УВЕЛИЧЕНИЕ ОБЪЁМА ИСПОЛЬЗУЕМОЙ ОПЕРАТИВНОЙ ПАМЯТИ КОМПЬЮТЕРА ПРИ НАСЛЕДОВАНИИ КЛАССОВ В ОБЪЕКТНО-ОРИЕНТИРОВАННОМ ПРОГРАММИРОВАНИИ

INCREASE OF THE AMOUNT OF RAM USED BY CLASS INHERITANCE IN OBJECT-ORIENTED PROGRAMMING

P. Novikov

Summary. The paper deals with the way to investigate by library function 'system' the amount of RAM used in Borland C++. The research shows that the accuracy of functioning 'system' is 1 Kb. Experimentally shown that the addition of classes in inheritance hierarchy increases the amount of RAM used. There is also the established coincidence of the research results given in the paper with the results of the other researches in Microsoft Visual Studio 98 based on analysis of physical addresses of the objects' components. The impossibility of reducing the researched amount of RAM used by overriding methods, by virtual methods and by using private access to the components of classes is under discussion in the paper.

Keywords: Class inheritance, amount of RAM used, inheritance hierarchy, override, virtual, private access.

Новиков Павел Владимирович
К.т.н., доцент, Московский авиационный институт (национальный исследовательский университет)
novikov.mai@mail.ru

Аннотация. В статье рассмотрен способ исследования затрат оперативной памяти в среде Borland C++ с помощью библиотечной функции system. Показано, что точность работы функции system равна 1 Кбайт. Экспериментально доказано, что за счёт увеличения количества классов в иерархии наследования растут затраты оперативной памяти. Установлено совпадение результатов исследований в настоящей статье с результатами альтернативных исследований в среде Microsoft Visual Studio 98 на основе анализа физических адресов компонентов объектов. Обсуждается невозможность сэкономить исследованные затраты оперативной памяти с помощью переопределения методов, виртуальности методов, а также с помощью закрытия доступа к компонентам классов.

Ключевые слова: Наследование классов, объём используемой оперативной памяти, иерархия наследования, переопределение, виртуальный, закрытый доступ.

В настоящей статье приведены результаты исследований по использованию оперативной памяти объектно-ориентированных программой. Несмотря на известный ответ на вопрос о влиянии наследования классов на рост используемой оперативной памяти (см., например, [1–2]), и по сию пору среди широкого круга начинающих программистов отсутствует единое мнение по этому вопросу, а часто имеет место ложная убеждённость о том, что наследование имеет только положительное влияние на программу без каких-либо возможных отрицательных последствий. Поводом к проведению самостоятельных исследований стал спор автора со студентами по этому вопросу. Важной особенностью исследований, результаты которых представлены в настоящей статье, является возможность без труда воспроизвести их любому читателю.

Вышеназванные исследования были проведены на персональном компьютере с операционной системой Microsoft Windows XP Professional SP2, в среде разработки Borland C++ 3.1. Для получения информации об объёме использованной программой оперативной памяти применялась встроенная функция system(…) из библиотеки stdlib.h [3–4]. Синтаксис вызываемой функции **system** в среде Borland C++ 3.1 следующий:

system(«mem /c >> c:\\Temp\\test.txt»);

Здесь строка «mem /c >> c:\\Temp\\test.txt» передаётся операционной среде для выполнения. При этом команда «**mem /c** … « в начале строки требует сформировать текстовый файл с информацией об объёме и структуре оперативной памяти, использованной вызывающей программой, а команда в конце этой строки «… **>> c:\\Temp\\test.txt**» задаёт адрес, куда следует поместить этот текстовый файл. В данном случае текстовый файл предлагается назвать test.txt и поместить на диск C: в папку Temp.

Команда (функция) **system** известна программистам, но имеет ограниченное использование. Это происходит из-за того, что, с одной стороны, названная функция **system** выдаёт информацию об используемой оперативной памяти с высокой точностью (до одного байта), а, с другой стороны, эта же функция **system** реагирует на изменение объёма используемой программой оперативной памяти только тогда, когда это изменение превысит один килобайт.

