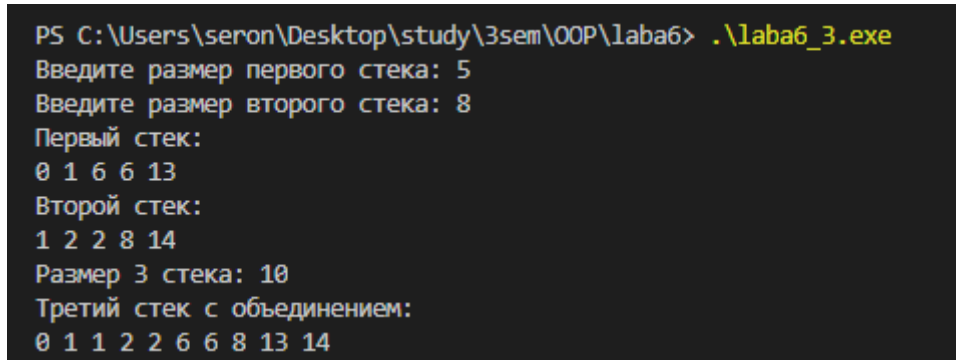
Рисунок 4 – Скриншот класса задания 3

```

Stack s1, s2, s3;
int k1, k2, n1 = 0, n2 = 0;
cout << "Введите размер первого стека: ";
cin >> k1;
cout << "Введите размер второго стека: ";
cin >> k2;
for (int i = 0; i < k1; i++)
{
    n1 = rand() % (n1 + 5) + n1;
    s1.push(n1);
}
for (int i = 0; i < k2; i++)
{
    n2 = rand() % (n2 + 5) + n2;
    s2.push(n2);
}
cout << "Первый стек:" << endl;
s1.show();
cout << "Второй стек:" << endl;
s2.show();
while ((s1.size() > 0) || (s2.size() > 0))
{
    if ((s1.size() > 0) && (s2.size() > 0))
    {
        if (s1.get_top() > s2.get_top())
        {
            s3.push(s1.get_top());
            s1.del();
        }
        else
        {
            s3.push(s2.get_top());
            s2.del();
        }
    }
    else if (s1.size() > 0)
    {
        s3.push(s1.get_top());
        s1.del();
    }
    else
    {
        s3.push(s2.get_top());
        s2.del();
    }
}
cout << "Размер 3 стека: " << s3.size() << endl;
cout << "Третий стек с объединением: " << endl;
while (s3.size())
{
    cout << s3.get_top() << " ";
    s3.del();
}
return 0;

```

Рисунок 5 – Скриншот функции main задания 3



```
PS C:\Users\seron\Desktop\study\3sem\OOP\laba6> .\laba6_3.exe
Введите размер первого стека: 5
Введите размер второго стека: 8
Первый стек:
0 1 6 6 13
Второй стек:
1 2 2 8 14
Размер 3 стека: 10
Третий стек с объединением:
0 1 1 2 2 6 6 8 13 14
```

Рисунок 6 – Скриншот результата задания 3