В.Л. Тарасов

Лекции по программированию на C++

Лекция 13

Конструктор копирования и оператор присваивания

13.1. Проблемы при копировании

При присваивании и копировании объектов выполняется *почленное* копирование, о чем говорилось в §11.6. Пример копирования приведен в программе 11.7 на примере класса `TimeDay`. Для простых классов, не содержащих указатели, проблем не возникает, достаточно встроенного *почленного* копирования одного объекта в другой объект. Посмотрим, что произойдет при выполнении копирования и инициализации объектов класса, имеющего члены-указатели на выделяемую в конструкторе класса память, на примере многоугольников из программы 11.11.

Программа 13.1. Многоугольники. Неправильное копирование

В данной программе многоугольники класса `Plgn` не рисуются на экране, но лишь копируются.

```
// Файл PlgnBadCopy.cpp
class Plgn                          // Класс многоугольников
{
    int n;                          // Число вершин
    float *x;                       // Указатель на массив абсцисс
    float *y;                       // Указатель на массив ординат
public:
    Plgn(int nn = 3);               // Конструктор с аргументом по умолчанию
    ~Plgn();                        // Деструктор
};
# include <cstdlib>                 // Доступ к rand()
# include <iostream>
# include <ctime>
using namespace std;
Plgn::Plgn(int nn)                  // Конструктор
{
    n = nn;
```

```
    x = new float [n];           // Выделение памяти под массив абсцисс
    y = new float [n];           // Выделение памяти под массив ординат
    for(int i = 0; i < n; i++){                        // Заполнение
        x[i] = 2 * float(rand()) / RAND_MAX - 1;       // массивов координат
        y[i] = 2 * float(rand()) / RAND_MAX - 1; ;     // случайными числами
    }
    cout << "Конструктор: " << "x = " << x
                            << ", y = " << y << endl;
}

Plgn::~Plgn()                    // Деструктор
{
    cout << "Деструктор: " << "x = " << x
                           << ", y = " << y << endl;
    delete [] x;                 // Освобождение
    delete [] y;                 // памяти
}

// democopy: создание и копирование многоугольников
void democopy()
{
    cout << "Создание пятиугольника P\n";
    Plgn P(5);                   // (1) Создание пятиугольника
    cout << "Создание треугольника T\n";
    Plgn T;                      // (2) Создание треугольника
    cout << "Создание пятиугольника Q как копии P\n";
    Plgn Q  =  P;                // (3) Создание копии пятиугольника P
    cout << "Присваивание T = P\n";
    T = P;                       // (4) Превращение треугольника
                                 // в пятиугольник
    cout << "Удаление Q, T, P\n";
}                                // (5) Вызов деструкторов для Q, T, P

void main()
{
    setlocale(LC_ALL, "Russian");
    srand(time(0));              // Инициализация генератора случайных чисел
    cout << "democopy() начинает\n";
    democopy();
    cout << "democopy() завершилась\n";
}
```

Рассмотрим действия с памятью, выполняемые `democopy()` (рис.13.1).

На шаге (1) создается пятиугольник `P` со своей переменной `n` и своими указателями `x`, `y`, которые указывают на массивы памяти, выделенные в конструкторе.

На шаге (2) создается треугольник `T`, имеющий свои `n`, `x`, `y` и свои массивы координат.

На шаге (3) создается многоугольник `Q`, у которого `n`, `x`, `y` имеют те же значения, что и у многоугольника `P`. Указатели `Q.x`, `Q.y` указывают на ту же память, что и `P.x`, `P.y`.

На шаге (4) за счет присваивания `T.n`, `T.x`, `T.y` получают такие же значения, как `P.n`, `P.x`, `P.y`.

В результате возникли две проблемы.

Первая проблема состоит в том, что потеряна связь с памятью, выделенной треугольнику `T`. Память, выделяемая динамически оператором `new`, должна освобождаться явно оператором `delete`, в противном случае она все равно будет числится за программой, потерявшей с ней связь. Произойдет «утечка памяти».

Вторая проблема состоит в том, что теперь на одну и ту же область памяти ссылаются три указателя. С этой памятью можно работать через объекты `P`, `Q` и `T`, и «правая рука не будет знать, что делает левая».