В Примере 1 приведён текст простейшей объектно-ориентированной программы, демонстрирующий класс

с одним полем данных и с конструктором «по умолчанию», задающим это поле данных. На основе этого класса создан единственный объект, после чего вызывается библиотечная функция system:

```
#include <stdlib.h>
class A {char a; public: A(){a=1;}};
void main() {A a;
system(«mem /c >> c:\\Temp\\test_op1.txt»);};
```

Пример 1. Программа, использующая один «пустой» класс и один объект класса

Результат работы функции system в этом примере — текстовый файл в Примере 2.

Подчёркнутая и выделенная жирным шрифтом строка в файле *test_op1.txt*, начинающаяся именем **TST_OP1**, содержит информацию об объёме оперативной памяти, затраченной во время выполнения программы из Примера 1. Видно, что размер затраченной оперативной памяти выводится в байтах и (округлённо) в килобайтах. Следует предположить, что с точностью до байта возможно обнаружить различие в затратах оперативной памяти разными программами. Однако следует обратить особое внимание на то, что обнаружить различие в затратах оперативной памяти двумя разными программами возможно лишь тогда, когда это различие достигает размера в 1 килобайт. Этот тезис подтверждают многократные эксперименты, в том числе Пример 3.

```
Обычная память:
Имя Размер (десятичный) Размер (Hex)
MSDOS11952 (11.7K) 2EB0
KBD3296 (3.2K) CE0
HIMEM 1248 (1.2K) 4E0
COMMAND3984 (3.9K) F90
dpmiload 36144 (35.3K) 8D30
DOSX 34720 (33.9K) 87A0
BC483664 (472.3K) 76150
KB166096 (6.0K) 17D0
TST_OP114208 (13.9K) 3780
COMMAND4640 (4.5K) 1220
СВОБОДНО 16 (0.0K) 10
СВОБОДНО 128 (0.1K) 80
СВОБОДНО 54784 (53.5K) D600
Всего СВОБОДНО: 54928 (53.6K)
Верхняя память:
Имя Размер (десятичный) Размер (Hex)
SYSTEM 229360 (224.0K) 37FF0
DOSX 128 (0.1K) 80
MOUSE12528 (12.2K) 30F0
MSCDEXNT 464 (0.5K) 1D0
REDIR2672 (2.6K) A70
СВОБОДНО 864 (0.8K) 360
СВОБОДНО 35056 (34.2K) 88F0
СВОБОДНО 46448 (45.4K) B570
Всего СВОБОДНО: 82368 (80.4K)
Всего доступно программам (обычной + верхней памяти): 137296 (134.1K)
Максимальный размер программы: 53888 (52.6K)
Максимальный размер блока верхней памяти: 46448 (45.4K)
1048576 байт — всего непрерывной дополнительной памяти
0 байт — доступно непрерывной дополнительной памяти
941056 байт — доступной памяти XMS
резидентная часть MS-DOS загружена в сегмент HMA
```

Пример 2. Файл *test_op1.txt.* Выходные данные о распределении оперативной

памяти компьютера, используемой при работе программы из Примера 1

```
#include <stdlib.h>
void main() {system(«mem /c >> c:\\Temp\\test_op0.
txt»);};
```

Пример 3. Простейшая программа, вызывающая system.

Объём оперативной памяти, используемый этой программой (Пример 3), согласно полученным результатам, оказывается также равен 14208 байт (13,9 К), как и у программы из Примера 1. (Текстовые файлы, аналогичные файлу из Примера 2, далее в тексте статьи не приводятся, чтобы уменьшить объём статьи. Их легко получить для подтверждения изложенных результатов при самостоятельном моделировании.) Хотя программы в примерах 1 и 3 не совпадают (в Примере 1 есть один непустой класс и создан один объект, чего нет в Примере 3), данные о затратах оперативной памяти совпадают. Такие же результаты получены для иерархии из вышеназванного класса-предка A и его класса-наследника B, также имеющего одно поле и конструктор. Если у такой иерархии создать одни объект (экземпляр класса B), то, согласно данным из файла-отчёта, снова требуется только 13,9 К (14208 байт), как и в программах из Примера 1 и Примера 3.

В то же время наличие иерархии наследования из трёх простейших классов A, B и C с конструкторами «по умолчанию», где класс A — предок класса B, а класс B — предок класса C (Пример 4), приводит к результату в 14,9 К (15232 байт).

