

Глава втора

Класове

Дефиниране на класове. Обекти. Конструктори. Указатели към обекти на класове. Массиви и обекти. Динамични обекти. Деструктори. Създаване и разрушаване на обекти на класове. Приятелски функции и класове. Оператори. Предефиниране на оператори. Указатели към компоненти на класове. Канонично представяне. Шаблони на функции и класове. Статични член-данни и член-функции

Дефиниране на класове. Обекти

Задача 1. Да се намери резултатът от изпълнението на програмата:

```
#include <iostream.h>
class A
{ public:
    void Hello();
};
void A::Hello()
{ cout << "Hello world!\n";
}
int main()
{ A e;
  e.Hello();
  return 0;
}
```

Решение: В програмата е дефиниран клас с една член-функция с име Hello в public секцията. В главната функция е дефиниран обектът е и чрез него е активирана член-функцията Hello(). В резултат на екрана се извежда текстът
Hello world!

Задача 2. Да се намери резултатът от изпълнението на програмата:

```
#include <iostream.h>
class B
{ public:
    int a, b;
    void init(int, int);
};
void B::init(int x, int y)
{ a = x;
```

```

        b = y;
    }
    int main()
    { B c;
      c.init(5, 8);
      cout << c.a << " " << c.b << endl;
      return 0;
    }

```

Решение: В програмата е дефиниран клас с две член-данни a и b от тип int и една член-функция с име init. И трите компоненти са декларираны в public секцията. В главната функция е дефиниран обектът c и е инициализиран чрез член-функцията init(). В резултат от това заделена е паметта за обекта c (обикновено по 4 байта за всяка член-данна) се инициализира, т.е.

```

c.a  c.b
5    8

```

Тъй като член-данните a и b са декларираны в public секцията на класа, възможен е пряк достъп до тях в главната функция main(). В резултат стойностите на член-данните на обекта c се извеждат на екрана и резултатът е:

```

5    8

```

Задача 3. Да се намери резултатът от изпълнението на програмата:

```

#include <iostream.h>
class C
{ private:
  int a, b;
public:
  void init(int, int);
  void display() const;
};
void C::init(int x, int y)
{ a = x;
  b = y;
}
void C::display() const
{ cout << a << " " << b << endl;
}
int main()
{ C c;
  c.init(5, 8);
  c.display();
  return 0;
}

```

Решение: В програмата е дефиниран клас с две член-данни a и b от тип int и две член-

функции с имена `init` и `display`. Член-данните са декларирани в `private` секцията, а член-функциите – в `public` секцията. В главната функция е дефиниран обектът `c` и е инициализиран чрез член-функцията `init()`. В резултат от това заделената за обекта `c` памет се инициализира, т.е.

```
c.a  c.b
5    8
```

Тъй като член-данните `a` и `b` са декларирани в `private` секцията на класа, не е възможен пряк достъп до тях в главната функция `main()`. Чрез член-функцията `display()` стойностите на член-данните на обекта `c` се извеждат на екрана. Резултатът от изпълнението е:

```
5    8
```

Задача 4. Да се дефинира повторно класът `C` от предходната задача като член-функциите му са вградени.

Решение:

първи начин:

```
class C
{ private:
    int a, b;
public:
    void init(int x, int y)
    { a = x;
      b = y;
    }
    void display() const
    { cout << a << " " << b << endl;
    }
};
```

втори начин:

```
class C
{ private:
    int a, b;
public:
    void init(int, int);
    void display() const;
};
inline void C::init(int x, int y)
{ a = x;
  b = y;
}
inline void C::display() const
{ cout << a << " " << b << endl;
}
```

Задача 5. Да се намерят и обяснят грешките в програмата:

```
#include <iostream.h>

void f()
{ class D
  { private:
    int a, b;
  public:
    void init(int, int);
    void display() const;
  };
  inline void D::init(int x, int y)
  { a = x;
    b = y;
  }
  inline void D::display() const
  { cout << a << " " << b << endl;
  }
  D e;
  e.init(12, 15);
  e.display();
}

int main()
{ D e;
  e.init(2, 5);
  e.display();
  f();
  return 0;
}
```

Решение: Тъй като класът D е дефиниран в тялото на функцията f, всички негови член-функции трябва да са дефинирани вътре в класа. В случая дефинициите на init и display са вградени, но дефинициите им са извън класа, което води до грешки. Освен това областта на класа D е тялото на функцията f(). Опитът за използването му в главната функция main() предизвиква грешки.

Задача 6. Да се дефинира клас angle, чрез който се намират синусът, косинусът и тангенсът на ъгъл, зададен в градуси.

Решение: Намирането на синуса, косинуса и тангенса на ъгъл, зададен чрез градуси се реализира от функциите за достъп getSin(), getCos() и getTan() на класа. Последните използват библиотечните функции sin, cos и tan след като преобразуват градусите в радиани.

```

#include <iostream.h>
#include <math.h>
const double DEG_TO_RAD = 3.14159265/180;
class angle
{ public:
    void setValue(double); // инициализиране на degrees
    double getSin() const; // намиране на sin
    double getCos() const; // намиране на cos
    double getTan() const; // намиране на tan
private:
    double degrees;
};

void angle::setValue(double a)
{ degrees = a;
}

double angle::getSin() const
{ return sin(DEG_TO_RAD * degrees);
}

double angle::getCos() const
{ return cos(DEG_TO_RAD * degrees);
}

double angle::getTan() const
{ return tan(DEG_TO_RAD * degrees);
}

int main()
{ // за ъгъл от 45 градуса се
  // намират sin, cos и tan
  angle deg;
  deg.setValue(45);
  cout << "sin(45) = " << deg.getSin() << endl
        << "cos(45) = " << deg.getCos() << endl
        << "tan(45) = " << deg.getTan() << endl;
  return 0;
}

```

Задача 7. Да се дефинира клас Employee, който пресмята заплата на чиновник, дефиниран чрез:

- име, зададено чрез структура с полета first (собствено име) и last (фамилно име) и
- работа, зададена чрез структура с полета hours (часове) и salary (заплащане за 1 час).

Решение: Класът Employee притежава две член-данни: name и work и две член-функции: read() и print(). Член-данните са структури. Член-функцията print() на класа намира и извежда заплата на чиновник.

```

#include <iostream.h>
#include <iomanip.h>
class Employee
{ public:
    void read();           // въвеждане на данни за чиновник
    void print() const;    // извеждане на данни за чиновник
private:
    struct
    { char first[16];      // собствено име
      char last[16];      // фамилно име
    } name;
    struct
    { double hours; // часове на работа
      double salary; // заплата за 1 час
    } work;
};

void Employee::read()
{ cout << "Собствено име: ";
  cin >> name.first;
  cout << "Фамилно име: ";
  cin >> name.last;
  cout << "Часове на работа: ";
  cin >> work.hours;
  cout << "Заплащане за 1 час: ";
  cin >> work.salary;
  cin.ignore(); // прескачане на знаците за край на ред
}

void Employee::print() const
{ cout << setiosflags(ios::fixed) << setprecision(2);
  cout << name.first << " " << name.last << " "
    << work.hours * work.salary << endl;
}

int main()
{ // дефиниране на чиновник
  Employee e;
  e.read();
  // извеждане на името и заплата на чиновник
  e.print();
  return 0;
}

```

Задача 8. Да се намери резултатът от изпълнението на програмата:

```
#include <iostream.h>
class E
{ private:
    int a, b;
public:
    void init();
    void init(int);
    void init(int, int);
    void display() const;
};

void E::init()
{ a = b = 0;
}

void E::init(int x)
{ a = x;
}

void E::init(int x, int y)
{ a = x;
  b = y;
}

void E::display() const
{ cout << a << " " << b << endl;
}

int main()
{ E e1, e2, e3;
  e1.init();
  e2.init(1);
  e3.init(1, 2);
  e1.display();
  e2.display();
  e3.display();
  return 0;
}
```

Решение: В резултат от декларацията

E e1, e2, e3;

за обектите e1, e2 и e3 се отделят по 8 байта (обикновено по 4 байта за всяка член-данна), която памет е неинициализирана

e1.a	e1.b	e2.a	e2.b	e3.a	e3.b
-	-	-	-	-	-

Член-функцията init на класа E е предефинирана. Всеки от обектите е инициализиран чрез прилагане на някоя от член-функциите init. В резултат се получава:

e1.a	e1.b	e2.a	e2.b	e3.a	e3.b
0	0	1	-	1	2

Забеляваме, че член-функцията `init(int)` оставя неинициализирана член-данната `b` на обекта `e2`. Обръщенията към член-функцията `display` чрез обектите `e1`, `e2` и `e3` извеждат на екрана стойностите на компонентите им. Резултатът от изпълнението е:

```
0 0
1 <неопределена стойност>
1 2
```

Задача 9. Да се дефинира клас `Absolute`, чрез който да се намира абсолютната стойност на цяло и на реално число.

