

Глава първа

Структури

Дефиниране и използване на структури. Указатели към структури.

Абстракция с данни

Дефиниране и използване на структури

Задача 1. За структурата Product (изделие), декларирана по следния начин:

```
struct Product
{ char description[32]; // описание на изделие
  int partNum; // номер на изделие
  double cost; // цена на изделие
};
```

да се извършат следните действия:

а) Да се създадат две изделия и се инициализират чрез следните данни:

<i>description</i>	<i>partNum</i>	<i>cost</i>
screw-driver	456	5.50
hammer	324	8.20

б) Да се изведат на екрана компонентите на двете изделия, дефинирани в а).

в) Да се дефинира масив от 10 структури Product. Да не се инициализира масивът.

г) Да се напише оператор за цикъл, който инициализира масива, дефиниран във в) чрез нулевите за съответните типове на полетата на Product стойности.

д) Да се дефинира масив от 5 структури Product и се инициализира чрез стойностите:

<i>description</i>	<i>partNum</i>	<i>cost</i>
screw-driver	456	5.50
hammer	324	8.20
socket	458	5.75
plier	929	10.50
hand-saw	536	7.45

е) Да се изведе на екрана масивът, дефиниран в д).

Решение:

а)

```
Product p1 = {"screw-driver", 456, 5.50},
```

```
p2 = {"hammer", 324, 8.20};
```

```

б)
cout << setiosflags(ios::fixed) << setprecision(2);
cout << setw(35) << p1.description
    << setw(15) << p1.partNum
    << setw(20) << p1.cost << endl;
cout << setw(35) << p2.description
    << setw(15) << p2.partNum
    << setw(20) << p2.cost << endl;

```

```

в)
Product P[10];

г)
for(int i = 0; i < 10; i++)
{ strcpy(P[i].description, "");
  P[i].partNum = 0;
  P[i].cost = 0.0;
}

```

```

д)
Product Q[5] = {{"screw-driver", 456, 5.50},
                {"hammer", 324, 8.20},
                {"socket", 458, 5.75},
                {"plier", 929, 10.50},
                {"hand-saw", 536, 7.45}};

```

```

е)
for(i = 0; i < 5; i++)
    cout << setw(35) << Q[i].description
        << setw(15) << Q[i].partNum
        << setw(20) << Q[i].cost << endl;

```

Задача 2. За структурата Rectangle, декларирана по следния начин:

```

struct Rectangle
{ int length;           // дължина
  int width;            // ширина
};

```

да се дефинира:

- функция void print_rectangle(Rectangle r), която извежда на екрана полетата на правоъгълника r;
- функция void init_rectangle(Rectangle& r), която инициализира полетата на правоъгълника r чрез стойности, въведени от клавиатурата;
- функция Rectangle init_rectangle(), която създава правоъгълник.

Решение:

a)

```
void print_rectangle(Rectangle r)
{ cout << setw(10) << r.length;
  << setw(10) << r.width << endl;
}
```

б)

```
void init_rectangle(Rectangle& r)
{ do
  { cout << "length: "; cin >> r.length;
    cout << "width: "; cin >> r.width;
  } while(r.length <= 0 || r.width <= 0);
}
```

в)

```
Rectangle init_rectangle()
{ Rectangle r;
  do
  { cout << "length: "; cin >> r.length;
    cout << "width: "; cin >> r.width;
  } while(r.length <= 0 || r.width <= 0);
  return r;
}
```

Задача 3. Да се дефинира структура, която задава триъгълник чрез трите му страни. Да се дефинират функции, които създават и извеждат триъгълник, намират периметъра и лицето му. Да се напише програма, която създава триъгълник по зададени страни, извежда триъгълника и намира и извежда периметъра и лицето му.

Решение:

```
#include <iostream.h>
#include <math.h>
// декларация на структурата triangle
struct triangle
{ double a, b, c; // страни на триъгълник
};
// създаване на триъгълник
void create_triangle(triangle& t)
{ bool tri;
  do
  { cout << "a, b, c (страни на триъгълник): ";
    cin >> t.a >> t.b >> t.c;
```

```

    tri = (t.a > 0) && (t.b > 0) && (t.c > 0) &&
        (t.a + t.b > t.c) && (t.a + t.c > t.b) &&
        (t.b + t.c > t.a);
    } while(!tri);
}
// извеждане на триъгълник
void show_triangle(const triangle &t)
{ cout << "triangle(" << t.a << ", " << t.b << ", "
    << t.c << ")\n";
}
// намиране на периметъра на триъгълник
double perimeter(const triangle &t)
{ return t.a + t.b + t.c;
}
// намиране на лицето на триъгълник
double surface(const triangle &t)
{ double p = perimeter(t)/2;
    return sqrt(p*(p - t.a)*(p - t.b)*(p - t.c));
}
int main()
{ // създаване на триъгълник
    triangle t; create_triangle(t);
    // извеждане на триъгълника
    show_triangle(t);
    // намиране и извеждане на периметъра на триъгълника
    cout << "Периметър: " << perimeter(t) << endl;
    // намиране и извеждане на лицето на триъгълника
    cout << "Лице: " << surface(t) << endl;
    return 0;
}

```

Функцията create_triangle може да бъде дефинирана и по следния начин:

```

triangle create_triangle()
{ triangle t;
    bool tri;
    do
    { cout << "a, b, c (страни на триъгълник): ";
        cin >> t.a >> t.b >> t.c;
        tri = (t.a > 0) && (t.b > 0) && (t.c > 0) &&
            (t.a + t.b > t.c) && (t.a + t.c > t.b) &&
            (t.b + t.c > t.a);
    } while(!tri);
}

```

```
return t;
}
```

Тогава създаването на триъгълник се осъществява чрез дефиницията

```
triangle t = create_triangle();
```

Задача 4. Да се дефинират структурите `polar` и `rect`, задаващи вектор с полярни и с правоъгълни координати съответно. Да се дефинират функции, които преобразуват вектор, зададен чрез правоъгълни, в полярни координати и обратно, както и функции, които извеждат вектор, зададен чрез полярните си и чрез правоъгълните си координати. В главната функция да се избере режим на въвеждане: `г` – за въвеждане в правоъгълни и `р` – в полярни координати. За всеки избран режим да се въведат произволен брой вектори, да се преобразуват в другия режим и да се изведат.

