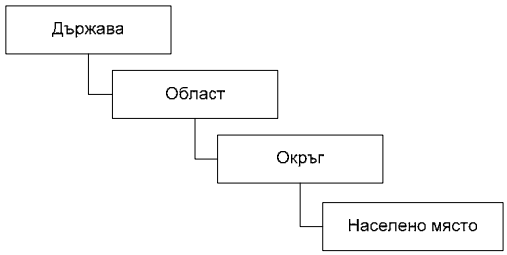
СОФИЙСКИ УНИВЕРСИТЕТ
"СВ. КЛИМЕНТ ОХРИДСКИ"

ФАКУЛТЕТ ПО МАТЕМАТИКА
И ИНФОРМАТИКА

ДЪРЖАВЕН ИЗПИТ
ЗА ПОЛУЧАВАНЕ НА ОКС "БАКАЛАВЪР ПО ИНФОРМАТИКА"
7 - 8.07.2007 г.

ЧАСТ I (ПРАКТИЧЕСКИ ЗАДАЧИ)

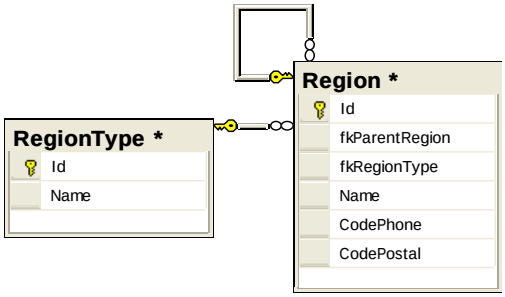
Задача 1. (5 т.) За съхраняване на регионалната структура на България е създадена база от данни **`Regions`**. Структурата на регион е йерархична, както е показано на фиг. 1:



фиг. 1

Релационната схема на базата от данни е:

`Region``(Id, fkParentRegion, fkRegionType, Name, CodePhone, CodePostal)`
`RegionType``(Id, Name)`



където:

* Таблица *Колона*	*Описание*	*Тип на данните*	*Ограничения*
*** Region**	Регион от административното деление		
Id	Генерира се автоматично	int	PK
fkParentRegion	Йерархичен родител - **Region.Id**	int	FK
fkRegionType	Тип на региона - **RegionType.Id**	int	FK
Name	Име	varchar(50)	
CodePhone	Телефонен код	varchar(10)	
CodePostal	Пощенски код	varchar(10)	
*** RegionType**	Тип на региона		
Id	Генерира се автоматично Уникално идентифицира типа на региона	int	PK
Name	Име на региона: - държава - област - окръг - населено място (град) - населено място (село) - населено място (махала)	varchar(20)	

Легенда	
PK	Първичен ключ (Primary Key)
FK	Вторичен ключ (Foreign Key)

Да се изведе информация за областта с най-много села. Справката да има вида:

Име на област	Брой села в областта

ЧЕРНОВА ЗА ЗАДАЧА 1

Задача 2. (6 т.) Дадено е множеството от двоични функции

$$\mathbf{A} = \{f(\tilde{x}^n), g(\tilde{x}^3) = \tilde{1} \downarrow (x_1 \vee (x_1 \oplus x_2) \vee x_3), h(\tilde{x}^n) = x_1 \oplus x_2 \oplus \ldots \oplus x_n\}$$

където $f(\tilde{x}^n), n \geq 2$, е двоична функция, такава че $f(x_1, x_2, 0, \ldots, 0) = x_1 \rightarrow x_2$

a) Да се изследва функцията $g(\tilde{x}^3)$ за принадлежност към всяко от множествата T_0, T_1, **S**, **M** и **L**;

b) Да се определи за кои стойности на **n≥2** функцията $h(\tilde{x}^n)$ принадлежи на всяко от множествата T_0, T_1, **S**, **M** и **L** и за кои - не принадлежи;

c) Да се докаже, че множеството **A** е пълно за всяко **n≥2.**

Задача 3. (6 т.) Даден е крайният автомат $\mathbf{A} = \langle \{q_0, q_1, q_2, q_3, q_4, q_5, q_6\}, \{0,1\}, q_0, \delta, \{q_6\} \rangle$