Решение:

```
#include <iostream.h>
#include <math.h>
class Absolute
{ public:
    int ab(int);
    double ab(double);
};
int Absolute::ab(int x)
{ return abs(x);
}
double Absolute::ab(double x)
{ return fabs(x);
}
int main()
{ Absolute num;
    cout << "Абсолютната стойност на -567 е: "
    << num.ab(-567)
    << endl << "Абсолютната стойност на -5.167 е: "
    << num.ab(-5.167) << endl;
    return 0;
}
```

Конструктори

Задача 10. Да се намери резултатът от изпълнението на програмата:

```
#include <iostream.h>
class F
{ private:
    int a, b;
public:
```

```

    // конструктор по подразбиране
    F();
    // конструктор с един параметър
    F(int);
    // конструктор с два параметъра
    F(int, int);
    // извеждане на член-данните на класа
    void display() const;
};
F::F()
{ a = b = 0; }
F::F(int x)
{ a = x; }
F::F(int x, int y)
{ a = x;
  b = y; }
void F::display() const
{ cout << a << " " << b << endl; }
int main()
{ F e1, e2(1), e3(1, 2);
  e1.display();
  e2.display();
  e3.display();
  return 0;
}

```

Решение: В класа F са дефинирани три конструктора: подразбиращ се, с един и с два явно указани параметри. В резултат от декларацията:

```
F e1, e2(1), e3(1, 2);
```

за обектите e1, e2 и e3 се отделят по 8 байта (обикновено по 4 байта за всяка член-данна). Тази памет се инициализира съответно от конструкторите F(), F(int) и F(int, int), т.е.

e1.a	e1.b	e2.a	e2.b	e3.a	e3.b
0	0	1	-	1	2

Член-данната b на обекта e2 остава неинициализирана. Обръщенията към член-функцията display(), чрез обектите e1, e2 и e3, извеждат на екрана стойностите на компонентите на обектите. Резултатът от изпълнението на програмата е:

```
0 0
```

```
1 <неопределена стойност>
```

```
1 2
```

Задача 11. Да се намери резултатът от изпълнението на програмата:

```
#include <iostream.h>
class G
{ private:
    int a, b;
public:
    // конструктор с подразбиращи се параметри
    G(int = 0, int = 0);
    // извеждане на член-данните на класа
    void display() const;
};
G::G(int x, int y)
{ a = x;
  b = y;
}
void G::display() const
{ cout << a << " " << b << endl;
}
int main()
{ G e1, e2(1), e3(1, 2);
  e1.display();
  e2.display();
  e3.display();
  return 0;
}
```

Решение: В класа G е дефиниран конструктор с два подразбиращи се параметъра. В резултат от декларацията

G e1, e2(1), e3(1, 2);

за обектите e1, e2 и e3 се отделят по 8 байта (обикновено по 4 байта за всяка член-данна). Тази памет се инициализира по следния начин: e1 – с подразбиращите се стойности на параметрите x и y, e2 – с указаната стойност 1 за x и подразбиращата се стойност 0 за y и e3 – с указаните фактически параметри 1 и 2 за формалните параметри x и y, т.е.

e1.a	e1.b	e2.a	e2.b	e3.a	e3.b
0	0	1	0	1	2

Обръщенията към член-функцията display чрез обектите e1, e2 и e3 извеждат на екрана стойностите на компонентите на обектите. Резултатът от изпълнението е:

```
0 0
1 0
1 2
```

Задача 12. Да се дефинира клас `fact` за намиране на факториела на естествено число. Да се напише програма, която като използва класа `fact` намира факториела на дадено естествено число.

Решение:

първи начин

В `public` секцията на класа `fact` са дефинирани конструктор с един явно указан параметър, инициализиращ член-данната `x` с указана стойност и член-функция `value()`, която намира факториела на член-данната `x` на класа. Ще напомним, че пресмятането стойността на функцията факториел е възможно за малки стойности на аргумента.

```
#include <iostream.h>
class fact
{ public:
    // конструктор
    fact(int a)
    { x = a;
    }
    // намиране на x!
    unsigned int value() const
    { int y = 1;
      for(int i = 1; i <= x; i++)
        y *= i;
      return y;
    }
private:
    int x;
};

int main()
{ int a;
  do
  { cout << "a = ";
    cin >> a;
  } while(a <= 0);
  // създаване на обект t от клас fact
  fact t(a);
  // намиране и извеждане на a!
  cout << t.value() << endl;
  return 0;
}
```

втори начин

Член-функцията `value()`, която намира факториела на член-данната `x` на класа `fact`, е реализирана рекурсивно. Обръщението `t.value()` в нея намира $(x-1)!$.

```

#include <iostream.h>
class fact
{ public:
    // конструктор
    fact(int a)
    { x = a;
    }
    // намиране на x!
    unsigned int value() const
    { if(x == 1) return 1;
      else
      { fact t(x-1);
        return x*t.value();
      }
    }
private:
    int x;
};

int main()
{ int a;
  do
  { cout << "a= "; cin >> a;
  } while(a <= 0);
  fact t(a);
  cout << t.value() << endl;
  return 0;
}

```

Задача 13. Да се дефинира клас gcd, който намира най-големия общ делител на две неотрицателни цели числа, поне едното от които е различно от 0. Да се напише програма, която като използва класа gcd намира най-големия общ делител на две неотрицателни цели числа, поне едното от които е различно от 0.

Решение: Член-функцията value() на класа gcd реализира следната рекурсивна функция за намиране на най-голям общ делител на две неотрицателни цели числа, поне едното от които е различно от 0:

```

int gcd(int x, int y)
{ if(y == 0) return x;
  return gcd(y, x % y);
}

```

```

#include <iostream.h>

```

```

class gcd
{ public:
    // конструктор
    gcd(int a, int b)
    { x = a;
      y = b;
    }
    // намиране на най-големия общ делител на x и y
    int value() const
    { if(y == 0) return x;
      else
      { gcd t(y, x%y);
        return t.value();
      }
    }
private:
    int x, y;
};

int main()
{ int a, b;
  do
  { cout << "a= "; cin >> a;
    cout << "b= "; cin >> b;
  } while(a < 0 || b < 0 || a == 0 && b == 0);
  // създаване на обект от клас gcd
  gcd t(a,b);
  // намиране и извеждане на най-големия
  // общ делител на a и b
  cout << t.value() << endl;
  return 0;
}

```

Задача 14. Да се дефинира клас Expr, чрез който да могат да се намират стойностите на изразите:

$$S1 = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \dots + \frac{x^n}{n!};$$

$$S2 = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \dots + (-1)^n \frac{x^{2n}}{(2n)!};$$

$$S3 = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \dots + (-1)^n \frac{x^{2n+1}}{(2n+1)!};$$

за произволно реално число x от интервала $[-1, 1]$ и $n \geq 0$. Да се напише програма, която като

използва класа Expr намира стойността на израз от горния вид.