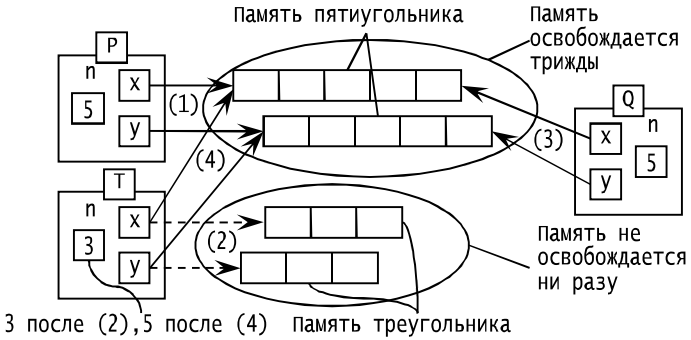

Рис. 13.1. Работа с памятью функции `democopy()`

При завершении функции `democopy()` будут вызваны деструкторы объектов `Q`, `T`, `P`, которые три раза освободят одну и ту же память.

Результаты выполнения программы показаны на рис.13.2. Из отладочных сообщений видно, что программа была остановлена при выполнении деструктора для `T`, когда повторно освобождалась одна и та же память.

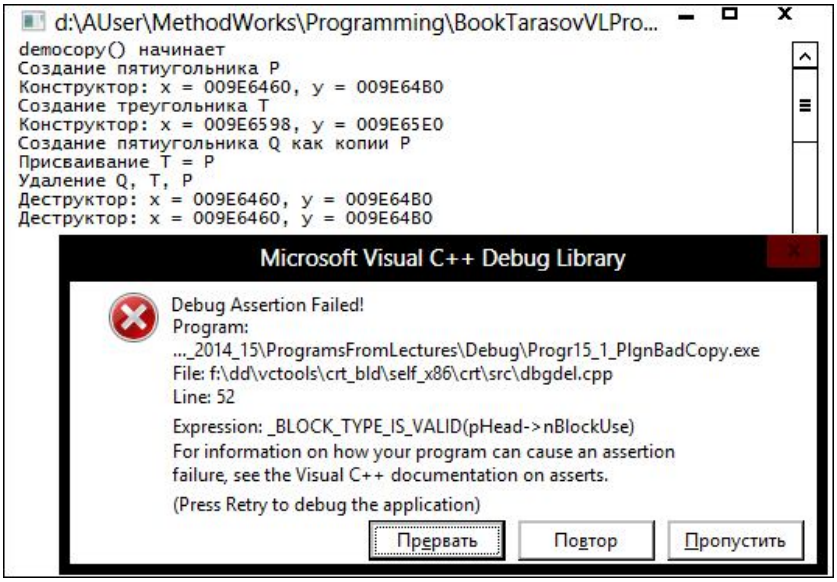

Рис. 13.2. Ошибка при повторном освобождении памяти

13.2. Решение проблем копирования

Чтобы избежать проблем, связанных с присваиванием объектов класса, содержащего указатели, следует перегрузить *оператор присваивания*, в котором предусмотреть выделение памяти под объект-копию.

Чтобы не возникало проблем при инициализации объектов, следует предусмотреть в классе *конструктор копирования*, который должен выделять память при создании объекта-копии.

Пример перегрузки оператора присваивания и создания конструктора копирования приведен в следующей программе.