Решение:

```
#include <iostream.h>
#include <math.h>
const double PI = 3.14159265;
// декларация на структурата polar
struct polar
{ double mag; // радиус
  double ang; // ъгъл
};
// декларация на структурата rect
struct rect
{ double x; // абсциса
  double y; // ордината
};
// преобразуване на вектор, зададен чрез
// правоъгълни, в полярни координати
polar rect_to_polar(const rect& r)
{ polar p;
  p.mag = sqrt(r.x * r.x + r.y * r.y);
  p.ang = atan2(r.y, r.x);
  return p;
}
// преобразуване на вектор, зададен чрез
// полярни, в правоъгълни координати
rect polar_to_rect(const polar& p)
{ const double DEG_TO_RAD = PI/180;
  rect r;
  r.x = p.mag * cos(p.ang*DEG_TO_RAD);
  r.y = p.mag * sin(p.ang*DEG_TO_RAD);
  return r;
}
```

```

// извеждане на вектор, зададен чрез
// полярни координати
void show_polar(const polar& p)
{ const double RAD_TO_DEG = 180/PI;
  cout << "радиус = " << p.mag;
  cout << ", ъгъл = " << p.ang * RAD_TO_DEG;
  cout << " градуса.\n";
}
// извеждане на вектор, зададен чрез
// правоъгълни координати
void show_rect(const rect& r)
{ cout << "абсциса = " << r.x
  << ", ордината = " << r.y << "\n";
}
int main()
{ rect r;
  polar p;
  cout << "Изберете режим на въвеждане на координатите: \n";
  cout << "r - за правоъгълни и "
    << "p - за полярни координати. \n";
  char ch; cin >> ch;
  switch(ch)
  { case 'r':
    cout << "Въведете правоъгълните координати x и y: ";
    while(cin >> r.x >> r.y)
    { p = rect_to_polar(r);
      show_polar(p);
      cout << "Следващи правоъгълни координати, "
        << "за край въведете низа end ";
    }
    break;
  case 'p':
    cout << "Въведете полярните координати mag и ang: ";
    while(cin >> p.mag >> p.ang)
    { r = polar_to_rect(p);
      show_rect(r);
      cout << "Следващи полярни координати, "
        << "за край въведете низа end ";
    }
  }
  return 0;
}

```

Завършването на изпълнението на операторите за цикъл в главната функция се осъществява чрез въвеждане на символния низ „end”. Операторите за вход в условията на операторите while очакват въвеждане на реални числа. Въвеждането на низа „end”, както и на който и да е друг символен низ, различен от число, ще доведе потока cin до състояние fail, което ще завърши изпълнението на операторите за цикъл.

Задача 5. Да се дефинират структурите: Person, определяща лице по собствено име и фамилия и Client, определяща клиент като лице, притежавашо банкова сметка с дадена сума. Да се дефинират функции, които въвеждат и извеждат данни за лице и клиент. Да се напише програма, която:

а) въвежда имената и банковите сметки на множество от клиенти, зададено чрез едномерен масив;

б) извежда имената и банковите сметки на клиентите от множеството;

в) намира сумата от задълженията на клиентите от множеството.

Решение:

```
#include <iostream.h>
const int SIZE = 16;
const int SIZE_MAX = 20;
// декларация на структурата Person
struct Person
{ char f_name[SIZE]; // собствено име
  char l_name[SIZE]; // фамилно име
};
// въвеждане на данни за лице
void read_person(Person& P)
{ cout << "Собствено име: "; cin >> P.f_name;
  cout << "Фамилно име: "; cin >> P.l_name;
}
// извеждане на данните за лице
void print_person(const Person& P)
{ cout << "Собствено име: " << P.f_name << endl;
  << "Фамилно име: " << P.l_name << endl;
}
// декларация на структурата Client
struct Client
{ Person name; // лице
  double amount; // сума в банкова сметка
};
// въвеждане на данни за клиент
void read_client(Client& C)
{ cout << "Име на клиент: \n"; read_person(C.name);
```

```

    cout << "Сума на клиента: "; cin >> C.amount;
}
// извеждане на данни за клиент
void print_client(const Client& C)
{ cout << "Име на клиент:\n";
  print_person(C.name);
  cout << "Сума на клиента: " << C.amount << endl;
}
int main()
{ // а)
  // въвеждане на имената и банковите
  // сметки на множество от клиенти
  cout << "Въведете стойност за n от 1 до "
        << SIZE_MAX << ": ";
  int n; cin >> n;
  Client a[SIZE_MAX];
  int i;
  for(i = 0; i < n; i++)
    read_client(a[i]);
  // б)
  // извеждане на имената и банковите
  // сметки на клиентите от множеството
  cout << "Клиенти на банката: \n";
  for(i = 0; i < n; i++)
    print_client(a[i]);
  // в)
  // намиране на сумата от задълженията
  // на клиентите от множеството
  double S = 0;
  for(i = 0; i < n; i++)
    if (a[i].amount < 0) S += a[i].amount;
  cout << "Сумата от задълженията на клиентите е: "
        << S << endl;
  return 0;
}

```

Задача 6. Да се дефинира структура Triangle, която определя триъгълник, разположен в равнина, чрез двумерните декартови координати на върховете му. Да се дефинират функции, които създават и извеждат триъгълник, намират периметъра и лицето му. Да се напише програма, която демонстрира дефинираните функции.

Решение: Полетата (компонентите) на структурата Triangle са точки, дефинирани чрез структурата Point.

```

#include <iostream.h>
#include <math.h>
// декларация на структурата Point
struct Point
{ double x, // абсциса на точка
  y; // ордината на точка
};
// създаване на точка от равнината
void create_point(Point& t)
{ cout << "абсциса: "; cin >> t.x;
  cout << "ордината: "; cin >> t.y;
}
// извеждане на точка от равнината
void show_point(const Point& p)
{ cout << "Point(" << p.x << ", " << p.y << ")";
}
// намиране на дължината на отсечка с дадени краища
double segment(const Point& p1, const Point& p2)
{ return sqrt((p2.x - p1.x)*(p2.x - p1.x) +
  (p2.y - p1.y)*(p2.y - p1.y));
}
// декларация на структурата Triangle
struct Triangle
{ Point A, B, C; // върхове на триъгълника
};
// създаване на триъгълник
void create_triangle(Triangle& t)
{ cout << "Въведете три точки в равнината, които "
  << "могат да са върхове на триъгълник!\n";
  create_point(t.A);
  create_point(t.B);
  create_point(t.C);
}
// извеждане на триъгълник
void show_triangle(const Triangle& t)
{ cout << "triangle("; show_point(t.A);
  cout << ", "; show_point(t.B);
  cout << ", "; show_point(t.C);
  cout << ")\n";
}
// намиране на периметъра на триъгълник
double perimeter(const Triangle& t)

```

```

{ double a = segment(t.B, t.C),
  b = segment(t.A, t.C),
  c = segment(t.A, t.B);
  return a + b + c;
}
// намиране на лицето на триъгълник
double surface(const Triangle &t)
{ double a = segment(t.B, t.C),
  b = segment(t.A, t.C),
  c = segment(t.A, t.B);
  double p = (a+b+c)/2;
  return sqrt(p*(p-a)*(p-b)*(p-c));
}
int main()
{ // създаване на триъгълник
  Triangle t; create_triangle(t);
  // извеждане на триъгълника
  show_triangle(t);
  // намиране и извеждане на периметъра на триъгълника
  cout << "Периметър: " << perimeter(t) << endl;
  // намиране и извеждане на лицето на триъгълника
  cout << "Лице: " << surface(t) << endl;
  return 0;
}

```

Задача 7. Да се дефинира функция, която намира разстоянието между две точки в равнина. Като се използва тази функция, да се напише програма, която въвежда координатите на n точки от равнината ($2 \leq n \leq 100$), намира и извежда най-голямото разстояние между тях. За целта да се дефинира структура, определяща точка от равнината с декартови координати.