Решение: Сумите намират приближено стойностите на функциите e^x , $\cos(x)$ и $\sin(x)$ съответно. В главната функция са включени и фрагменти, чрез които се сравняват получените резултати със съответните от изпълнението на библиотечните функции.

```
#include <iostream.h>
#include <iomanip.h>
#include <math.h>
class Expr
{ public:
    // конструктор
    Expr(int, double);
    // Член-функция за намиране на S1
    double Ex1() const;
    // Член-функция за намиране на S2
    double Ex2() const;
    // Член-функция за намиране на S3
    double Ex3() const;
private:
    int n;
    double x;
};
Expr::Expr(int m, double y)
{ n = m;
  x = y;
}
double Expr::Ex1() const
{ double S = 1, x1 = 1;
  for(int i = 1; i <= n; i++)
  { x1 = x1*x/i;
    S += x1;
  }
  return S;
}
double Expr::Ex2() const
{ double S = 1, x1 = 1;
  for(int i = 1; i <= 2*n; i = i+2)
  { x1 = -x1*x*x/(i*(i+1));
    S += x1;
  }
  return S;
}
```

```
double Expr::Ex3() const
{ double S = x, x1 = x;
  for(int i = 2; i <= 2*n; i = i+2)
  { x1 = -x1*x*x/(i*(i+1));
    S += x1;
  }
  return S;
}

int main()
{ int n;
  do
  { cout << "n= ";
    cin >> n;
  } while(n < 0);
  double x;
  do
  { cout << "x= ";
    cin >> x;
  } while(fabs(x) > 1);
  Expr t(n, x);
  cout << setiosflags(ios::fixed) << setprecision(6);
  cout << "S1= " << t.Ex1() << endl;
  cout << "exp(" << x << ")= " << exp(x) << endl;
  cout << "S2= " << t.Ex2() << endl;
  cout << "cos(" << x << ")= " << cos(x) << endl;
  cout << "S3= " << t.Ex3() << endl;
  cout << "sin(" << x << ")= " << sin(x) << endl;
  return 0;
}
```

Задача 15. Да се дефинира клас differ, който проверява дали елементите на непразна редица от цели числа са различни (няма повтарящ се елемент). Създаването на редицата да се осъществява чрез подходящ конструктор. Да се напише програма, която като използва класа differ проверява дали елементите на редица са различни.

Решение:

```
#include <iostream.h>
const int MAX = 20;
class differ
{ public:
  // конструктор с параметър, задаващ дължината на редица
  differ(int m)
  { n = m;
```

```

for(int i = 0; i < m; i++)
    cin >> a[i];

// функция, която определя дали подразбиращата
// се редица се състои от различни числа
bool diff() const
{ if(n == 1) return true;
  int i = -1;
  bool b; int j;
  do
  { i++; j = i;
    do
    { j++;
      // b = a[i] != a[j];
    } while(b && j < n-1);
    // } while(b && i < n-2);
    return b;
  }
private:
  int n; // брой елементи на редица
  int a[MAX]; // носител на редица
};

int main()
{ // въвеждане на редица
  int n;
  do
  { cout << "n= ";
    cin >> n;
  } while(n < 1);
  // създаване на обект t на класа differ,
  // задаващ непразна редица от цели числа
  differ t(n);
  // проверка дали елементите на редицата,
  // зададена чрез обекта t, са различни
  cout << "Елементите на редицата "
    << (t.diff()? "са различни.\n" : "не са различни.\n");
  return 0;
}

```

Задача 16. Като се използва класът Point, определящ точка от равнината и дефиниран по следния начин:

```

class Point
{ public:

```

```

    Point(double, double);
    double xcoord() const;
    double ycoord() const;
private:
    double x, // абсциса
    y; // ордината
};
Point::Point(double a, double b)
{ x = a;
  y = b;
}
double Point::xcoord() const
{ return x;
}
double Point::ycoord() const
{ return y;
}

```

се дефинират член-функциите на клас Line, определящ права линия, преминаваща през две точки от равнината и деклариран по следния начин:

```

class Line
{ public:
    // конструктор на линия, минаваща
    // през зададени две точки
    Line(const Point&, const Point&);
    // функция за достъп до едната точка
    Point point1() const;
    // функция за достъп до другата точка
    Point point2() const;
    // разстояние между двете точки, през
    // които минава правата линия
    double len() const;
    // намира ъгловия коефициент на линия
    double slope() const;
    // намира пресечната точка на две линии
    Point intersection(const Line&) const;
private:
    Point p1, p2; // точки, определящи линия
};

```

Да се напише програма, която намира ъгловите коефициенти и пресечната точка на линиите, които се явяват диагонали на правоъгълника с върхове: (0, 0), (8, 0), (8, 4) и (0, 4), както и дължината на диагонала на правоъгълника.

Решение:

```

Line::Line(const Point& pt1, const Point& pt2)
    : p1(pt1), p2(pt2)
{}

Point Line::point1() const
{ return p1;
}

Point Line::point2() const
{ return p2;
}

double Line::len() const
{ double xdist = p2.xcoord() - p1.xcoord();
  double ydist = p2.ycoord() - p1.ycoord();
  return sqrt(xdist* xdist + ydist* ydist);
}

double Line::slope() const
{ double xdist = p2.xcoord() - p1.xcoord();
  double ydist = p2.ycoord() - p1.ycoord();
  if(fabs(xdist) < 1E-14)
  { cout << "Ъгловият коефициент е неопределен!\n";
    return 0;
  }
  else return ydist / xdist;
}

Point Line::intersection(const Line& lin) const
{ // преобразуване на първата линия във вида
  //  $a_1x + b_1y + c_1 = 0$ 
  double a1 = p1.ycoord() - p2.ycoord();
  double b1 = p2.xcoord() - p1.xcoord();
  double c1 = p1.xcoord() * p2.ycoord() -
    p2.xcoord() * p1.ycoord();
  // преобразуване на втората линия във вида
  //  $a_2x + b_2y + c_2 = 0$ 
  double a2 = lin.p1.ycoord() - lin.p2.ycoord();
  double b2 = lin.p2.xcoord() - lin.p1.xcoord();
  double c2 = lin.p1.xcoord() * lin.p2.ycoord() -
    lin.p2.xcoord() * lin.p1.ycoord();
  if(fabs(a1*b2 - a2*b1) < 1E-14)
  { if(fabs(b1*c2 - b2*c1) < 1E-14)
    { cout << "Линиите съвпадат!\n";
      return p1;
    }
    else
    { cout << " Линиите са успоредни!\n";

```

```

        return p1;
    }
    else
    {
        double x1 = (b1*c2 - b2*c1)/(a1*b2 - a2*b1);
        double y1 = (a2*c1 - a1*c2)/(a1*b2 - a2*b1);
        return Point(x1, y1);
    }
}

int main()
{
    // създаване на точките - върхове на правоъгълника
    Point P1(0, 0);
    Point P2(8, 0);
    Point P3(8, 4);
    Point P4(0, 4);
    // създаване на диагоналите на правоъгълника
    Line line1(P1, P3);
    Line line2(P2, P4);
    // намиране и извеждане на дължината
    // на диагонала на правоъгълника
    cout << "дължина на диагонала на правоъгълника: "
    << line1.len() << endl;
    // намиране и извеждане на ъгловите коефициенти
    // на диагоналите на правоъгълника
    cout << "ъглови коефициенти: "
    << line1.slope() << " "
    << line2.slope() << endl;
    // намиране и извеждане на координатите на пресечната
    // точка на диагоналите на правоъгълника
    Point intersect = line1.intersection(line2);
    cout << "координати на пресечната точка на диагоналите: "
    << intersect.xcoord() << ", "
    << intersect.ycoord() << endl;
    return 0;
}

```

Задача 17. Да се дефинира клас BankAccount, определящ банкова сметка на клиент, съдържаща: име на клиент (символен низ с дължина до 23 знака), номер на банкова сметка (символен низ с дължина до 15 знака) и налична сума на клиент (реално число). Класът да притежава методи, чрез които: да създава банкова сметка на клиент, да извежда на екрана банкова сметка, да внася и тегли пари от банкова сметка. Да се дефинира главна функция, която демонстрира класа.