Такой же объём в 14,9К требует программа с количеством аналогичных классов в иерархии от четырёх

до 17-и. Если же количество таких классов в иерархии наследования увеличить сразу до восемнадцати (см. Пример 5), то используемый программой объём оперативной памяти возрастёт до 15,9К (16256 байт). Когда количество таких классов в иерархии достигает 33-х, то объём оперативной памяти, используемый программой, возрастёт до — 17280 байт (16,9 К). Иерархия наследования с числом таких классов от 19-и до 32-х выдаст 15,9К.

```
#include <stdlib.h>
class A {char a; public: A(){a=1;};};
class B: public A {char b; public: B(){b=2;};};
class C: public B {char c; public: C(){c=3;};};
void main() {C c;
            system(«mem /c >> c:\\Temp\\test_op2.
            txt»);};
```

Пример 4. Программа, использующая иерархию наследования трёх классов

и один объект — экземпляр младшего класса-наследника.

```
#include <stdlib.h>
class A{char a; public: A(){a=1;}};
class B: public A{char b; public: B(){b=2;}};
…… // всего 18 классов иерархии наследования

class R: public Q{char r; public: R(){r=18}};
void main() {R robj;
             system(«mem /c >> c:\\Temp\\tst_op3.
             txt»);};
```

Пример 5. Программа, использующая иерархию наследования из восемнадцати

классов и один объект — экземпляр младшего класса-наследника

Результаты, приведённые в примерах 1–5, показывают, что информация об используемой программами оперативной памяти, представляемая в текстовом файле при использовании команды **system**, меняется, очевидно, с минимальным шагом в 1 килобайт. Именно поэтому дальнейшая информация об используемом объёме оперативной памяти будет приводиться в килобайтах, а не в байтах.

Также на основании результатов из примеров 1–5 справедливо сделать однозначный вывод, что наследование классов приводит к увеличению объёма оперативной памяти, используемой объектно-ориентированной программой.

Важно заметить, что все классы в рассматриваемых примерах имеют закрытые по умолчанию поля данных, к которым нет доступа при наследовании. Однако, как показали эксперименты, если эти поля сделать открытыми, то результаты не изменятся. Это означает, что при наследовании копируются как открытые, так и защищённые компоненты классов-предков. Такие выводы совпадают с выводами статьи «Исследование распределения памяти компьютера при объектно-ориентированном программировании» (авторы — Далека В.Д., Вдовиченко С.С.): *«Простое наследование (как открытое, так и закрытое) приводит к копированию всех открытых, закрытых и защищённых данных базового класса в производный класс. Таким образом, размер объекта производного класса определяется суммарным размером данных — членов базового и этого производного класса.» … «Многоуровневое наследование приводит к копированию всех открытых, закрытых и защищённых данных из базовых классов в производный класс. При этом наследовании копирование данных производится последовательно, начиная от базового класса до производного через промежуточные, добавляя к данным-членам каждого предыдущего класса данные последующего.»* ([2], с. 51–52, п. 4–5).

Авторы статьи [2] использовали другой инструмент исследования. Работая с той же операционной системой Microsoft Windows XP Professional SP2, но со средой разработки Microsoft Visual Studio 98, они изучали шестнадцатеричные коды адресов отдельных компонент объектов («данных-членов» объектов, как их называют в C++ [2–5]). На основании анализа последовательности физических адресов оперативной памяти авторы [2] сделали выводы, совпавшие с выводами настоящей статьи. Одинаковые результаты, полученные разными способами на разных средах разработки, ещё раз подтверждают истинность и общность выводов этой статьи.

Если немного усложнить наследуемые классы, добавив в них по одному методу, изменяющему значение единственного поля данных, то можно получить новые результаты к уже полученным. Далее программах (см. примеры 6–9) введение наследования классов ведёт к увеличению размера используемой программой оперативной памяти с 13,9 К до 15,9 К. Этот размер может быть таким:

а1) 13,9 К (Пример 6), когда представленные в примере 16 классов не связаны в иерархии наследования, один из этих классов имеет свой экземпляр (объект) и этот объект вызывает единственный метод своего класса (не конструктор).

б1) 14,9 К (Пример 7), когда все эти 16 классов объединены в иерархию наследования, и последний в иерар-

хии класс (последний наследник) имеет свой экземпляр (объект), а этот объект последнего наследника вызывает свой единственный метод.