Решение:

```

#include <iostream.h>
#include <math.h>
const int SIZE_MAX = 100;
// декларация на структурата Point
struct Point
{ double x, y; // декартови координати на точка от равнината
};
// създаване на точка от равнината
void create_point(Point& t)
{ cout << "абсциса: "; cin >> t.x;
  cout << "ордината: "; cin >> t.y;
}

```

```

}
// извеждане на точка от равнината
void show_point(const Point& p)
{ cout << "Point(" << p.x << ", " << p.y << ")\n";
}
// намиране на разстоянието между две точки от равнината
double segment(const Point& p1, const Point& p2)
{ return sqrt((p2.x - p1.x)*(p2.x - p1.x) +
              (p2.y - p1.y)*(p2.y - p1.y));
}
// въвеждане на масив от точки
void read_points(int n, Point *P)
{ for(int i = 0; i < n; i++)
  { cout << "x, y= ";
    cin >> P[i].x >> P[i].y;
    // или create_point(P[i]);
  }
}
// извеждане на масив от точки
void show_points(int n, Point *P)
{ for(int i = 0; i < n; i++)
  { cout << "Point(" << P[i].x << ", " << P[i].y << ")\n";
    // или show_point(P[i]);
  }
}
// намиране на най-голямото разстояние между точките
double max_distance(int n, Point *P)
{ double max = 0;
  for(int i = 0; i < n-1; i++)
    for(int j = i+1; j < n; j++)
      { double temp = segment(P[i], P[j]);
        if (temp > max) max = temp;
      }
  return max;
}

int main()
{ // въвеждане на броя на точките
  int n;
  do
  { cout << "Въведете стойност на n - от 2 до "
    << SIZE_MAX << ": ";
    cin >> n;
  } while(n < 2 || n > SIZE_MAX);
  // деклариране на масив от точки в равнината

```

```

Point P[SIZE_MAX];
// въвеждане на точките на масива
read_points(n, P);
// извеждане на точките на масива
show_points(n, P);
// извеждане на най-голямото разстояние между точките
cout << "Максималното разстояние между точките е "
    << max_distance(n, P) << endl;
return 0;
}

```

Задача 8. Да се дефинира функция, която сортира лексикографски във възходящ ред редица от точки в равнината. За целта да се използва дефинираната в предишната задача структура Point, определяща точка от равнината с декартови координати.

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

```

void sort_points(int n, Point* P)
{ for(int i = 0; i < n-1; i++)
    { int k = i;
      Point Min = P[i];
      for(int j = i+1; j < n; j++)
          if(P[j].x < Min.x || P[j].x == Min.x && P[j].y < Min.y)
              { Min = P[j];
                k = j;
              }
      P[k] = P[i]; P[i] = Min;
    }
}

```

Указатели към структури

Задача 9. Структурата Rectangle е декларирана по следния начин:

```

struct Rectangle
{ int length;           // дължина
  int width;            // ширина
};

```

- Да се дефинира указател p към Rectangle и се инициализира с валидна стойност.
- Да се напише оператор, който извежда полетата на структурата, сочена от указателя.
- Ако Rectangle * ptr, кой от следните изрази е еквивалентен на ptr -> width:
 - *ptr.width;

- (*ptr).width;
- ptr.(*width)?

Решение:

a)

```
Rectangle r = {15, 20};
```

```
Rectangle *p = &r;
```

b)

```
cout << setw(10) << p->length
```

```
<< setw(10) << p->width << endl;
```

в)

ptr -> width е еквивалентен на втория поред израз.

Задача 10. Да се напише програма, която:

a) въвежда факултетните номера (цели положителни числа с максимум 5 цифри), фамилиите (символни низове с максимална дължина 11) и оценките по k ($1 \leq k \leq 10$) учебни дисциплини на студентите от една група;

b) извежда следната таблица:

Фак. ном.	Фамилия	Дисциплина1	ДисциплинаK	среден успех
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.

среден успех

...

среден успех

среден успех

в) сортира в низходящ ред по среден успех въведената информация за студентите от групата;

г) извежда сортираната информация за студентите от групата във вида на горната таблица, но без последните й два реда.

Решение: С цел ефективно реализиране на сортировката е дефиниран помощен масив от указатели към структурата Student, определяща студент. Първоначално i-тият елемент на масива от указатели е инициализиран да сочи i-тия елемент на масива от структури, съдържащ данните за студентите. Във функцията за сортиране при необходимост от размяна, тя се осъществява не със структурите, а с адресите на съответните им указатели.

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

```
#include <iomanip.h>
```

```

int k; // брой на учебните дисциплини
const int SIZE = 12; // максимална дължина на име
const int SIZE_MAX1 = 10; // максимален брой на
// учебните дисциплини
const int SIZE_MAX2 = 25; // максимален брой на
// студентите от групата
// декларация на структурата Student
struct Student
{
    int fac_n; // факултетен номер на студент
    char name[SIZE]; // фамилия на студент
    double subjects[SIZE_MAX1]; // масив с оценките на студент
};
// въвеждане на данни за студент
void create_student(Student& s)
{
    do
    {
        cout << "Факултетен номер: ";
        cin >> s.fac_n;
    } while(s.fac_n <= 0 || s.fac_n > 99999);
    cout << "Фамилия: "; cin >> s.name;
    for(int i = 0; i < k; i++)
    {
        cout << "Оценка по " << i+1 << "-и предмет: ";
        cin >> s.subjects[i];
    }
}
// извеждане на данните за студент
void print_student(const Student& s)
{
    cout << setw(5) << s.fac_n
        << setw(SIZE+2) << s.name;
    for(int i = 0; i < k; i++)
        cout << setw(5) << s.subjects[i];
}
// намиране на средния успех на студент
double result(const Student& s)
{
    double u = 0;
    for(int i = 0; i < k; i++)
        u += s.subjects[i];
    u /= k;
    return u;
}
// a)
// въвеждане на данни за студентите от една група
void create_group(int n, Student *s)
{
    for(int i = 0; i < n; i++)