Программа 13.2. Правильное копирование многоугольников

```
// Файл PlgnRightCopy.cpp
class Plgn                      // Класс многоугольников
{
    int n;                      // Число вершин
    float *x;                   // Указатель на массив абсцисс
    float *y;                   // Указатель на массив ординат
public:
    Plgn(int nn = 3);           // Конструктор с аргументом по умолчанию
```

```
    ~Plgn();                        // Деструктор

    Plgn(const Plgn& mn);           // Конструктор копирования.
                // Создает новый объект как копию существующего mn

    Plgn& operator= (const Plgn& mn); // Оператор присваивания.
                // Присваивает многоугольнику значение другого mn

    void PrnCoord();                    // Вывод координат

    friend Plgn scale_2(Plgn mn);       // Увеличение координат в 2 раза
};

# include <cstdlib>                    // Доступ к rand()
# include <iostream>
# include <ctime>
using namespace std;
# include <windows.h>

Plgn::Plgn(int nn)                  // Конструктор
{
    n = nn;
    x = new float [n];              // Выделение памяти под массив абсцисс
    y = new float [n];              // Выделение памяти под массив ординат
    for(int i = 0; i < n; i++){                         // Заполнение
        x[i] = 2 * float(rand()) / RAND_MAX - 1;        // массивов координат
        y[i] = 2 * float(rand()) / RAND_MAX - 1; ;      // случайными числами
    }
    cout << "Конструктор: " << "x = " << x
                           << ", y = " << y << endl;
}

Plgn::~Plgn()                       // Деструктор
{
    cout << "Деструктор: " << "x = " << x
                           << ", y = " << y << endl;
    delete [] x;                    // Освобождение
    delete [] y;                    // памяти
}

Plgn::Plgn(const Plgn& mn)          // Конструктор копирования
{
    n = mn.n;                       // Количество вершин
    x = new float [n];              // Выделение памяти под массив абсцисс
    y = new float [n];              // Выделение памяти под массив ординат
    for(int i = 0; i < n; i++){     // Копирование координат
        x[i] = mn.x[i];
        y[i] = mn.y[i];
    }
    cout << "Конструктор копирования: " << "x = " << x
                                        << ", y = " << y << endl;
}

Plgn& Plgn :: operator= ( const Plgn& mn )    // Оператор присваивания
{
    if ( this != &mn ) {     // Если присваивание не самому себе
```

```
        delete [] x;         // Удаление старой памяти
        delete [] y;
        n = mn.n;            // Новый размер многоугольника
        x = new float [n];   // Выделение новой памяти под массив абсцисс
        y = new float [n];   // Выделение новой памяти под массив ординат

        for(int i = 0; i < n; i++){  // Копирование координат
            x[i] = mn.x[i];
            y[i] = mn.y[i];
        }
    }
    cout << "Оператор присваивания: " << "x = " << x
                                      << ", y = " << y << endl;
    return *this;        // Возвращаем ссылку на левую часть
}

void Plgn::PrnCoord()    // Вывод координат
{
    for(int i = 0; i < n; i++)
        cout << i << " ( " << x[i] << ", " << y[i] << ")\n";
}

Plgn scale_2 ( Plgn mn ) // Увеличивает координаты mn в 2 раза
{   // Многоугольник mn передается в функцию по значению,
    // то есть внутри функции создается его копия с использованием
    // конструктора копирования

    for(int i = 0; i < mn.n; i++){
        mn.x[i] *= 2; mn.y[i] *= 2;
    }
    return mn;

    // При возвращения объекта из функции создается его копия
    // с использованием конструктора копирования, которая
    // передается в вызывающую функцию. Сам локальный объект,
    // существовавший внутри функции, уничтожается с
    // использованием деструктора
}

// democopy: создание и копирование многоугольников
void democopy()
{
    cout << "Создание пятиугольника P\n";
    Plgn P(5);                   // (1) Создание пятиугольника
    cout << "Создание треугольника T\n";
    Plgn T;                      // (2) Создание треугольника
    cout << "Создание пятиугольника Q как копии P\n";
    Plgn Q  =  P;                // (3) Создание копии пятиугольника P
    cout << "Присваивание T = P\n";
    T = P;                       // (4) Превращение треугольника
                                 // в пятиугольник
    cout << "Удаление Q, T, P\n";
}                                // (5) Вызов деструкторов для Q, T, P

void main()
```

```
{
    SetConsoleOutputCP(1251);
    srand(time(0));                 // Инициализация генератора случайных чисел
    cout << "democopy() начинает\n";
    democopy();
    cout << "democopy() завершилась\n";
    Plgn tr;                        // Треугольник
    cout << "Исходный треугольник:\n";
    tr.PrnCoord();
    cout << "Работает scale_2()\n";
    tr = scale_2(tr);
    cout << "Масштабированный треугольник:\n";
    tr.PrnCoord();
    cin.get();
}
```

Схема работы с памятью функции democopy() при наличии конструктора копирования и оператора присваивания приведена на рис.13.3.