В другом случае размер используемой оперативной памяти меняется так:

а2) 14,9 К (Пример 8), когда представленные в примере 16 классов не связаны в иерархии наследования, но каждый класс имеет свой экземпляр (объект) и объект одного из классов вызывает свой единственный метод (не конструктор).

```
#include <stdlib.h>
class A{public: char a; A(){a=1;}; void pa(char na)
{a=na;};};
class B{public: char b; B(){b=2;}; void pb(char nb)
{b=nb;};};
……. // всего 16 независимых классов ……..
class P{public: char p; P(){p=16;}; void pp(char np)
{p=np;};};
void main() {P p; p.pp(32);
            system(«mem /c >> c:\\Temp\\tst_op4.
            txt»);};
```

Пример 6. Шестнадцать классов без наследования, один объект одного класса

```
#include <stdlib.h>
class A {public: char a; A(){a=1;}; void pa(char na)
{a=na;};};
class B: public A {public: char b; B(){b=2;}; void
pb(char nb){b=nb;};};
…….. // всего 16 классов в иерархии наследования
class P: public O {public: char p; P(){p=16;}; void
pp(char np){p=np;};};
void main(){P p;
p.pp(32);
system(«mem /c >> c:\\Temp\\tst_op5.txt»);};
```

Пример 7. Шестнадцать классов с наследованием. Объект у младшего наследника

б2) 15,9 К (Пример 9), когда вышеназванные 16 классов объединены в иерархию наследования, и каждый имеет свой экземпляр (объект), а объект последнего класса-наследника вызывает свой единственный метод (не конструктор).

```
#include <stdlib.h>
class A{public: char a; A(){a=1;}; void pa(char na)
{a=na;};};
class B{public: char b; B(){b=2;}; void pb(char nb)
{b=nb;};};
…….. // всего 16 независимых классов
class P{public: char p; P(){p=16;}; void pp(char np)
{p=np;};};
void main() {A a; B b; C c; D d; E e; F f; G g; H h;
            I i; J j; K k; L l; M m; N n; O o; P p;
            p.pp(32);
            system(«mem /c >> c:\\Temp\\tst_op6.
            txt»);};
```

Пример 8. Шестнадцать классов без наследования и шестнадцать объектов

этих классов. Вызов метода одним из объектов

```
#include <stdlib.h>
class A {public: char a; A(){a=1;}; void pa(char na)
{a=na;};};
class B: public A {public: char b; B(){b=2;}; void
pb(char nb){b=nb;};};
…….. // всего 16 классов в иерархии наследования
class P: public O {public: char p; P(){p=16;}; void
pp(char np){p=np;};};
void main() {A a; B b; C c; D d; E e; F f; G g; H h;
            I i; J j; K k; L l; M m; N n; O o; P p;
            p.pp(32);
            system(«mem /c >> c:\\Temp\\tst_op7.
            txt»);};
```

Пример 9. Шестнадцать классов с наследованием и шестнадцать объектов

этих классов. Вызов метода объектом последнего класса в иерархии

Эти результаты наглядно демонстрируют то, как наследование увеличивает объём используемой программой оперативной памяти. Одновременно с этим видно как увеличивается размер оперативной памяти при увеличении количества объектов. Весьма важные выводы можно сделать и из результата Примера 10.

В этом случае объём используемой программой оперативной достигает 15,9К, когда 16 классов не связаны в иерархии наследования, но каждый класс имеет свой экземпляр (объект) и каждый объект вызывает свой единственный метод (не конструктор). Сравнивая это с результатом (а1–Пример 6) можно видеть: вызов объектами методов своего класса приводит к росту занятой оперативной памяти.

```
#include <stdlib.h>
class A{public: char a; A(){a=1;}; void pa(char na)
{a=na;};};
class B{public: char b; B(){b=2;}; void pb(char nb)
{b=nb;};};
```

```
…… // всего 16 независимых классов
class P{public: char p; P(){p=16;}; void pp(char np)
{p=np;};};
void main(){A a; B b; C c; D d; E e; F f; G g; H h;
            I i; J j; K k; L l; M m; N n; O o; P p;
            a.pa(17); b.pb(18); c.pc(19); d.pd(20);
            e.pe(21); f.pf(22); g.pg(23); h.ph(24);
            i.pi(25); j.pj(26); k.pk(27); l.pl(28);
            m.pm(29); n.pn(30); o.po(31); p.pp(32);
            system(«mem /c >> c:\\Temp\\tst_op8.
            txt»);};
```