Решение:

```
#include <iostream.h>
```

```

#include <string.h>
const int MAX_NAME = 24;
const int MAX_NUM = 16;
class BankAccount
{ private:
    char name[MAX_NAME]; // име на клиент
    char accountnum[MAX_NUM]; // номер на банкова сметка
    double balance; // баланс
public:
    // конструктор
    BankAccount(const char * = "", const char * = "0",
                double = 0.0);
    // член-функция, която извежда банкова сметка
    void print() const;
    // член-функция, която реализира внасяне
    // на сума в банкова сметка
    void put_in(double);
    // член-функция, която реализира теглене
    // на сума от банкова сметка
    void draw(double);
};

BankAccount::BankAccount(const char * cl, const char * n,
                        double b)
{ strncpy(name, cl, MAX_NAME-1);
  name[MAX_NAME-1] = '\0';
  strncpy(accountnum, n, MAX_NUM);
  accountnum[MAX_NUM] = '\0';
  balance = b;
}

void BankAccount::print() const
{ cout << "Име на клиент: " << name << endl;
  cout << "Номер на сметка: " << accountnum << endl;
  cout << "Баланс: " << balance << endl;
}

void BankAccount::put_in(double cash)
{ if(cash >= 0)
  balance = balance + cash;
else
  cout << "Некоректен опит за внасяне на пари.\n";
}

void BankAccount::draw(double cash)
{ if(cash < 0)
  cout << "Некоректен опит за теглене на пари.\n";
}

```

```

else
    if(cash <= balance)
        balance = balance - cash;
    else
        cout << "Сметката не съдържа достатъчно пари.\n";
}
int main()
{ // създаване на банкови сметки x, y и z и
  // извеждане на банковите сметки
  BankAccount x;
  x.print();
  BankAccount y("Peter Ivanov", "PIvanov1923", 6123.65);
  y.print();
  BankAccount z = BankAccount("Georgi Petrov",
                                "GPetrov3917", 819.60);
  z.print();
  // внасяне на сума в банкова сметка
  // x и извеждане на сметката
  x.put_in(20);
  x.print();
  // теглене на суми от банкови сметки y и z
  // и извеждане на сметките
  y.draw(250);
  y.print();
  z.draw(50);
  z.print();
  return 0;
}

```

Задача 18. Да се намери резултатът от изпълнението на програмата:

```

#include <iostream.h>
class H
{ private:
    int a, b;
public:
    // конструктор с подразбиращи се параметри
    H(int = 1, int = 1);
    // конструктор за присвояване
    H(H const &);
    // член-функция за извеждане на член-данните на класа
    void display() const;
};
H::H(int x, int y)

```

```

{ a = x;
  b = y;
}
H::H(H const &r)
{ a = r.a;
  b = r.b;
}
void H::display() const
{ cout << a << " " << b << endl;
}
int main()
{ // създаване на обектите e1, e2 и e3 чрез
  // конструктора с подразбиращи се параметри
  H e1, e2(6), e3(3, 8);
  // извеждане на обектите
  e1.display();
  e2.display();
  e3.display();
  // създаване на обект e4 чрез конструктора за присвояване
  H e4 = e3;
  // извеждане на обекта e4
  e4.display();
  return 0;
}

```

Решение: В програмата са дефинирани два конструктора: с два подразбиращи се параметра и конструктор за присвояване. Декларацията

```
H e1, e2(6), e3(3, 8);
```

инициализира отделената за обектите e1, e2 и e3 памет като прилага конструктора с подразбиращите се параметри. В резултат се получава:

e1.a	e1.b	e2.a	e2.b	e3.a	e3.b
1	1	6	1	3	8

Декларацията

```
H e4 = e3;
```

заделя памет за член-данните на обекта e4 и я инициализира, като използва конструктора за присвояване, със стойностите на член-данните на обекта e3, т.е.

e4.a	e4.b
3	8

Обръщенията към член-функцията display() извеждат на екрана стойностите на компонентите на дефинираните в главната функция обекти, т.е.

```

1 1
6 1
3 8

```

Задача 19. Да се напише програма, която дефинира клас Point3, определящ точка в пространството с целочислени координати. Класът да притежава:

- конструктор с параметри, определящи координатите на точката, като се подразбира точката (0, 0, 0);
- конструктор за присвояване;
- функции за достъп до координатите на точка;
- мутатори на координатите на точка.

Като се използват двата конструктора, в главната функция да се дефинират няколко точки и да се изведат координатите им. Като се използват мутаторите на координатите на точка да се променят координатите на някоя от дефинираните точки.

Да се промени конструкторът за присвояване като на координатите на неявния обект се присвоят координатите на указания като формален параметър обект, увеличени с 1, 2 и 3 съответно.

Решение:

```
#include <iostream.h>
class Point3
{ public:
    // конструктор с подразбиращи се параметри
    Point3(int = 0, int = 0, int = 0);
    // конструктор за присвояване
    Point3(const Point3 &);
    // член-функции за достъп до координатите на точка
    int getX() const;
    int getY() const;
    int getZ() const;
    // мутатори на координатите на точка
    void setX(int);
    void setY(int);
    void setZ(int);
private:
    int x;           // абсциса
    int y;           // ордината
    int z;           // апликата
};

Point3::Point3(int _x, int _y, int _z)
{ x = _x;
  y = _y;
  z = _z;
}

Point3::Point3(const Point3& p)
```

```

{ x = p.x;
  y = p.y;
  z = p.z;
}

int Point3::getX() const
{ return x;
}

int Point3::getY() const
{ return y;
}

int Point3::getZ() const
{ return z;
}

void Point3::setX(int _x)
{ x = _x;
}

void Point3::setY(int _y)
{ y = _y;
}

void Point3::setZ(int _z)
{ z = _z;
}

int main()
{ // създаване на точките P и P1 чрез
  // конструктора с подразбиращи се параметри
  Point3 P, P1(1, 1, 1);
  // извеждане на точките P и P1
  cout << "P = (" << P.getX() << "," << P.getY()
    << "," << P.getZ() << ")" << endl;
  cout << "P1 = (" << P1.getX() << "," << P1.getY()
    << "," << P1.getZ() << ")" << endl;
  // създаване на точката P2 чрез
  // конструктора за присвояване
  Point3 P2 = P1;
  // извеждане на точката P2
  cout << "P2 = (" << P2.getX() << "," << P2.getY() << "
    << P2.getZ() << ")" << endl;
  // промяна на координатите на точката P2
  P2.setX(5); P2.setY(10); P2.setZ(15);
  // извеждане на точката P2
  cout << "P2 = (" << P2.getX() << "," << P2.getY() << "
    << P2.getZ() << ")" << endl;
  return 0;
}

```

```
}
```

Конструкторът за присвояване, който на координатите на неявния обект присвоява координатите на указания като формален параметър обект, увеличени с 1, 2 и 3 съответно, има вида:

```
Point3::Point3(const Point3& p)
{ x = p.x + 1;
  y = p.y + 2;
  z = p.z + 3;
}
```

Извършете експерименти с този конструктор за присвояване.

Задача 20. Стек е редица от 0 или повече елементи, в която включването и изключването на елемент се осъществява в единия край на редицата, известен като връх на стека (top). Да се дефинира клас, който създава стек от символи.

Решение: В програмата по-долу стекът е представен чрез масива items, който може да има не повече от MaxStack елемента. Член-данната top е индекс, указващ елемента на върха на стека. В случай, че top е -1, стекът е празен. Ако стекът е празен и в него се включи символ, последният става стойност на items[0], а стойността на top става 0.

Програмата по-долу позволява на потребителя както да включва и изключва елемент в/от стек и да извежда на екрана елементите на стек, така и да създава копия на стек чрез конструктора за присвояване.

```
#include <iostream.h>
const int MaxStack = 100; // максимален брой елементи на стека
const int FullStack = MaxStack; // константа, индицираща
// пълен стек
const int EmptyStack = -1; // константа, индицираща
// празен стек
const char EmptyFlag = '\0'; // помощен флаг за празен стек
class Stack
{ private:
    char items[MaxStack]; // носител на стека
    int top; // указател към върха на стека
public:
    // конструктор по подразбиране,
    // създаващ празен стек
    Stack();
    // конструктор за присвояване
    Stack(const Stack&);
    // член-функция, която включва символа
```

```

// с в неявния стек
void push(char c);
// член-функция, която изключва елемент от
// неявния стек, ако последният не е празен
char pop();
// член-функция, която проверява дали
// неявният стек е празен
bool empty() const;
// член-функция, която проверява дали
// неявният стек е пълен
bool full() const;
// член-функция, която извежда елементите
// на неявния стек, като го разрушава
void print();
};

Stack::Stack()
{ top = EmptyStack;
}

Stack::Stack(const Stack& s)
{ top = s.top;
  for(int i = 0; i <= top; i++)
    items[i] = s.items[i];
}

void Stack::push(char c)
{ if(full())
  { cout << "Стекът е пълен!\n";
    return;
  }
  items[++top] = c;
}

char Stack::pop()
{ if(empty())
  { cout << "Стекът е празен!\n";
    return EmptyFlag;
  }
  return items[top--];
}

bool Stack::empty() const
{ return top == EmptyStack;
}

bool Stack::full() const
{ return top == FullStack - 1;
}