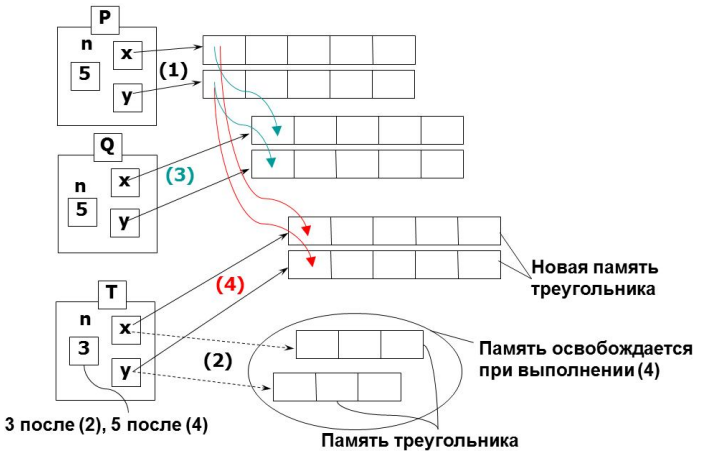

Рис. 13.3. Работа конструктора копирования и оператора присваивания

Программа выводит:

```
democopy() начинает
Создание пятиугольника P
Конструктор: x = 01181F58, y = 01182D60
Создание треугольника T
Конструктор: x = 01182C90, y = 01182CC8
Создание пятиугольника Q как копии P
Конструктор копирования: x = 01189680, y = 0118A0B8
Присваивание T = P
Оператор присваивания: x = 01189C38, y = 01189C78
Удаление Q, T, P
```

```
Деструктор: x = 01189680, y = 0118A0B8
Деструктор: x = 01189C38, y = 01189C78
Деструктор: x = 01181F58, y = 01182D60
democopy() завершилась
Конструктор: x = 01181F58, y = 01182D60
Исходный треугольник:
0 ( 0.756157, 0.100436)
1 ( -0.61095, 0.97174)
2 ( 0.639027, -0.890866)
Работает scale_2()
Конструктор копирования: x = 01182C90, y = 01182CC8
Конструктор копирования: x = 01189680, y = 011896B8
Деструктор: x = 01182C90, y = 01182CC8
Оператор присваивания: x = 01181F58, y = 01182D60
Деструктор: x = 01189680, y = 011896B8
Масштабированный треугольник:
0 ( 1.51231, 0.200873)
1 ( -1.2219, 1.94348)
2 ( 1.27805, -1.78173)

Деструктор: x = 01181F58, y = 01182D60
```

Проследим работу программы по отладочной печати.

Как и в предыдущей программе при выполнении функции `democopy()` для пятиугольника `P` и треугольника `T` выделяется отдельная память в конструкторе. При создании объекта `Q` вызывается конструктор копирования, который выделяет для `Q` отдельную память.

При выполнении присваивания `T = P;` вызывается оператор присваивания, который удаляет старую память треугольника `T`, выделяет новую память и копирует в нее координаты пятиугольника `P`, благодаря чему треугольник превращается в пятиугольник.

При завершении `democopy()` три раза вызывается деструктор, который удаляет ранее выделенную память.

Затем в `main()` создается треугольник `tr`, причем для для него выделилась память, которую ранее занимал пятиугольник `P` из функции `democopy()`. При вызове `scale_2(tr)` вызывается конструктор копирования для создания копии аргумента `tr` для использовании внутри функции. Второй раз конструктор копирования вызывается при передаче объекта из `scale_2()` в `main()`. Затем деструктор уничтожает копию `tr`, существовавшую в `scale_2()`. После присваивания в `main()` треугольнику `tr` значения безымянного многоугольника, возвращенного из `scale_2()`, вызывается деструктор для уничтожения этого безымянного многоугольника.

Наконец, при завершении `main()` вызывается деструктор для уничтожения `tr`.