δ :

q	0	1
q_0	$\{q_1, q_3, q_6\}$	$\{q_4, q_6\}$
q_1	$\varnothing$	$\{q_2\}$
q_2	$\{q_1, q_6\}$	$\{q_2\}$
q_3	$\{q_5, q_6\}$	$\varnothing$
q_4	$\{q_3\}$	$\{q_4, q_6\}$
q_5	$\{q_3\}$	$\{q_4, q_6\}$
q_6	$\varnothing$	$\varnothing$

a) Да се построи детерминиран краен автомат **B**, еквивалентен на автомата **A**;

b) Да се построи минимален детерминиран краен автомат **C**, еквивалентен на автомата **B**.

ЧЕРНОВА ЗА ЗАДАЧИ 2 и 3

Задача 4. (5 т.) В текущия каталог се намира текстов файл **`fileC.txt`** със следното съдържание:

```
0123456789
abcdefg
xxxxxxxxxx
```

Напишете какво ще бъде изведено на стандартния изход (терминала) като резултат от изпълнението на файла, получен при успешна компилация на зададения по-долу програмен код на C, в който са използвани системни примитиви на ОС UNIX и LINUX:

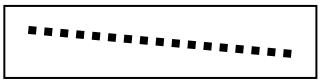
```
#include  <stdio.h>
#include  <fcntl.h>
main( )
{
     int fdi, n_byt, pid,  i = 0 ;
     char sline [ 40 ], c ;

     if ( ( pid = fork( ) ) == 0 )
     {
          if  (  ( fdi = open ("fileC.txt", O_RDONLY ) ) == -1 )
          { printf ("\n Cannot open  \n" ); exit (1); }
          n_byt = read ( fdi, sline, 40 );
          c = sline[ i++];
          if  ( c <= '0' || c >= '9' )
          {    while  ( sline [ i ] != '\n' && i < 40 )
                    write ( 1, &sline [ i++ ], 1 );
               write ( 1, "\n", 1 );
          }
          close ( fdi );
     }
     else
     {  wait ( );
        if  (  ( fdi = open ("fileC.txt", O_RDONLY) ) == -1 )
          { printf ("\n Cannot open  \n" ); exit (1); }
        else execlp ("wc", "wc", "-l","fileC.txt",0 );
     }
}
```

Решение:

ЧЕРНОВА ЗА ЗАДАЧА 4

Задача 5. (4 т.) Разполагаме с функции **`PutPixel(X,Y,Value)`** и **`HLine(X1,X2,Y,Value)`**, изчертаващи съответно точка и хоризонтална линия с даден цвят. Да се даде псевдокод от тип на Брезенхам за запълване на кръг с център **`(Xc,Yc)`** и радиус **`R`**. Да се оптимизира алгоритъма, така че всеки пиксел да се изчертава по веднъж.

Задача 6. (3 т.) Напишете (на езика C или чрез псевдокод) модифициран алгоритъм на Брезенхам или на средната точка за изчертаване на отсечка с пунктирна линия (през пиксел):

Пример: 

Считайте, че процедурата **`PutPixel(x,y)`** променя пиксела с координати **`x`** и **`y`** в желания цвят.

ЧЕРНОВА ЗА ЗАДАЧИ 5 и 6

Задача 7. (4 т.) Нека **P** е следната логическа програма:

```
p( c ).
p( f( f( X ))) :-  p( X ).
```

Определете минималния Ербранов модел $\mathbf{M_P}$ на програмата **P**.

Задача 8. (4 т.) Дадена е следната рекурсивна програма **R** над *целите* числа:

```
R:  F( X, Y, 0)   where
      F( X, Y, S) = if  Y = 0  then  S  else  F( X, Y-1,  G( X, S))
      G( X, Y) = if  X = 0  then  Y  else G( X+1, Y-1).
```

Кои от изброените условия са верни за $D_V(R)$ (предполагаме, че **x** и **y** пробягват множеството на целите числа):

a) $\forall x \, \forall y \; ((x \geq 0 \;\&\; y \geq 0) \Rightarrow D_V(R)(x, y) \cong x \,.\, y)$;
b) $\forall x \, \forall y \; ((x \geq 0 \;\&\; y \leq 0) \Rightarrow D_V(R)(x, y) \cong x \,.\, y)$;
c) $\forall x \, \forall y \; ((x \leq 0 \;\&\; y \geq 0) \Rightarrow D_V(R)(x, y) \cong x \,.\, y)$;
d) нито едно от горните три.