Пример 10. Шестнадцать классов без наследования и шестнадцать объектов этих классов. Вызовы методов каждым объектом

При увеличении числа классов до 26 используемая память растёт с 13,9К до 16,9К:

а3) 13,9 К, когда 26 классов не связаны иерархией наследования. Один из классов имеет единственный объект, который вызывает единственный метод (аналогично см. Пример 1 и Пример 6) Очевидно, что простое увеличение количества классов, не связанных между собой наследованием без создания экземпляров этих классов, не ведёт к увеличению затрат оперативной памяти.

```
#include <stdlib.h> // наследование без переопределения методов
class A{public: char a; A(){a=1;}; void a(char na)
{a=na;};};
class B: public A{public: char b; B(){b=2;}; void
b(char na) {b=na;};};
……// всего 26 классов в иерархии наследования
class Z: public Y{public: char z; Z(){z=1;}; void
z(char na){z=na;};};
void main(){a; B b; C c; D d; E e; F f; G g; H h; I i;
            J j; K k; L l; M m;
            N n; O o; P p; Q q; R r; S s; T t; U u; V
            v; W w; X x; Y y; Z z;
            a.a(17); b.b(18); c.c(19); d.d(20);
            e.e(21); f.f(22); g.g(23); h.h(24);
            i.i(25);
            j.j(26); k.k(27); l.l(28); m.m(29);
            n.n(30); o.o(31); p.p(32); q.q(17); r.r(18);
            s.s(19); t.t(20); u.u(21); v.v(22);
            w.w(23); x.x(24); y.y(25); z.z(26);
            system(«mem /c >> c:\\Temp\\tst_op9.
            txt»);};
```

Пример 11.Двадцать шесть классов с наследованием без переопределения методов.

Один объект у каждого класса. Вызов метода своего класса каждым объектом

б3) 16,9 К (Пример 11), когда 26 классов в иерархии наследования, каждый класс имеет свой экземпляр (объект) и каждый объект вызывает свой единственный метод (не конструктор).в3) 16,9 К (Пример 12), когда 26 классов в иерархии наследования, каждый класс имеет свой экземпляр (объект), единственный метод переопределяется во всех классах, и каждый объект вызывает единственный переопределённый метод.

г3) 16,9 К (Пример 13), когда 26 классов в иерархии наследования, каждый класс имеет свой экземпляр (объект), единственный метод переопределяется во всех классах и объявляется как виртуальный, каждый объект вызывает единственный переопределённый метод.

```
#include <stdlib.h> // наследование с переопределением
class A{public: char a; A(){a=1;}; void p(char na)
{a=na;};};
class B: public A{public: char b; B(){b=2;}; void
p(char na) {b=na;};};
…… // всего 26 классов в иерархии наследования
class Z: public Y{public: char z; Z(){z=1;}; void
p(char na){z=na;};};
void main() {A a; B b; C c; D d; E e; F f; G g; H h;
            I i; J j; K k; L l; M m;
            N n; O o; P p; Q q; R r; S s; T t; U u;
            V v; W w; X x; Y y; Z z;
            a.p(17); b.p(18); c.p(19); d.p(20);
            e.p(21); g.p(22); g.p(23); h.p(24);
            i.p(25);
            j.p(26); k.p(27); l.p(28); m.p(29);
            n.p(30); o.p(31); p.p(32); q.p(17);
            r.p(18);
            s.p(19); t.p(20); u.p(21); v.p(22);
            w.p(23); x.p(24); y.p(25); z.p(26);
            system(«mem /c >> c:\\Temp\\tst_op9.
            txt»);};
```

Пример 12. Двадцать шесть классов с наследованием и переопределением методов.

Один объект у каждого класса. Вызов переопределённого метода каждым объектом

д3) 16,9 К (Пример 14), когда 26 классов в иерархии наследования, последний класс имеет свой экземпляр (объект) и этот объект вызывает каждый унаследованный метод.