```

```

    { cout << "Данни за " << i+1 << "-и студент:";
      create_student(s[i]);
    }
}
// б)
// извеждане на заглавието (антетката) на таблицата
void print_table()
{ cout << setiosflags(ios::fixed|ios::showpoint)
  << setprecision(2);
  int i;
  for(i = 0; i < 5+SIZE+2+5*(k+1)+1; i++)
    cout << "=";
  cout << endl;
  cout << setw(5) << "Ф.Н."
    << setw(SIZE+2) << "Фамилия";
  for(i = 0; i < k; i++)
    cout << setw(4) << "Д-" << i+1;
  cout << setw(6) << "Ср.у." << endl;
  for(i = 0; i < 5+SIZE+2+5*(k+1)+1; i++)
    cout << "=";
  cout << endl;
}
// извеждане на таблицата с данните
// за студентите от групата
void print_group(int n, Student *s)
{ // извеждане на заглавието на таблицата
  print_table();
  int i, j;
  // намиране на средния успех по всяка учебна дисциплина
  double temp[SIZE_MAX1] = {0, };
  for(j = 0; j < k; j++)
    { for(i = 0; i < n; i++)
      { temp[j] = temp[j] + s[i].subjects[j];
        temp[j] = temp[j]/n;
      }
    }
  // извеждане на данните за студентите и
  // намиране на сумата sum от средния
  // успех на всички студенти
  double sum = 0;
  for(i = 0; i < n; i++)
    { print_student(s[i]);
      double res = result(s[i]);
      sum += res;
    }
}

```

```

    cout << setw(6) << res << endl;
}
for(i = 0; i < 5+SIZE+2+5*(k+1)+1; i++)
    cout << "=";
// извеждане на средния успех на студентите
// от групата по всички предмети
cout << endl << setw(5+SIZE+2+5) << temp[0];
for(j = 1; j < k; j++)
    cout << setw(5) << temp[j];
cout << setw(6) << sum/n << endl;
}
// в)
// сортиране в низходящ ред по среден успех на
// масива от указатели към данните за студентите
// чрез метода на пряката селекция
void sortgroup(int n, Student **s)
{ for(int i = 0; i < n-1; i++)
    { int k = i;
      double max = result(*s[k]);
      for(int j = i+1; j < n; j++)
          { double d = result(*s[j]);
            if(d > max)
                { max = d;
                  k = j;
                }
            }
    }
    Student* x;
    x = s[i]; s[i] = s[k]; s[k] = x;
}
// г)
// извеждане на сортираните в низходящ ред по среден
// успех данни за студентите от групата
void print_sort_group(int n, Student **s)
{ // извеждане на заглавието на таблицата
  print_table();
  // извеждане на сортираните данни
  for(int i = 0; i < n; i++)
      { print_student(*s[i]);
        double res = result(*s[i]);
        cout << setw(6) << res << endl;
      }
}

```

```

int main()
{ // въвеждане на броя на учебните дисциплини
  do
  { cout << "Въведете стойност за k от 1 до "
    << SIZE_MAX1 << ": ";
    cin >> k;
  } while(k < 1 || k > SIZE_MAX1);
  // въвеждане на броя на студентите на групата
  int n;
  do
  { cout << "Въведете стойност за n от 1 до "
    << SIZE_MAX2 << ": ";
    cin >> n;
  } while(n < 1 || n > SIZE_MAX2);
  Student s[SIZE_MAX2]; // ще съдържа данни за студентите
  Student* ps[SIZE_MAX2]; // ще съдържа указатели към
  // елементите на масива s
  // създаване на масива s, съдържащ информация
  // за студентите от дадена група
  create_group(n, s);
  // извеждане на информацията за студентите от групата
  print_group(n, s);
  // инициализиране на елементите на масива от указатели
  for(int i = 0; i < n; i++)
    ps[i] = &s[i];
  // сортиране на информацията за студентите от групата
  sortgroup(n, ps);
  cout << endl;
  // извеждане на сортираната информация за студентите
  print_sort_group(n, ps);
  return 0;
}

```

Задача 11. Да се промени процедурата `sortgroup`, дефинирана в предходната задача, така че да сортира в низходящ ред по фамилия и среден успех информацията за студентите от дадена група.

Решение:

```

#include <string.h>
void sortgroup(int n, Student **s)
{ for(int i = 0; i < n-1; i++)
  { int k = i;
    Student* Max_Student = s[i];
    for(int j = i+1; j < n; j++)

```

```

    { int x = strcmp(s[j]->name, Max_Student->name);
      if(x > 0 || x == 0 &&
        result(*s[j]) > result(*Max_Student))
      { k = j;
        Max_Student = s[j];
      }
    }
    Student* x;
    x = s[i]; s[i] = s[k]; s[k] = x;
  }
}

```

Задача 12. Да се промени процедурата `sortgroup`, дефинирана в предходната задача, така че да сортира във възходящ ред по фамилия и в низходящ ред по среден успех информацията за студентите от дадена група.

Решение:

```

void sortgroup(int n, Student **s)
{ for(int i = 0; i < n-1; i++)
  { int k = i;
    Student* Max_Student = s[i];
    for(int j = i+1; j < n; j++)
    { int x = strcmp(s[j]->name, Max_Student->name);
      if(x < 0 || x == 0 &&
        result(*s[j]) > result(*Max_Student))
      { k = j;
        Max_Student = s[j];
      }
    }
    Student* x;
    x = s[i]; s[i] = s[k]; s[k] = x;
  }
}

```

Абстракция с данни

Задача 13. Да се дефинира абстрактен тип от данни (АТД), който определя триъгълник по трите му страни. АТД да притежава функции за:

- създаване на триъгълник;
- извеждане на триъгълник;
- намиране на периметъра и лицето на триъгълник;
- създаване на триъгълник със страни равни на страните на даден триъгълник, умножени с даден коефициент.

Да се напише програма, която създава редица от n триъгълника ($1 \leq n \leq 20$), определени от

страните им и намира и извежда триъгълника с минимално лице и триъгълника с максимален периметър. Програмата да конструира и редицата от триъгълниците, получена от зададената редица, като страните на всеки триъгълник са умножени с коефициент, равен на поредния номер на триъгълника в редицата (номерацията да започва от 1).

Решение: За дефинирането на АТД ще изпълним следните стъпки:

а) избор на представяне на данните

```
struct triangle
{ double a, b, c;          // страни на триъгълника
};
```

б) избор на примитивни операции

Следващите функции извършват действия над полетата на структурата.

създаване на триъгълник по зададени страни

```
void create(double x, double y, double z)
```

```
{ a = x;
  b = y;
  c = z;
}
```

функции за достъп

извеждане на триъгълник

```
void print() const
```

```
{ cout << "triangle" << a << ", " << b << ", "
    << c << "\n";
}
```

достъп до страните на триъгълника

```
double get_a() const
```

```
{ return a;
}
```

```
double get_b() const
```

```
{ return b;
}
```

```
double get_c() const
```

```
{ return c;
}
```

намиране на периметъра на триъгълник

```
double perimeter() const
```

```
{ return a + b + c;
}
```

намиране на лицето на триъгълник

```
double surface() const
```

```
{ double p = (a+b+c)/2;
```

```
return sqrt(p*(p-a)*(p-b)*(p-c));
```

в) избор на основни операции над триъгълници

Следващата функция извършва действия над полетата на структурата.