ЧЕРНОВА ЗА ЗАДАЧИ 7 и 8

Задача 9. (4 т.) Оценете изразите:

a)
```
(map atom? '(12.34 'a "a" #f "(1 2 3)" '() a 0 '(a)))
```
.....

.....

b)
```
((lambda (x) (x 5)) (lambda (y) (/ 15 y)))
```
.....

.....

c)
```
(let* ((x (list (lambda (x) (* x 2))))
       (x (cons (lambda (x) (+ 5 x)) (cons ((car x) 1) x))))
      ((caddr x) 3))
```
.....

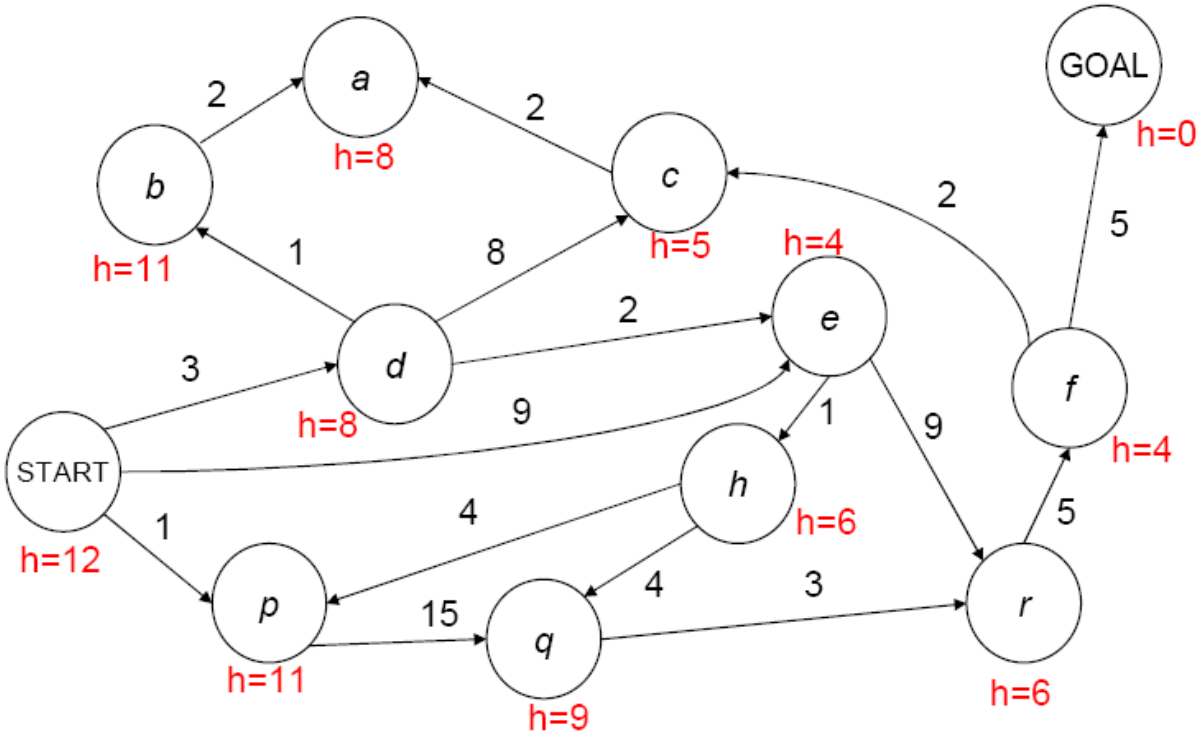
.....

d)
```
(cadr (list '(1 (2 (3))) (caar '(((4 5) 6) (7) ((8))))))
```
.....