```

```

void Stack::print()
{ for(int i = top; i >= 0; i--)
    cout << pop() << endl;
}

int main()
{ // създаване на празен стек s
  Stack s;
  // включване на символите 'a', 'b' и 'c' в s
  s.push('a');
  s.push('b');
  s.push('c');
  // създаване на нов стек t, копие на стека s
  Stack t = s;
  // извеждане на елементите на стека s
  s.print();
  // изключване на елемент от стека t
  if(!t.empty())
    cout << t.pop() << endl;
  // извеждане на останалите елементи на стека t
  t.print();
  return 0;
}

```

Указатели към обекти на класове. Масиви и обекти

Задача 21. Да се намери резултатът от изпълнението на програмата:

```

#include <iostream.h>
class K
{ private:
  int a;
  double b;
public:
  K(int, double);
  K(K const &);
  void display() const;
};

K::K(int x, double y)
{ a = x;
  b = y;
}

K::K(K const &r)
{ a = r.a;
  b = r.b;
}

```

```

}
void K::display() const
{ cout << a << " " << b << endl;
}
int main()
{ K e1(4, 1.8), *e2 = &e1;
  e1.display();
  e2->display();
  K e3 = *e2;
  e3.display();
  return 0;
}

```

Решение: В програмата е дефиниран клас K, притежаващ два конструктора и член-функция, извеждаща член-данните на класа. В главната функция са дефинирани два обекта e1 и e3 и указател e2 към обекта e1. Обектът e1 е инициализиран от двупараметричния конструктор, указателят e2 сочи обекта e1, а обектът e3 е инициализиран от конструктора за присвояване с обекта e1, явяващ се съдържание на указателя e2. В главната функция са направени три обръщения към член-функцията display() – чрез обектите e1 и e3 и чрез указателя към обект на класа e2. Резултатът от изпълнението е:

```

4 1.8
4 1.8
4 1.8

```

Задача 22. Като се използва класът Point3, дефиниран в задача 19, да се дефинира функция, която:

- въвежда масив от точки в тримерното пространство;
- променя координатите на всяка точка от масива като ги заменя с координати, симетрични спрямо координатното начало;
- извежда масив от точки в тримерното пространство.

Решение:

```

a)
void Input(int n, Point3 a[])
{ for(int i = 0; i < n; i++)
  { int p;
    cout << "x= "; cin >> p;
    a[i].setX(p);
    cout << "y= "; cin >> p;
    a[i].setY(p);
    cout << "z= "; cin >> p;
    a[i].setZ(p);
  }
}

```

```

    }
}

б)
void Symmetry(int n, Point3 a[])
{
    for(int i = 0; i < n; i++)
    {
        a[i].setX(-a[i].getX());
        a[i].setY(-a[i].getY());
        a[i].setZ(-a[i].getZ());
    }
}

```

```

в)
void Output(int n, Point3 a[])
{
    for(int i = 0; i < n; i++)
    {
        cout << "Point3("
            << a[i].getX() << ", "
            << a[i].getY() << ", "
            << a[i].getZ() << ")\n";
    }
}

```

Задача 23. Да се дефинира функция, която сортира лексикографски във възходящ ред редица от точки в пространството. За целта да се използва класът Point3, определящ точка в пространството с целочислени координати.

Решение: Лексикографската наредба на точките $P1(x1, y1, z1)$ и $P2(x2, y2, z2)$ се задава чрез булевия израз $x1 < x2 \parallel x1 == x2 \ \&\& \ y1 < y2 \parallel x1 == x2 \ \&\& \ y1 == y2 \ \&\& \ z1 < z2$. За сортировката е използван методът "пряка селекция".

```

bool lexic(Point3 P1, Point3 P2)
{
    return P1.getX() < P2.getX() ||
        P1.getX() == P2.getX() && P1.getY() < P2.getY() ||
        P1.getX() == P2.getX() && P1.getY() == P2.getY() &&
        P1.getZ() < P2.getZ();
}

void Sort_Points(int n, Point3* P)
{
    for(int i = 0; i < n-1; i++)
    {
        int k = i;
        Point3 Min = P[i];
        for(int j = i+1; j < n; j++)
            if(lexic(P[j], Min))
                { Min = P[j];
                  k = j;
                }
        P[k] = P[i];
        P[i] = Min;
    }
}

```

```

    }
    P[k] = P[i]; P[i] = Min;
}
}

```

Задача 24. Дадена е редица от указатели към точки от пространството. Да се дефинира функция, която сортира лексикографски във възходящ ред редицата от точки. За целта да се използва класът Point3, определящ точка в пространството с целочислени координати.

Решение: Булевата функция lexic сравнява лексикографски във възходящ ред точките, сочени от зададените указатели.

```

bool lexic(Point3* P1, Point3* P2)
{ return P1->getX() < P2->getX() ||
    P1->getX() == P2->getX() &&
    P1->getY() < P2->getY() ||
    P1->getX() == P2->getX() &&
    P1->getY() == P2->getY() &&
    P1->getZ() < P2->getZ();
}

void Sort_Points(int n, Point3** P)
{ for(int i = 0; i < n-1; i++)
    { int k = i;
      Point3* Min = P[i];
      for(int j = i+1; j < n; j++)
          if(lexic(P[j], Min))
          { Min = P[j];
            k = j;
          }
      P[k] = P[i]; P[i] = Min;
    }
}

```

Задача 25. Лекар трябва да поддържа статистика за броя на посещенията, които са направили пациентите му. За всеки пациент лекарят записва името, датата на раждане и броя на посещенията, които е направил пациентът през една календарна година. Да се напише програма, която за лекар създава списък от пациентите му, намира средния брой на посещенията на пациентите, както и пациентите, направили повече от зададен брой посещения.

Решение: Класът Patient представя „пациент”. В него името на пациент (Name) е дефинирано чрез символен низ с максимална дължина MaxLengthName, датата на раждане (DOB) е обект на класа Date, а броят на посещенията (Visits) – цяло число. Освен конструктори, в класа Patient са дефинирани функции за достъп до член-данните и член-функция за извеждане. Класът Doctor определя „лекар”. В него са дефинирани: името на лекар (Name) – чрез симво-

ден низ с максимална дължина MaxLengthName, списък от пациенти (PatientsList), съдържащ регистрираните пациенти на лекар – чрез едномерен масив от обекти от клас Patient и брой на регистрираните пациенти (PatientsRegistered) – чрез цяло число. Освен конструктор в класа Doctor са дефинирани:

- член-функция за достъп до името на лекаря (getName);
- член-функция за регистриране на пациент (registerPatient);
- член-функция за намиране на средния брой на визити при лекаря (average);
- член-функция за извеждане на пациентите на лекар, направили по-голям брой визити от зададено количество (printUpper).

```
#include <iostream.h>
#include <string.h>
const int MaxLengthName = 30; // максимална дължина на име
const int PATIENTS = 200; // максимален брой пациенти
// дефиниция на клас Date
class Date
{ private:
    int theDay, // Ден
        theMonth, // Месец
        theYear; // Година
public:
    Date(int = 1, int = 1, int = 2000); // конструктор
    int Day() const; // достъп до ден
    int Month() const; // достъп до месец
    int Year() const; // достъп до година
    void setDay(int); // промяна на ден
    void setMonth(int); // промяна на месец
    void setYear(int); // промяна на година
    void print() const; // извеждане на дата
};
Date::Date(int d, int m, int y)
{ theDay = d;
  theMonth = m;
  theYear = y;
}
int Date::Day() const
{ return theDay;
}
int Date::Month() const
{ return theMonth;
}
int Date::Year() const
```

```

{ return theYear;
}
void Date::setDay(int d)
{ theDay = d;
}
void Date::setMonth(int m)
{ theMonth = m;
}
void Date::setYear(int y)
{ theYear = y;
}
void Date::print() const
{ cout << theDay << "." << theMonth
  << "." << theYear << endl;
}
// дефиниция на клас Patient
class Patient
{ public:
    Patient(); // конструктор по подразбиране
    Patient(char *, const Date&, int); // конструктор с
    // параметри
    const char* getName() const; // достъп до име
    Date getDOB() const; // достъп до дата на раждане
    int getVisits() const; // достъп до брой посещения
    void print() const; // извеждане на пациент
private:
    char Name[MaxLengthName]; // име на пациент
    Date DOB; // дата на раждане на пациент
    int Visits; // брой посещения на пациента
};

Patient::Patient()
{ Visits = 0;
}

Patient::Patient(char * n, const Date& d, int v)
{ strcpy(Name, n);
  DOB = d;
  Visits = v;
}

const char* Patient::getName() const
{ return Name;
}

Date Patient::getDOB() const
{ return DOB;
}