създаване на триъгълник със страни, равни на страните на даден триъгълник, умножени с коефициента k

```
triangle mult_k(int k)
{ triangle t;
  t.create(a * k, b * k, c * k);
  return t;
}
```

г) приложения в проблемна област (външни функции)

намиране на триъгълника от редица от триъгълници, който е с минимално лице

```
triangle MinSurface(int n, triangle T[])
```

```
{ int k = 0;
  double Min = T[0].surface();
  for(int i = 1; i < n; i++)
    if(T[i].surface() < Min)
    { Min = T[i].surface();
      k = i;
    }
  return T[k];
}
```

намиране на триъгълника от редица от триъгълници, който е с максимален периметър

```
triangle MaxPerimeter(int n, triangle T[])
```

```
{ int k = 0;
  double Max = T[0].perimeter();
  for(int i = 1; i < n; i++)
    if(T[i].perimeter() > Max)
    { Max = T[i].perimeter();
      k = i;
    }
  return T[k];
}
```

намиране на редица от триъгълници, получена от дадената редица, като страните на всеки триъгълник са умножени с коефициент, равен на поредния номер на триъгълника в редицата

```
void NewSequence(int n, triangle * T, triangle * NewT)
{ for(int i = 0; i < n; i++)
  NewT[i] = T[i].mult_k(i+1);
}
```

Следва и програмата, която съдържа структурата triangle, реализира пакетиране на данни-

те и операциите в единно цяло и информационно скриване на реализацията на типа данни.

```
#include <iostream.h>
#include <math.h>
const int MAX_SIZE = 20;
struct triangle
{ private:
    double a, b, c; // страни на триъгълника
public:
    // създаване на триъгълник по дадени страни
    void create(double x, double y, double z)
    { a = x;
      b = y;
      c = z;
    }
    // извеждане на триъгълник
    void print() const
    { cout << "triangle(" << a << ", " << b << ", "
      << c << ")\n";
    }
    // достъп до страните на триъгълник
    double get_a() const
    { return a;
    }
    double get_b() const
    { return b;
    }
    double get_c() const
    { return c;
    }
    // намиране на периметъра на триъгълник
    double perimeter() const
    { return a + b + c;
    }
    // намиране на лицето на триъгълник
    double surface() const
    { double p = (a+b+c)/2;
      return sqrt(p*(p-a)*(p-b)*(p-c));
    }
    // създаване на триъгълник със страни,
    // равни на страните на даден триъгълник,
    // умножени с коефициента k
    triangle mult_k(int k)
```

```

    { triangle t;
      t.create(a * k, b * k, c * k);
      return t;
    }
};
// намиране на триъгълника от дадена редица,
// който е с минимално лице
triangle MinSurface(int n, triangle T[])
{ int k = 0;
  double Min = T[0].surface();
  for(int i = 1; i < n; i++)
    if(T[i].surface() < Min)
      { Min = T[i].surface();
        k = i;
      }
  return T[k];
}
// намиране на триъгълника от дадена редица,
// който е с максимален периметър
triangle MaxPerimeter(int n, triangle T[])
{ int k = 0;
  double Max = T[0].perimeter();
  for(int i = 1; i < n; i++)
    if(T[i].perimeter() > Max)
      { Max = T[i].perimeter();
        k = i;
      }
  return T[k];
}
// намиране на редица от триъгълници, получена от
// дадена редица, като страните на всеки триъгълник
// са умножени с коефициент, равен на поредния
// номер на триъгълника в редицата
void NewSequence(int n, triangle * T, triangle * NewT)
{ for(int i = 0; i < n; i++)
  { NewT[i] = T[i].mult_k(i+1);
  }
}
int main()
{ int n;
  do
  { cout << "n= "; cin >> n;
  } while(n < 1 || n > MAX_SIZE);
  triangle T[MAX_SIZE];

```

```

int i;
for(i = 0; i < n; i++)
{ double x, y, z; // страни на триъгълник
  bool tri;
  // въвеждане на стойности на страните на триъгълник
  do
  { cout << "x, y, z (страни на триъгълник): ";
    cin >> x >> y >> z;
    tri = (x > 0) && (y > 0) && (z > 0) &&
          (x+y > z) && (x+z > y) && (y+z > x);
  } while(!tri);
  // създаване на i-я триъгълник
  T[i].create(x, y, z);
}
// извеждане на дефинираната редица от триъгълници
cout << "Редица от триъгълници:\n";
for(i = 0; i < n; i++)
  T[i].print();
// намиране и извеждане на триъгълника с най-малко лице
cout << "Триъгълник с най-малко лице:\n";
MinSurface(n, T).print();
// намиране и извеждане на триъгълника с най-голям периметър
cout << "Триъгълник с най-голям периметър:\n";
MaxPerimeter(n, T).print();
// намиране на редица от триъгълници, получена от
// дадената редица, като страните на всеки триъгълник
// са умножени с коефициент, равен на поредния
// номер на триъгълника в редицата
triangle NT[MAX_SIZE];
cout << "Нова редица от триъгълници:\n";
NewSequence(n, T, NT);
// извеждане на получената редица от триъгълници
for(i = 0; i < n; i++)
  NT[i].print();
return 0;
}

```

Задача 14. Като се използва следната реализация на АТД „рационално число“:

```

struct rat
{ private:
  int num; // числител
  int den; // знаменател
public:

```

```

void makerat(int, int); // създава рационално число
int numer()const; // намира числителя на рационално число
int denom()const; //намира знаменателя на рационално число
void printrat() const; // извежда рационално число
};
// намиране на най-големия общ делител
// на две естествени числа
int gcd(int a, int b)
{ while(a != b)
    if (a > b) a = a-b; else b = b-a;
    return a;
}
void rat::makerat(int a, int b)
{ // предполага се, че b е различно от 0
    if(a == 0)
    { num = 0;
      den = 1;
    }
    else
    { int g = gcd(abs(a), abs(b));
      if(a > 0 && b > 0 || a < 0 && b < 0)
      { num = abs(a)/g;
        den = abs(b)/g;
      }
      else
      { num = - abs(a)/g;
        den = abs(b)/g;
      }
    }
}
int rat::numer() const
{ return num;
}
int rat::denom() const
{ return den;
}
void rat::printrat() const
{ cout << num << "/" << den << endl;
}

```

да се дефинира:

a) булева функция equal(const rat& x, const rat& y), която установява дали рационалните числа x и y са равни.

б) булева функция `grthan(const rat& x, const rat& y)`, която установява дали рационалното число x е по-голямо от рационалното число y .

в) функция `maxrat(int n, rat x[])`, която намира най-голямото от рационалните числа на редицата от рационални числа $x_0, x_1, \dots, x_{n-1}$.

г) функция `sortrat(int n, rat x[])`, която сортира в низходящ ред по метода на пряката селекция елементите на редицата от рационални числа $x_0, x_1, \dots, x_{n-1}$.

Решение: Ще отбележим, че АТД „rat” е дефиниран при предположение, че знаменателят `den` е различно от 0 цяло число.