.....

Задача 10. (3 т.) Даден е списък **Nat** от естествени числа. Като използвате процедурите **accumulate, map** и **filter** намерете произведението от нечетните утроени квадрати на елементите на **Nat**.

Задача 11. (7 т.) Даден е ориентиран граф (фиг. 2), като за всяка дъга е указано нейното тегло и за всеки възел е зададена стойността на евристичната функция **h,** определяща разстоянието от дадения възел до целевия възел **Goal**. Като се използва алгоритъма **A*** да се намери път от върха **START** до върха **GOAL**.



фиг. 2

a) Каква е максималната евристична оценка, която се получава в процеса на търсене на решението и за кой възел/път ?

b) Кой път ще бъде намерен от алгоритъма?

ЧЕРНОВА ЗА ЗАДАЧИ 9, 10 и 11

Задача 12. (2 т.) Допълнете кода на функцията **differ** така, че да проверява дали редицата от цели числа $a_0, a_1, \ldots, a_{n-1}$ се състои от различни елементи.

```
bool differ(int n, int a[])
     {int i = -1;
      bool b; int j;
      do
      { i++; j = .........;
        do
          {j++;
           b = a[i] != a[j];
          } while (b && ......................);
      } while (b && ........................);
       return ..................;
     }
```

Задача 13. (3 т.) Дефинирайте рекурсивна функция, която да проверява дали редица от числа е симетрична относно средата си.

ЧЕРНОВА ЗА ЗАДАЧИ 12 и 13

Задача 14. (6 т.) Намерете резултата от изпълнението на програмата.

```
#include <iostream.h>
class F
{public:
   F(double, double);
   void print() const;
   double f_x() const;
   double f_y() const;
   F& operator+(F&);
   F& operator-(F&);
 private:
     double x, y;
};
F::F(double a, double b)
{   x = a;
    y = b;
}
void F::print() const
{    cout << x << " " << y << endl;
}
double F::f_x() const
{    return x;
}
double F::f_y() const
{    return y;
}
F& F::operator+(F& a)
{    x = x + a.y;
     y = y + a.x;
     return *this;
}
F& F::operator-(F& a)
{    x = x - a.y;
     y = y - a.x;
     return *this;
}
class G
{public:
   G(F, F);
   void print() const;
   F f_x() const;
   F f_y() const;
 private:
      F x, y;
};
```

(Продължение на програмата)

```
G::G(F x, F y) : x(x+y), y(y-x){}
void G::print() const
{   x.print();
    y.print();
}
F G::f_x() const
{    return x;
}
F G::f_y() const
{    return y;
}
void main()
{   F f1(3,5), f2(1,1), f3(5,8);
    f1.print(); f2.print();
    (f1+f2+f3).print();
    (f1-f2-f3).print();
    G g(f2, f3);
    cout << g.f_x().f_x() +
            g.f_x().f_y() << " "
         << g.f_y().f_x() -
            g.f_y().f_y() << endl;
    g.print();
}
```

Решение:

Задача 15. (8 т.)Намерете резултата от изпълнението на програмата.

```
#include <iostream.h>
class A
{public:
    A(int a, double* b)
    {n = a;
     x = new double;
     *x = 1.5;
     cout << "A: " << n << "," << *x << endl;
    }
   A(const A& p)
   {n = p.n;
    x = new double;
    *x = *p.x;
    cout << "A(const A&): " << n << endl
         << *x << endl;
    }
   A& operator=(const A& p)
    {if(this != &p)
       {delete x;
        n = p.n + 5;
        x = new double;
        *x = *p.x + 1.5;
        cout << "A::operator=(p): " << n << endl
             << *x << endl;
       }
     return *this;
    }
   ~A()
   {cout << "~A()\n";
     delete x;
    }
 private:
     int n;
     double* x;
 };
class B
{   public:
    B(int a, double b)
    {n = a;
     x = b;
     cout << "B: " << n << "," << x << endl;
    }
   ~B()
   {cout << "~B()\n";
   }
 private:
    int n;
    double x;
  };
  class C
 {public:
    C(int a, double b)
    {n = a;
     x = b;
     cout << "C: " << n << "," << x << endl;
    }
   ~C()
   {cout << "~C()\n";
```

```
   }
   C(const C& p)
    {n = p.n + 3;
     x = p.x + 1.5;
     cout << "C(const C&): " << n << endl
          << x << endl;
    }
 private:
    int n;
    double x;
};
class D : public B, protected C, A
{public:
    D(int x=1, int y=3, double z=2.5): A(x, &z), B(y+x, z-x), C(x, z)
    {n = y;
     m = x;
     cout << "D: " << n << "," << m << endl;
    }
    ~D()
    {cout << "~D()\n";
    }
    D& operator=(const D& p)
    {if(this!=&p)
        {A::operator =(p);
         cout << "D::operator=(p)\n";
         n = p.n;
         m = p.m;
        }
       return *this;
     }
private:
     int n, m;
};
void main()
{D x(2), y(3, 7), z(5, 3, 1);
 D t = x;
 z = y;
}
```