```

```

}
int Patient::getVisits() const
{ return Visits;
}
void Patient::print() const
{ cout << "Име: " << Name << "\n";
  cout << "Дата: "; DOB.print();
  cout << "Брой на визитите: " << Visits << endl;
}
// дефиниция на клас Doctor
class Doctor
{ public:
    Doctor(char*); // конструктор
    const char* getName() const; // достъп до име
    // регистриране на пациент
    void registerPatient(const Patient&);
    // намиране на средния брой на посещенията на пациентите
    double average() const;
    // извеждане на пациентите, направили по-голям брой
    // посещения от указания като параметър брой
    void printUpper(int) const;
private:
    char Name[MaxLengthName]; // име на лекар
    Patient PatientsList[PATIENTS]; // масив с пациентите
    // на лекаря
    int PatientsRegistered; // брой на регистрираните пациенти
};
Doctor::Doctor(char* n)
{ strcpy(Name, n);
  PatientsRegistered = 0;
}
const char* Doctor::getName() const
{ return Name;
}
void Doctor::registerPatient(const Patient& P)
{ PatientsList[PatientsRegistered] = P;
  PatientsRegistered++;
}
double Doctor::average() const
{ int totalVisits = 0;
  for(int i = 0; i < PatientsRegistered; i++)
    totalVisits += PatientsList[i].getVisits();
  if(PatientsRegistered != 0)

```

```

        return double(totalVisits)/PatientsRegistered;
    else return 0;
}

void Doctor::printUpper(int lim) const
{ cout << "Пациенти с посещения над лимита!\n";
  for(int i = 0; i < PatientsRegistered; i++)
    if(PatientsList[i].getVisits() > lim)
      PatientsList[i].print();
}

int main()
{ // дефиниране на лекар
  Doctor dl("Petrov");
  // въвеждане на броя на пациентите на лекаря
  int n;
  do
  { cout << "Брой на пациентите: ";
    cin >> n;
  } while(n < 0 || n > PATIENTS);
  // регистриране на пациентите
  for(int i = 0; i < n; i++)
  { cin.ignore(); // прескачане на знака за край на ред
    cout << "Име на пациент: ";
    char NamePatient[MaxLengthName];
    cin.getline(NamePatient, MaxLengthName+1);
    int d, m, y;
    cout << "ден: "; cin >> d;
    cout << "месец: "; cin >> m;
    cout << "година: "; cin >> y;
    Date dat(d, m, y);
    int vis;
    cout << "Брой визити на пациент:";
    cin >> vis;
    Patient P(NamePatient, dat, vis);
    dl.registerPatient(P);
  }
  // извеждане на средния брой посещения при лекаря
  cout << "Среден брой посещения при лекаря "
        << dl.average() << endl;
  // извеждане на пациентите, направили
  // повече от 3 посещения
  if(n > 0) dl.printUpper(3);
  return 0;
}

```

Динамични обекти. Деструктори. Създаване и разрушаване на обекти на класове

Задача 26. Да се намери резултатът от изпълнението на програмата:

```
#include <iostream.h>
class L
{ private:
    int a;
    double b;
public:
    L(int = 5, double = 2.5);
    void display() const;
};
L::L(int x, double y)
{ a = x;
  b = y;
}
void L::display() const
{ cout << a << " " << b << endl;
}
int main()
{ L *p;
  p = new L(8); // заделяне на памет за обект
  if(!p) // проверка дали е заделена памет
  { cout << "Грешка при заделяне на памет!\n";
    return 1;
  }
  p -> display(); // извеждане на обекта, сочен от p
  delete p; // освобождаване на паметта
  p = new L(1, 3.5); // заделяне на памет за нов обект
  if(!p) // проверка дали е заделена памет
  { cout << "Грешка при заделяне на памет!\n";
    return 1;
  }
  p -> display(); // извеждане на обекта, сочен от p
  delete p; // освобождаване на паметта
  return 0;
}
```

Решение: В главната функция е дефиниран указателят `p` към класа `L`. Чрез

`p = new L(8);`

се търси памет за обект на класа `L` в хийпа. Ако памет не е намерена се извежда съобщение за

грешка и изпълнението на main завършва. Ако памет е намерена, чрез конструктора на класа паметта се инициализира с 8 и подразбиращата се стойност 2.5 за втория параметър, а указателят p се свързва с адреса на тази памет. След това се извежда обектът, сочен от указателя p и динамичната памет, свързана с указателя p, се освобождава. Тези действия се повтарят. Единствената разлика са параметрите на конструктора на класа L. Резултатът от изпълнението е:

8 2.5

1 3.5

Забележете, че върнатата от оператора new стойност се проверява, преди да се използва. Тази проверка има смисъл ако компилаторът реализира new така, че да връща стойност NULL при неуспех.

Задача 27. Да се намери резултатът от изпълнението на програмата:

```
#include <iostream.h>
class M
{ private:
    int a;
    double b;
public:
    M(int = 5, double = 2.5);
    void set(int, double);
    void display() const;
};
// конструктор
M::M(int x, double y)
{ a = x;
  b = y;
}
// мутатор
void M::set(int x, double y)
{ a = x;
  b = y;
}
// член-функция за извеждане на обект на класа
void M::display() const
{ cout << a << " " << b << endl; }
int main()
{ M *p;
  // заделяне на памет за масив от 5 обекта
  p = new M[5];
  if(!p)
  { cout << "Грешка при заделяне на памет!\n";
```

```

    return 1;
}
int i;
// извеждане на масива, заделен динамично
for(i = 0; i < 5; i++)
    p[i].display();
// променяне на компонентите на обектите
// на масива, заделен динамично
for(i = 0; i < 5; i++)
    p[i].set(i, i+1.5);
// извеждане на масива
for(i = 0; i < 5; i++)
    p[i].display();
// освобождаване на заделената за масива памет
delete [] p;
return 0;
}

```

Решение: В главната функция е дефиниран указател p към класа M . Чрез

$p = \text{new } M[5];$

се търси памет за масив от 5 обекта от клас M в хийпа. Ако памет не е намерена се извежда съобщение за грешка и изпълнението на `main` завършва. Ако памет е намерена, указателят p се свързва с тази памет. Чрез конструктора по подразбиране обектите на масива, заделен динамично, се инициализират със стойностите 5 и 2.5. Операторът за цикъл извежда тези стойности. Следващият фрагмент променя компонентите на обектите на масива като ги свързва съответно със стойностите $(i, i+1.5)$ за $i = 0, 1, \dots, 4$. След извеждане на масива, заделен динамично, паметта, сочена от указателя p се освобождава. Резултатът от изпълнението на програмата е:

```

5 2.5
5 2.5
5 2.5
5 2.5
0 1.5
1 2.5
2 3.5
3 4.5
4 5.5

```

Задача 28. Да се напише програма, която заделя памет за масив от 10 указателя към точки в тримерното пространство с цели координати. Инициализира заделената памет и намира точките, към които сочат указателите на създадения масив, които са на най-малко и на най-голямо разстояние до координатното начало.