а)

```
bool equal(const rat& x, const rat& y)
{ return x.numer() * y.denom() ==
      x.denom() * y.numer();
}
```

б)

```
bool grthan(const rat& x, const rat& y)
{ return x.numer() * y.denom() >
      x.denom() * y.numer();
}
```

в)

```
rat maxrat(int n, rat x[])
{ rat max = x[0];
  for(int i = 1; i < n; i++)
    if(grthan(x[i], max))
      max = x[i];
  return max;
}
```

г)

```
void sortrat(int n, rat x[])
{ for(int i = 0; i < n-1; i++)
  { int k = i; rat max = x[i];
    for(int j = i+1; j < n; j++)
      if(grthan(x[j], max))
      { max = x[j];
        k = j;
      }
    x[k] = x[i]; x[i] = max;
  }
}
```

Задача 15. Като се използва АТД `rat` от предходната задача, да се напише програма, която

решава системата уравнения

$$\begin{cases} ax + by = e \\ cx + dy = f \end{cases}$$

където коефициентите a, b, c, d, e, f , а също и неизвестните x и y са дадени рационални числа.

Решение: В случай, че $ad - cb \neq 0$ системата е определена и решението ѝ е:

$$x = \frac{\begin{vmatrix} e & b \\ f & d \end{vmatrix}}{\begin{vmatrix} a & b \\ c & d \end{vmatrix}}; y = \frac{\begin{vmatrix} a & e \\ c & f \end{vmatrix}}{\begin{vmatrix} a & b \\ c & d \end{vmatrix}}.$$

Когато $ad - cb = 0$ и $bf - de = 0$ (т.е. $\frac{a}{c} = \frac{b}{d} = \frac{e}{f}$), системата има безброй много решения. А

когато $ad - cb = 0$ и $bf - de \neq 0$ (т.е. $\frac{a}{c} = \frac{b}{d} \neq \frac{e}{f}$), системата няма решение. За решението на системата са необходими функциите за рационално-числова аритметика и сравнение за равенство. Тези функции са външни за дефиницията на структурата `rat`.

```
#include <iostream.h>
#include <math.h>
// включване на реализацията на класа rat

// намиране на сумата на рационалните числа r1 и r2
rat sumrat(const rat& r1, const rat& r2)
{ rat r;
  r.makerat(r1.numer() * r2.denom() +
            r2.numer() * r1.denom(),
            r1.denom() * r2.denom());
  return r;
}

// намиране на разликата на рационалните числа r1 и r2
rat subrat(const rat& r1, const rat& r2)
{ rat r;
  r.makerat(r1.numer() * r2.denom() -
            r2.numer() * r1.denom(),
            r1.denom() * r2.denom());
  return r;
}

// намиране на произведението на рационалните числа r1 и r2
rat multrat(const rat& r1, const rat& r2)
{ rat r;
```

```

    r.makerat(r1.numer()*r2.numer(),
              r1.denom()*r2.denom());
    return r;
}
// намиране на частното на рационалните числа r1 и r2
rat quotrat(const rat& r1, const rat& r2)
{ rat r;
  r.makerat(r1.numer()*r2.denom(),
            r1.denom()*r2.numer());
  return r;
}
// проверка дали рационалните числа x и y са равни
bool equal(const rat& x, const rat& y)
{ return x.numer()*y.denom() ==
        x.denom()*y.numer();
}
// решаване на системата
void rat_system(const rat& a, const rat& b,
               const rat& c, const rat& d,
               const rat& e, const rat& f)
{ rat det = subrat(multrat(a, d), multrat(c, b)),
  zero;
  zero.makerat(0, 1);
  if(equal(det, zero))
    if(equal(multrat(b, f), multrat(d, e)))
      cout << "Системата има безброй много решения.\n";
    else cout << "Системата няма решение.\n";
  else
    { rat x, y;
      x = quotrat(subrat(multrat(e, d), multrat(b, f)), det);
      y = quotrat(subrat(multrat(a, f), multrat(c, e)), det);
      x.printrat();
      y.printrat();
    }
}
int main()
{ // дефиниране на коефициентите на системата
  rat a, b, c, d, e, f;
  // свързване на коефициентите на системата
  // със стойности
  a.makerat(1,2); b.makerat(-1,4);
  c.makerat(3,2); d.makerat(1,4);
  e.makerat(3,6); f.makerat(0,4);
}

```

```

// решаване на системата
rat_system(a, b, c, d, e, f);
return 0;
}

```

Задача 16. Нека $a_0, a_1, \dots, a_{n-1}, a_n$ и x са дадени рационални числа. Да се напише функция, която намира стойността на полинома $P(x) = a_0x^n + a_1x^{n-1} + \dots + a_{n-1}x + a_n$.

Решение:

```

#include <iostream.h>
#include <math.h>
// включване на реализацията на класа rat
// включване на функциите за рационално-числова аритметика

// намиране на стойността на полинома
rat Horner(int n, rat a[], const rat& x)
{
    rat s;
    s = a[0];
    for(int i = 1; i <= n; i++)
        s = sumrat(multrat(s, x), a[i]);
    return s;
}

int main()
{
    rat x, a[10];
    // създаване на рационалното число x
    int p, q;
    do
    {
        cout << "x_numer, x_denom? ";
        cin >> p >> q;
    } while(q == 0);
    x.makerat(p, q);
    // въвеждане на степента на полинома
    cout << "n= "; int n; cin >> n;
    // въвеждане на коефициентите на полинома
    for(int i = 0; i <= n; i++)
    {
        do
        {
            cout << "numer, denom? ";
            cin >> p >> q;
        } while(q == 0);
        a[i].makerat(p, q);
    }
    // намиране и извеждане на стойността на полинома
    Horner(n, a, x).printrat();
}

```

```

    return 0;
}

```

Задача 17. Да се дефинира АТД complex, който определя комплексно число и реализира аритметика с комплексни числа.

Решение:

а) избор на представяне на комплексно число

```

struct complex
{ double Re,           // реална част
  double Im;           // имагинерна част
};

```

б) примитивни операции

Следващите функции извършват действия над полетата на структурата.

създаване на комплексно число

```

void create(double x, double y)
{ Re = x;
  Im = y;
}

```

извеждане на комплексно число

```

void print() const
{ cout << Re << (Im < 0 ? "-" : "+") << Im << "i\n";
}

```

намиране на реалната част на комплексно число

```

double get_Re() const
{ return Re;
}

```

намиране на имагинерната част на комплексно число

```

double get_Im() const
{ return Im;
}

```

проверка дали комплексно число е равно на 0

```

bool IsZero() const
{ return (Re == 0) && (Im == 0);
}

```

в) модули, реализиращи основни операции над комплексни числа

Следващите функции извършват действия над полетата на структурата.