е3) 16,9 К (Пример 15), когда 26 классов в иерархии наследования, единственный метод переопределяется во всех классах, последний класс имеет свой экземпляр (объект) и этот объект вызывает унаследованный

метод столько раз, сколько классов в иерархии. Этот искусственный приём используется для того чтобы сравнить с такой же программой, но без переопределения, из Примера 14.

```
#include <stdlib.h> // наследование с переопределением и виртуальными методами
class A{public: char a; A(){a=1;}; virtual void p(char na) {a=na;};};
class B: public A{public: char b; B(){b=2;}; void p(char na) {b=na;};};
…… // всего 26 классов в иерархии наследования
class Z: public Y{public: char z; Z(){z=1;}; void p(char na){z=na;};};
void main() {a; B b; C c; D d; E e; F f; G g; H h;
            I i; J j; K k; L l; M m;
            N n; O o; P p; Q q; R r; S s; T t; U u;
            V v; W w; X x; Y y; Z z;
            a.p(17); b.p(18); c.p(19); d.p(20);
            e.p(21); g.p(22); g.p(23); h.p(24); i.p(25);
            j.p(26); k.p(27); l.p(28); m.p(29);
            n.p(30); o.p(31); p.p(32); q.p(17); r.p(18);
            s.p(19); t.p(20); u.p(21); v.p(22);
            w.p(23); x.p(24); y.p(25); z.p(26);
            system(«mem /c >> c:\\Temp\\tst_op9.
            txt»);};
```

Пример 13. Двадцать шесть классов с наследованием, с виртуальными методами. Один объект у каждого класса. Вызов виртуального метода каждым объектом

```
#include <stdlib.h> // наследование без переопределения методов
class A{public: char a; A(){a=1;}; void a(char na) {a=na;};};
class B: public A{public: char b; B(){b=2;}; void b(char na) {b=na;};};
…… // всего 26 классов в иерархии наследования
class Z: public Y{public: char z; Z(){z=1;}; void z(char na){z=na;};};
void main(){Z z;
            z.a(17); z.b(18); z.c(19); z.d(20);
            z.e(21); z.f(22); z.g(23); z.h(24);
            z.i(25);
            z.j(26); z.k(27); z.l(28); z.m(29);
            z.n(30); z.o(31); z.p(32); z.q(17); z.r(18);
            z.s(19); z.t(20); z.u(21); z.v(22);
            z.w(23); z.x(24); z.y(25); z.z(26);
            system(«mem /c >> c:\\Temp\\tst_op9.
            txt»);};
```

Пример 14.Двадцать шесть классов с наследованием без переопределения методов.

Один объект у каждого класса. Вызов метода своего класса каждым объектом

ж3) 15,9 К (Пример 16), когда 26 классов в иерархии наследования, единственный метод переопределяется во всех классах, последний класс имеет свой экземпляр (объект) и этот объект вызывает многократно переопределённый метод один раз.

Результаты примеров 12 и 16, где используется переопределение методов, (переопределение метода, вызываемого объектом, представляет собой замещение метода класса-предка методом класса-наследника, имеющим то же имя и ту же сигнатуру) при сравнении с результатами примера 11 и примера 15 показывают, что переопределение не приводит к снижению размера используемой оперативной памяти.

```
#include <stdlib.h> //наследование с переопределением (override)
class A{public: char a; A(){a=1;}; void p(char na) {a=na;};};
class B: public A{public: char b; B(){b=2;}; void p(char na) {b=na;};};
…… // всего 26 классов в иерархии наследования
class Z: public Y{public: char z; Z(){z=1;}; void p(char na){z=na;};};
void main() {Z z;
            for (int i=1; i<=26; i++) z.p(i);
            system(«mem /c >> c:\\Temp\\tst_op9.
            txt»);};
```

Пример 15. 26-ть классов с наследованием и переопределением. Один объект младшего класса. Вызов метода объектом столько раз, сколько классов в иерархии