Решение: Външната функция `Distance` намира разстоянието на указаната като неин пара-

метър точка до координатното начало.

```
#include <iostream.h>
#include <math.h>
// включване на реализацията на класа Point3
double Distance(const Point3& p)
{ return sqrt((double) (p.getX() * p.getX() +
                        p.getY() * p.getY() +
                        p.getZ() * p.getZ()));
}

int main()
{ // заделяне на памет за масив от 10 указателя
  Point3 ** array = new Point3*[10];
  if(!array)
  { cout << "Грешка при заделяне на памет!\n";
    return 1;
  }
  int i;
  // създаване на масива от указатели
  for(i = 0; i < 10; i++)
  { int x, y, z;
    cout << "**array[" << i << "] (x, y, z) = ";
    cin >> x >> y >> z;
    array[i] = new Point3(x, y, z);
    if(!array[i])
    { cout << "Грешка при заделяне на памет!\n";
      return 1;
    }
  }
  // намиране на минималното и на максималното
  // разстояние на точките, сочени от указателите
  // на масива до координатното начало
  double min = Distance(*array[0]),
        max = Distance(*array[0]);
  for(i = 1; i < 10; i++)
  { double d = Distance(*array[i]);
    if(d < min) min = d;
    if(d > max) max = d;
  }
  // намиране и извеждане на точките на най-малко
  // разстояние от координатното начало
  cout << "Точки на най-малко разстояние (" << min
```

```

    << ") от координатното начало\n";
for(i = 0; i < 10; i++)
{ if(fabs(Distance(*array[i]) - min) < 1E-14)
    cout << "Point(" << array[i]->getX() << ", "
        << array[i]->getY() << ", "
        << array[i]->getZ() << ")\n";
}
// намиране и извеждане на точките на най-голямо
// разстояние от координатното начало
cout << "Точки на най-голямо разстояние (" << max
    << ") от координатното начало\n";
for(i = 0; i < 10; i++)
{ if(fabs(Distance(*array[i]) - max) < 1E-14)
    cout << "Point(" << array[i]->getX() << ", "
        << array[i]->getY() << ", "
        << array[i]->getZ() << ")\n";
}
// освобождаване на заделената в хийпа памет
for(i = 0; i < 10; i++)
    delete array[i];
delete [] array;
return 0;
}

```

Задача 29. Да се намери резултатът от изпълнението на програмата:

```

#include <iostream.h>
class N
{ private:
    int a;
    double b;
public:
    // конструктор
    N(int = 5, double = 2.5);
    // деструктор
    ~N();
    // член-функция за извеждане на обект
    void display() const;
};

N::N(int x, double y)
{ a = x;
  b = y;
}

N::~~N()

```

```

{ cout << "~N()\n";
}

void N::display() const
{ cout << a << " " << b << endl;
}

int main()
{ N *p = NULL;
  p = new N(8); // заделяне на памет за обект
  if(!p) // проверка дали памет е заделена
  { cout << "Грешка при заделяне на памет!\n";
    return 1;
  }
  p -> display(); // извеждане на обекта, сочен от указателя p
  delete p; // освобождаване на паметта
  p = new N(1, 3.5); // заделяне на памет за нов обект
  if(!p) // проверка дали памет е заделена
  { cout << "Грешка при заделяне на памет!\n";
    return 1;
  }
  p -> display(); // извеждане на обекта, сочен от указателя p
  delete p; // освобождаване на паметта
  return 0;
}

```

Решение: В класа N е дефиниран деструктор. При освобождаването на заделената динамична памет, свързана с указателя p, се извиква деструкторът. Резултатът от изпълнението на програмата е:

```

8 2.5
~N()
1 3.5
~N()

```

Задача 30. Да се намери резултатът от изпълнението на програмата:

```

#include <iostream.h>
class P
{ private:
  int a;
  double b;
public:
  P(int = 5, double = 2.5);
  ~P();
  void display() const;
};

```

```

P::P(int x, double y)
{ a = x;
  b = y;
}
P::~~P()
{ cout << "~P()\n";
}
void P::display() const
{ cout << a << " " << b << endl;
}
int main()
{ P *p;
  // заделяне на памет за масив от 5 обекта на клас P
  p = new P[5];
  if(!p)
  { cout << "Грешка при заделяне на памет!\n";
    return 1;
  }
  int i;
  // извеждане на масива, заделен динамично
  for(i = 0; i < 5; i++)
    p[i].display();
  // освобождаване на заделената за масива памет
  delete [] p;
  return 0;
}

```

Решение: В main() е дефиниран указател p към масив в хийпа, съдържащ 5 обекта на класа P. Обектите са инициализирани от конструктора с подразбиращи се параметри на класа. При освобождаването на сочената от p динамична памет, деструкторът на класа се извиква за всеки обект на масива. Резултатът от изпълнението на програмата е:

```

5 2.5
5 2.5
5 2.5
5 2.5
5 2.5
~P()
~P()
~P()
~P()
~P()

```

Задача 31. Да се дефинира клас BankAccount, определящ банкова сметка на клиент, състояща се от: име на клиент (символен низ), номер на банкова сметка (символен низ) и налична сума на клиент (реално число). Името на клиента и номерът на банковата му сметка да се реали-

зират в динамичната памет, като за всяка от тях се отдели толкова памет колкото е необходимо. Класът да притежава методи, чрез които може да:

- създава банкова сметка;
- извежда на екрана банкова сметка;
- внася пари в банкова сметка;
- тегли пари от банкова сметка.

Да се дефинира главна функция, която създава две банкови сметки, извежда информацията в сметките, внася сума в една от сметките и тегли сума от другата сметка.

Решение: Тъй като две от член-данните на класа са реализирани в динамичната памет, в класа е дефиниран деструктор.

```
#include <iostream.h>
#include <string.h>
#include <stdlib.h>
const int MAX = 100;           // максимална дължина на име и
                                // номер на банкова сметка

class BankAccount
{ private:
    char* name;                 // име на клиент
    char* accountnum;          // номер на банкова сметка
    double balance;            // баланс
public:
    BankAccount(const char* = "", const char* = "",
                 double = 0.0);
    BankAccount(const BankAccount&);
    ~BankAccount();
    void print() const;
    void put_in(double);
    void draw(double);
};

// конструктор с подразбиращи се параметри
BankAccount::BankAccount(const char* n, const char* an,
                          double b)
{ name = new char[strlen(n)+1];
  if(!name)
  { cout << "Грешка при заделяне на памет!\n";
    exit(1);
  }
  strcpy(name, n);
  accountnum = new char[strlen(an)+1];
  if(!accountnum)
  { cout << "Грешка при заделяне на памет!\n";
```

```

    exit(1);
}
strcpy(accountnum, an);
balance = b;
}
// конструктор за присвояване
BankAccount::BankAccount(const BankAccount& b)
{ name = new char[strlen(b.name)+1];
  if(!name)
  { cout << "Грешка при заделяне на памет!\n";
    exit(1);
  }
  strcpy(name, b.name);
  accountnum = new char[strlen(b.accountnum)+1];
  if(!accountnum)
  { cout << "Грешка при заделяне на памет!\n";
    exit(1);
  }
  strcpy(accountnum, b.accountnum);
  balance = b.balance;
}
// деструктор
BankAccount::~BankAccount()
{ delete [] accountnum;
  delete [] name;
}
// извеждане на сметка на клиент
void BankAccount::print() const
{ cout << "Име на клиент: " << name << endl;
  cout << "Номер на сметка: " << accountnum << endl;
  cout << "Баланс: " << balance << endl;
}
// внасяне на пари в банкова сметка
void BankAccount::put_in(double cash)
{ if(cash >= 0)
  balance = balance + cash;
  else
  cout << "Некоректен опит за внасяне на пари.\n";
}
// теглене на пари от банкова сметка
void BankAccount::draw(double cash)
{ if(cash < 0)
  cout << "Некоректен опит за теглене на пари.\n ";
}

```

```

else
    if(cash <= balance)
        balance = balance - cash;
    else
        cout << "Сметката не съдържа достатъчно пари.\n";
}
int main()
{
    cout << "Име на клиент: ";
    char n1[MAX]; cin.getline(n1, MAX);
    cout << "Номер на сметка: ";
    char an1[MAX]; cin.getline(an1, MAX);
    double b1;
    cout << "Баланс: "; cin >> b1;
    cin.ignore(); // прескачане на знака за край на ред
    // създаване на банкова сметка
    BankAccount x(n1, an1, b1);
    // извеждане на банковата сметка
    x.print();
    // внасяне на сума в банкова сметка
    // и извеждане на сметката
    x.put_in(200);
    x.print();
    cout << "Име на клиент: ";
    char n2[MAX]; cin.getline(n2, MAX);
    cout << "Номер на сметка: ";
    char an2[MAX]; cin.getline(an2, MAX);
    double b2;
    cout << "Баланс: "; cin >> b2;
    cin.ignore(); // прескачане на знака за край на ред
    // създаване на нова банкова сметка
    BankAccount y(n2, an2, b2);
    // извеждане на банковата сметка
    y.print();
    // теглене на сума от банковата сметка
    // и извеждане на сметката
    y.draw(250);
    y.print();
    return 0;
}

```

Задача 32. Да се дефинира клас `IntArray`, наподобяващ едномерен масив, но проверяващ границите на индексирания променлива. Класът да притежава и член-функции за търсене на елемент и за сортиране на едномерен масив.