събиране на комплексни числа

```

complex sum(const complex& c) const
{ complex p;
  p.create(Re + c.Re, Im + c.Im);
  return p;
}

```

изваждане на комплексни числа

```

complex sub(const complex& c) const
{ complex p;
  p.create(Re - c.Re, Im - c.Im);
  return p;
}

```

умножение на комплексни числа

```

complex mult(const complex& c) const
{ complex p;
  p.create(Re*c.Re - Im*c.Im, Re*c.Im + Im*c.Re);
  return p;
}

```

деление на комплексни числа

```

complex quot(const complex& c) const
{ // предполага се, че комплексното
  // число c е различно от 0
  complex p;
  double den = c.Re*c.Re + c.Im*c.Im;
  p.create((Re*c.Re + Im*c.Im)/den,
           (Im*c.Re - Re*c.Im)/den);
  return p;
}

```

Следва програмата, която съдържа структурата complex, реализира пакетиране на данните и операциите в единно цяло и информационно скриване на реализацията на типа данни.

```

#include <iostream.h>
struct complex
{ private:
  double Re, Im; // реална и имагинерна части
public:
  // създаване на комплексно число
  void create(double x, double y)
  { Re = x;
    Im = y;
  }
  // извеждане на комплексно число
  void print() const
  { cout << Re << (Im < 0 ? "" : "+") << Im << "i\n";
  }
  // намиране на реалната част на комплексно число
  double get_Re() const
  { return Re;
  }
  // намиране на имагинерната част на комплексно число

```

```

double get_Im() const
{ return Im;
}
// проверка дали комплексно число е равно на 0
bool IsZero() const
{ return (Re == 0) && (Im == 0);
}
// събиране на комплексни числа
complex sum(const complex& c) const
{ complex p;
  p.create(Re + c.Re, Im + c.Im);
  return p;
}
// изваждане на комплексни числа
complex sub(const complex& c) const
{ complex p;
  p.create(Re - c.Re, Im - c.Im);
  return p;
}
// умножение на комплексни числа
complex mult(const complex& c) const
{ complex p;
  p.create(Re*c.Re - Im*c.Im, Re*c.Im + Im*c.Re);
  return p;
}
// деление на комплексни числа
// предполага се, че делителят c
// е различно от 0 комплексно число
complex quot(const complex& c) const
{ complex p;
  double den = c.Re*c.Re + c.Im*c.Im;
  p.create((Re*c.Re + Im*c.Im)/den,
           (Im*c.Re - Re*c.Im)/den);
  return p;
}
};

int main()
{ complex c1, c2;
  // създаване на комплексното число c1
  double a, b;
  cout << "Re, Im? "; cin >> a >> b;
  c1.create(a, b);
  // създаване на комплексното число c2

```

```

cout << "Re, Im? "; cin >> a >> b;
c2.create(a, b);
// намиране на c1+c2 и извеждане
c1.sum(c2).print();
// намиране на c1-c2 и извеждане
c1.sub(c2).print();
// намиране на c1*c2 и извеждане
c1.mult(c2).print();
// намиране на c1/c2 и извеждане
if(!c2.IsZero()) c1.quot(c2).print();
else cout << "Комплексното число c2 е равно на 0.";
return 0;
}

```

Задача 18. Да се напише програма, която решава системата уравнения

$$\begin{cases} ax + by = e \\ cx + dy = f \end{cases}$$

където коефициентите a, b, c, d, e, f , а също и неизвестните x и y са комплексни числа.

Решение: В случай, че $ad - cb \neq 0$ системата е определена и решението ѝ е:

$$x = \frac{\begin{vmatrix} e & b \\ f & d \end{vmatrix}}{\begin{vmatrix} a & b \\ c & d \end{vmatrix}}; y = \frac{\begin{vmatrix} a & e \\ c & f \end{vmatrix}}{\begin{vmatrix} a & b \\ c & d \end{vmatrix}}.$$

Когато $ad - cb = 0$ и $bf - de = 0$ (т.е. $\frac{a}{c} = \frac{b}{d} = \frac{e}{f}$), системата има безброй много решения. А когато $ad - cb = 0$ и $bf - de \neq 0$ (т.е. $\frac{a}{c} = \frac{b}{d} \neq \frac{e}{f}$), системата няма решение. Решението на системата използва булевата функция `equal`, която проверява дали две комплексни числа са равни.

```

#include <iostream.h>
// включване на реализацията на класа complex
// проверява дали две комплексни числа са равни
bool equal(const complex& x, const complex& y)
{ return x.get_Re() == y.get_Re() &&
      x.get_Im() == y.get_Im();
}
// решаване на системата
void complex_system( const complex& a, const complex& b,
                     const complex& c, const complex& d,
                     const complex& e, const complex& f )

```

```

{ complex det = a.mult(d).sub(c.mult(b)),
  zero;
  zero.create(0, 0);
  if(equal(det, zero))
    if(equal(b.mult(f), d.mult(e)))
      cout << "Системата има безброй много решения.\n";
    else cout << "Системата няма решение.\n";
  else
  { complex x, y;
    x = e.mult(d).sub(b.mult(f)).quot(det);
    y = a.mult(f).sub(c.mult(e)).quot(det);
    x.print();
    y.print();
  }
}

int main()
{ // дефиниране на коефициентите на системата
  complex a, b, c, d, e, f;
  // създаване на комплексните числа,
  // явяващи се коефициенти на системата
  a.create(1,2); b.create(-1,4);
  c.create(1,3); d.create(1,4);
  e.create(3,6); f.create(3,6);
  // решаване на системата
  complex_system(a, b, c, d, e, f);
  return 0;
}

```

Задача 19. Нека $a_0, a_1, \dots, a_n$, a_n и x са дадени комплексни числа. Да се дефинира функция, която намира стойността на полинома $P(x) = a_0 x^n + a_1 x^{n-1} + \dots + a_n$. Да се включи дефинираната функция в демонстрационна програма.

Решение:

```

#include <iostream.h>
// включване на реализацията на класа complex

// намиране на стойността на полинома
complex Horner(int n, complex a[], complex x)
{ complex s = a[0];
  for(int i = 1; i <= n; i++)
    s = s.mult(x).sum(a[i]);
  return s;
}

```

```

int main()
{ complex x, a[10];
  // създаване на комплексното число x
  double p, q;
  cout << "Re, Im? ";
  cin >> p >> q;
  x.create(p, q);
  // въвеждане на степента на полинома
  cout << "n= "; int n; cin >> n;
  // въвеждане на коефициентите на полинома
  for(int i = 0; i <= n; i++)
  { cout << "Re, Im? ";
    cin >> p >> q;
    a[i].create(p, q);
  }
  // намиране и извеждане на стойността на полинома
  Horner(n, a, x).print();
  return 0;
}

```

Задачи за самостоятелна работа

Дефиниране и използване на структури

Задача 1. Структурата Destination, декларирана по следния начин:

```
struct Destination
{ char city[32];           // град
  int kilometers;         // километри
};
```

определя дестинация чрез задаване на град и разстоянието му до град София.

а) да се създадат две дестинации и се инициализират със следните данни:

destination	kilometers
Plovdiv	165
Varna	469

б) да се изведат на екрана полетата на двете дестинации, дефинирани в а);

в) да се дефинира масив от 30 структури Destination. Да не се инициализира масивът.