По этому поводу Далека В.Д. и Вдовиченко С.С говорят: «В случае замещения данных базового класса в объект производного класса включаются как замещённая, так и замещающая версия этих данных, что отражается на его размере и структуре выделенной для него памяти.» (см.[2], с. 51, п. 4) Также говорится: «При замещении данных-членов одного из классов, стоящих выше в иерархии наследования, в текущий и во все нижестоящие классы, копируется как замещённая, так и замещающая копии данного. В результате размер объекта класса, находящегося в конце цепочки наследования, равен сумме размеров всех данных-членов объявленных в каждом из вышестоящих классов, включая замещающие данные». (см. [2], с. 51–52, п. 5)

```
#include <stdlib.h> //наследование с переопределением методов
```

```
class A{public: char a; A(){a=1;}; void p(char na)
{a=na;};};
class B: public A{public: char b; B(){b=2;}; void
p(char na) {b=na;};};
…… // всего 26 классов в иерархии
наследования
class Z: public Y{public: char z; Z(){z=1;}; void
p(char na){z=na;};};
void main(){Z z;
        z.p(26);
        system(«mem /c >> c:\\Temp\\tst_op9.
        txt»);};
```

Пример 16.Двадцать шесть классов с наследованием и переопределением методов. Один объект у младшего наследника. Вызов объектом переопределённого метода

Увеличение размера оперативной памяти, используемой объектно-ориентированной программой при применении механизма наследовании классов изначально было известно специалистам в программировании. Однако в 90-е годы, в период «программного голода», когда большая часть прикладных задач программирования только находилась в стадии решения, скорость разработки программных продуктов считалась важнее аппаратных затрат. Всякая впервые решённая прикладная задача делала решающую её программу единственной в своём роде. По этому вопросу Т. Бадд в [1] в 1997 году пишет: «Использование любой программной библиотеки часто приводит к увеличению размера программ *(размера оперативной памяти, как следует из контекста* — примечание автора*)*… Хотя такие затраты могут быть существенными, по мере уменьшения стоимости памяти размер программы перестаёт быть критичным.» (см. [1], с. 160, п. 7.6.2). Но там же: «Снизить затраты на разработку и быстро выдать высококачественный и свободный от ошибок программный код значит сейчас гораздо больше, чем малый размер приложения (*малый размер используемой памяти, как следует из контекста* — Прим. автора)».

Ныне же возрастающая конкуренция между программными продуктами ставит на первый план вопросы оптимизации программ по различным критериям.

Так, в противовес Т. Бадду, Алан Голуб утверждает: «Разбухание программ является огромной проблемой. … бóльшая часть этого разбухания памяти является результатом небрежного программирования.» «Если только вы не проникнитесь сознанием необходимости дисциплинировать самого себя, то можете закончить гигантским модулем из неподдающейся сопровождению тарабарщины, только притворяющейся компьютерной программой.» ([1], с. 160, *Прим. Перев.*, также [5]).

Итак, в настоящей статье рассмотрен способ исследования затрат оперативной памяти в среде Borland C++ с помощью библиотечной функции system.

Показано, что функция system реагирует на изменение объёма оперативной памяти с дискретностью 1 Кбайт: новый размер используемой оперативной памяти появляется в выходном текстовом файле только тогда, когда новая величина затрат оперативной памяти превысит старую величину не менее чем на один килобайт.

Однозначно показано, что при увеличении количества идентичных классов в иерархии растут затраты оперативной памяти. Для этого намеренно использованы классы с одним полем данных (размером в 1 байт) и с одним конструктором.

Видно, что переопределение методов и объявление этих методов виртуальными, также, как и закрытие доступа к компонентам классов, не позволяют сэкономить затраты на оперативную память, возросшие при наследовании классов.

Установлено совпадение результатов исследований в настоящей статье с результатами альтернативных исследований в среде Microsoft Visual Studio 98 на основе анализа физических адресов компонентов объектов.

ЛИТЕРАТУРА

1. Бадд Т. «Объектно-ориентированное программирование в действии».— СПб, Питер, 1997.— 464 с.
2. Далека В. Д., Вдовиченко С. С. «Исследование распределения памяти компьютера при объектно-ориентированном программировании».— Харьков, Вестник НТУ «Харьковский политехнический институт», Сер. Информатика и моделирование, 2007,48–53
3. Borland C++ v.3.1 Library Reference.
4. Страуструп Б. «Программирование. Принципы и практика использования C++», 2-е издание.— М.: Вильямс, 2015, 1328 с.
5. Алан И. Голуб «Верёвка достаточной длины, чтобы выстрелить себе в ногу. Правила программирования на C и C++».— М.: Бином, 2001, 171 с.

Журнал «Современная наука: актуальные проблемы теории и практики»