Решение: Член-данните array и num на класа IntArray задават едномерен масив array с num на брой елемента. Последният е реализиран в динамичната памет. За това в класа е дефиниран и деструктор. В класа са включени две член-функции за сортиране елементите на масив и две член-функции за търсене на елемент: linearSearch и binarySearch. Член-функцията int linearSearch(int x) намира позицията на първото срещане на елемента x в неявния едномерен масив. Ако x не се среща в масива, linearSearch връща num. Член-функцията int binarySearch(int x) връща целочислена стойност i, дефинирана по следния начин:

$$i = \begin{cases} -1, & \text{ако } x < \text{array}[0] \\ 0, & \text{ако } x = \text{array}[0] \\ j, & \text{ако } \text{array}[j-1] < x \leq \text{array}[j] \\ \text{num}, & \text{ако } \text{array}[\text{num}-1] < x \end{cases}$$

```
#include <iostream.h>
#include <stdlib.h>
const int MAX_SIZE = 20;
class IntArray
{ public:
    IntArray(int);           // конструктор с параметър
    ~IntArray();            // деструктор
    IntArray(const IntArray&); // конструктор за присвояване
    bool set(int, int);      // присвоява стойност на елемент
    bool get(int, int&);     // намира стойност на елемент
    int linearSearch(int);   // реализира линейно търсене
    int binarySearch(int);   // реализира двоично търсене
    void selectionSort();    // реализира сортиране по метода
                             // "пряка селекция"
    void bubbleSort();       // реализира сортиране по метода
                             // на мехурчето

private:
    int* array;              // указател към масива
    int num;                 // брой на елементите
    bool isValid(int);       // проверява валидността на
                             // индекса на масива
};

// конструктор
// Всеки елемент на масива е инициализиран с 0.
IntArray::IntArray(int size)
{ if(size > MAX_SIZE)
    { cout << size << " надхвърля " << MAX_SIZE
      << ". \n";
      exit(1);
    }
}
```

```

if(size <= 0)
{ cout << "Некоректна размерност!\n";
  exit(1);
}
array = new int[size];
if(!array)
{ cout << "Грешка при заделяне на памет!\n";
  exit(1);
}
num = size;
for(int i = 0; i < size; i++)
  array[i] = 0;
}
// деструктор
// Освобождава паметта, заета от масива.
IntArray::~IntArray()
{ delete [] array;
}
// конструктор за присвояване
IntArray::IntArray(const IntArray& p)
{ array = new int[p.num];
  if(!array)
  { cout << "Грешка при заделяне на памет!\n";
    exit(1);
  }
  for(int i = 0; i < p.num; i++)
    array[i] = p.array[i];
  num = p.num;
}
// isValid връща true, ако аргументът ѝ е валиден
// индекс на масива и false - в противен случай.
bool IntArray::isValid(int elem)
{ return elem >= 0 && elem < num;
}
// set присвоява стойност на указан елемент на масива.
// Ако стойността се присвоява на валиден елемент, set
// връща true. В противен случай set връща false.
bool IntArray::set(int elem, int value)
{ if(isValid(elem))
  { array[elem] = value;
    return true;
  }
  return false;
}

```

```

}
// get намира стойността на указан от даден индекс елемент.
// Ако индексът е коректен, get връща true,
// в противен случай връща false.
bool IntArray::get(int elem, int& value)
{ if(isValid(elem))
    { value = array[elem];
      return true;
    }
    return false;
}
// linearSearch реализира метода на последователното търсене.
int IntArray::linearSearch(int x)
{ int i = 0;
  while(i < num-1 && x != array[i])
    i++;
  if(x == array[i]) return i;
  else return num;
}
// binarySearch реализира метода на двоичното търсене
// в сортиран във възходящ ред едномерен масив.
int IntArray::binarySearch(int x)
{ int middle, left = 0, right = num-1;
  if(x < array[left]) return -1;
  if(x == array[left]) return 0;
  if(x > array[right]) return num;
  while(right-left > 1)
  { middle = (left + right)/2;
    if(x <= array[middle]) right = middle;
    else left = middle;
  }
  return right;
}
// selectionSort реализира метода на пряката
// селекция за сортиране на едномерен масив
void IntArray::selectionSort()
{ int i, j, k;
  int min;
  for(i = 0; i < num-1; i++)
  { k = i; min = array[k];
    for(j = i+1; j < num; j++)
      if(array[j] < min)
        { k = j;

```

```

        min = array[k];
    }
    array[k] = array[i]; array[i] = min;
}

// bubbleSort реализира метода на мехурчето
// за сортиране на едномерен масив
void IntArray::bubbleSort()
{ int i, k, right = num-1;
  while(right > 0)
  { k = 0; int swap;
    for(i = 0; i < right; i++)
      if(array[i+1] < array[i])
      { swap = array[i];
        array[i] = array[i+1];
        array[i+1] = swap;
        k = i;
      }
    right = k;
  }
}

int main()
{ // създаване на едномерен масив
  int n;
  cout << "n= "; cin >> n;
  IntArray a(n);
  int val, i;
  for(i = 0; i < n; i++)
  { cout << "Елемент? "; cin >> val;
    if(!a.set(i, val))
      cout << "Грешка в инициализацията на елемент "
        << "на масива!\n";
  }
  cout << endl;
  // извеждане на едномерния масив
  for(i = 0; i < n; i++)
  { if(a.get(i, val)) cout << val << " ";
    else cout << "Грешка при намирането на "
      << "елемент от масива!\n";
  }
  cout << endl;
  // опит да се запомни елемент извън границите на масива
  if(a.set(n, MAX_SIZE+1))
    cout << "Елементът " << MAX_SIZE+1

```

```

        cout << " е запомнен в масива успешно.\n";
    else
        cout << "Грешка при запомнене на елемента " << MAX_SIZE+1
            << ".\n";
    // търсене на елемент в масива
    cout << "Елемент? "; cin >> val;
    if(a.get(a.linearSearch(val), val))
        cout << val << " се съдържа в масива.";
    else cout << val << " не се съдържа в масива.";
    // сортиране на масива по метода на мехурчето
    a.bubbleSort();
    cout << "Сортиран масив!\n";
    // извеждане на сортирания едномерен масив
    for(i = 0; i < n; i++)
        if(a.get(i, val)) cout << val << " ";
        else cout << "Грешка при получаване на елемент "
            << " на масива!\n";
    cout << endl;
    // търсене на мястото на даден елемент в
    // сортирания масив
    cout << "Елемент? "; cin >> val;
    cout << "Позицията на " << val << " в масива е: "
        << a.binarySearch(val) << endl;
    return 0;
}

```

Приятелски функции и класове

Задача 33. Да се дефинира клас Fraction, съдържащ две цели член-данни n и d ($d \neq 0$), конструктор и функции за достъп до компонентите. Да се дефинира приятелска функция IsFraction на класа, която определя дали n не се дели целочислено на d .

Решение: Тъй като външната функция IsFraction е приятелска на класа Fraction, тя има пряк достъп до член-данните n и d на класа Fraction. Този достъп тя осъществява чрез обекта obj, който е неин формален параметър.

```

#include <iostream.h>
class Fraction
{ private:
    int n, d;
public:
    // конструктор
    Fraction(int x, int y);

```

```

// функции за достъп до компонентите
int getN() const;
int getD() const;
// декларация на функция приятел на класа
friend bool IsFraction(const Fraction& obj);
};

Fraction::Fraction(int x, int y)
{ n = x;
  d = y;
}

int Fraction::getN() const
{ return n;
}

int Fraction::getD() const
{ return d;
}

// външна функция, приятел на класа Fraction
bool IsFraction(const Fraction& obj)
{ return obj.n % obj.d != 0;
}

int main()
{ // дефиниране на обект на класа Fraction
  int a, b;
  do
  { cout << "a, b (b различно от 0): ";
    cin >> a >> b;
  } while(b == 0);
  Fraction f(a, b);
  // проверка дали първото цяло число на обекта f1
  // не се дели на второто цяло число
  if(IsFraction(f))
    cout << f.getN() << " не се дели на "
    << f.getD() << endl;
  else
    cout << f.getN() << " се дели на "
    << f.getD() << endl;
  return 0;
}

```

Задача 34. Да се дефинират класове Cylinder и SquarePyramid, определящи геометричните фигури цилиндър и квадратна пирамида. Всеки от класовете да притежава конструктор и функции за достъп до член-данните му. Да се дефинира приятелска функция IsHigher на двата класа, която определя дали цилиндър е по-висок от пирамида. В главната функция да се