г) да се напише оператор за цикъл, който инициализира масива, дефиниран във в) чрез нулевите за съответните типове на полетата на Destination стойности.

д) да се дефинира масив от 5 структури Destination и се инициализира чрез следните данни:

destination	kilometers
Plovdiv	165
Varna	469
Burgas	393
Vidin	199
Dobrich	512

е) да се изведе на екрана масивът, дефиниран в д).

Задача 2. Да се намерят, обяснят и поправят грешките в програмата:

```
#include <iostream.h>
#include <math.h>
// дефиниция на структурата triangle
struct triangle
{ double a, b, c;           // страни на триъгълник
};
// създаване на триъгълник
void create_triangle(const triangle& t)
{ bool tri;
  do
  { cout << "a, b, c (страни на триъгълник): ";
    cin >> t.a >> t.b >> t.c;
    tri = (t.a > 0) && (t.b > 0) && (t.c > 0) &&
```

```

        (t.a + t.b > t.c) && (t.a + t.c > t.b) &&
        (t.b + t.c > t.a);
    } while(!tri);
}
// проверка дали два триъгълника
// са със съответно равни страни
bool eq(const triangle& t1, const triangle& t2)
{ return t1 == t2;
}
int main()
{ // създаване на триъгълник
  triangle t1; create_triangle(t1);
  // създаване на триъгълник
  triangle t2; create_triangle(t2);
  // проверка дали двата триъгълника са
  // със съответно равни страни
  if(eq(t1, t2))
    cout << "Двата триъгълника са със съответно равни страни";
  else
    cout << "Двата триъгълника не са "
           "с със съответно равни страни";
  return 0;
}

```

Задача 3. Да се дефинира структура, която определя правоъгълник по двете му страни. Да се дефинират функции, които създават и извеждат правоъгълник, намират радиуса на описаната около правоъгълник окръжност, както и лицето и периметъра на правоъгълник. Да се напише програма, която създава правоъгълник по зададени страни. Извежда правоъгълника. Намира и извежда радиуса на описаната около правоъгълника окръжност, периметъра и лицето му.

Задача 4. Да се дефинира структура, определяща точка от равнината с координати (x, y), където x и y приемат за стойности числата от 1 до 100. Да се напише програма, която чете координатите на четири точки, представляващи върховете A, B, C и D на четириъгълник в цикличен ред и определя дали ABCD е квадрат, правоъгълник или друга фигура.

Задача 5. Да се дефинира структура Planet, определяща планета по име (символен низ), разстояние от слънцето, диаметър и маса (реални числа). Да се дефинира функция, която:

- въвежда данни за планета;
- извежда данните за планета;
- връща като резултат броя секунди, които са необходими на светлината да достигне от слънцето до планетата (да се приеме, че светлината има скорост 299 792 км/сек. и че разстоянието на планетата до слънцето е зададено в километри).

Да се напише програма, която:

- въвежда множество от планети, реализирано чрез едномерен масив;
- извежда данните за планетите от множеството;

- в) намира планетата с най-голям диаметър;
- г) намира планетата с най-малка маса;
- д) намира планетата, до която светлината достига последна (първа).

Задача 6. Структурата Fruit, определяща плод, е дефинирана по следния начин:

```
struct Fruit
{ char name[32];           // име на плод
  color cl;                // цвят на плод
  yn annual;               // едногодишен плод
  yn perennial;            // многогодишен плод
  yn tropical;             // тропически плод
};
```

където

```
enum color {brown, green, orange, red, yellow};
```

```
enum yn {yes, no};
```

Да се дефинира функция, която:

- а) създава едномерен масив от плодове;
- б) извежда едномерния масив от плодове;
- в) извежда едногодишните плодове с даден цвят, записани в масива, сортирани във възходящ ред по име на плода;
- г) извежда многогодишните, тропически плодове с даден цвят, сортирани в низходящ ред по име на плода.

Указатели към структури

Задача 7. Да се дефинира структура Planet, определяща планета по име (символен низ), разстояние от слънцето, диаметър и маса (реални числа). Да се напише програма, която:

- а) създава едномерен масив от указатели към структури, задаващи планети;
 - б) сортира в низходящ ред по разстояние до слънцето и диаметър планетите, зададени чрез масива;
 - в) извежда данните за планетите, зададени чрез масива, след сортирането им.
- За целта да се дефинират подходящи функции.

Задача 8. Структурата Fruit, определяща плод, е дефинирана по следния начин:

```
struct Fruit
{ char name[32];           // име на плод
  color cl;                // цвят на плод
  yn annual;               // едногодишен плод
  yn perennial;            // многогодишен плод
  yn tropical;             // тропически плод
};
```

където

```
enum color {brown, green, orange, red, yellow};
```

```
enum yn {yes, no};
```

Да се напише програма, която:

- а) създава едномерен масив от указатели към структури, задаващи плодове;
- б) сортира по име, цвят и продължителност на плододаване плодовете, зададени чрез масива;
- в) извежда данните за плодовете, зададени чрез масива, след сортирането им.

За целта да се дефинират подходящи функции.

Абстракция с данни

Задача 9. Да се дефинира АТД Vector, който определя вектор в тримерното пространство и притежава функции за:

- а) създаване на вектор;
- б) извеждане на вектор;
- в) намиране на сумата на два вектора;
- г) намиране на разликата на два вектора;
- д) намиране на произведението на вектор с число;
- е) намиране на скаларното произведение на два вектора;
- ж) намиране на векторното произведение на два вектора;
- з) намиране на смесеното произведение на три вектора.

Да се напише програма, която създава няколко вектора и намира и извежда сумата, разликата и различните произведения на векторите.

Задача 10. Като се използва АТД Vector (Задача 9), да се напише програма, която въвежда n вектора от тримерното пространство и проверява дали векторите са компланарни (лежат в една равнина или в успоредни равнини).

Задача 11. Като се използва АТД Vector (Задача 9), да се напише програма, която въвежда n вектора от тримерното пространство ($n \geq 3$) и проверява дали съществуват три вектора от тях, които са взаимно перпендикулярни.

Задача 12. Като се използва АТД Vector (Задача 9), да се напише програма, която въвежда n вектора от тримерното пространство ($n \geq 3$) и за всяка тройка некомпланарни вектори намира обема на призмата, определена от тях.

Задача 13. Дадени са естественото число n и комплексното число z . Като се използва реализацията на АТД „комплексно число“ да се напише програма, която пресмята стойността на следната комплексна функция:

$$F = 1 + \frac{z}{1!} + \frac{z^2}{2!} + \dots + \frac{z^n}{n!}$$

Задача 14. Модул или абсолютна стойност на комплексното число $z = a + i.b$ е реалното число $\sqrt{a^2 + b^2}$. Като се използва реализацията на АТД „комплексно число“ да се напише програма, която въвежда редица от n комплексни числа и проверява дали всички те имат еднакъв модул.