Решение:

ЧЕРНОВА ЗА ЗАДАЧИ 14 и 15

Класът stack реализира стек с елементи от тип int.

фиг. 3

```
struct elem
{int inf;
 elem* link;
};
class stack
{public:
     stack();                                  // конструктор по подразбиране
     ~stack();                                                  // деструктор
     stack(stack const&);                       // конструктор за присвояване
     stack& operator=(stack const&);    //операторна функция за присвояване
     void push(int const&);                          // включва елемент в стек
     bool  pop(int &);            //изключва елемент от стек, ако е възможно
     bool empty() const;                        // проверка дали стек е празен
     void print();                                              // извежда стек
 private:
     elem *start;                              // указател към върха на стека
     void delstack();                                           // изтрива стек
     void copy(stack const&);             // копира указания стек в неявния
};
```

Задача 16. (3 т.) Реализирайте член-функциите от каноничното представяне за фиг.3.

Задача 17. (4 т.) Реализирайте член-функциите **`void copy(stack const&)`** и **`void delstack()`** от фиг.3.

Задача 18. (3 т.) Един стек е по-къс (**<**) от друг, ако се съдържа в него. Например, стекът с елементи **1,3,4** се съдържа в стека 9,1,4,2,**1,3,4**,7,8 и не се съдържа в стека **1**,2,**3**,5,6,**4**. Предефинирайте оператора **<** чрез член-функция на класа стек от фиг.3.

Задача 19. (3 т.) Нека **S** е стек от цели числа, а **f** е едноаргументна функция от вида: **f: int → int.** Дефинирайте функция от по-висок ред **map**:

```
stack map(int (*f)(int), stack& S);
```

която прилага **f** над всеки от елементите на **S** и връща получения стек.

ЧЕРНОВА ЗА ЗАДАЧИ 16, 17, 18 и 19

```
Шаблонът на класа BinOrdTree реализира двоично-наредено дърво с върхове от тип T.
template <class T>
struct node_bin
{T inf;
 node_bin<T> *Left;
 node_bin<T> *Right;
};
template <class T>
class BinOrdTree
{public:
    BinOrdTree();                                  // конструктор по подразбиране
    ~BinOrdTree();                                                  // деструктор
    BinOrdTree(BinOrdTree<T> const&);          // конструктор за присвояване
    BinOrdTree<T>& operator=(BinOrdTree<T> const&);// операторна функция за =
    bool empty() const;                       // проверява дали дървото е празно
    T RootTree() const;              // връща корена на двоично-наредено дърво
    BinOrdTree LeftTree() const;                     // връща лявото поддърво
    BinOrdTree RightTree() const;                   // връща дясното поддърво
    void AddNode(T const & x)  // включва указания елемент в дв. нар. дърво
    { Add(root, x);
    }
    void DeleteNode(T const&);                     // изтрива указания елемент
// Create3 създава дв. нар. дърво по указани корен, ляво и дясно поддърво
    void Create3(T const&, BinOrdTree<T> const&, BinOrdTree<T> const&);
   // ……….
 private:
    node_bin<T> *root;
  // DeleteTree изтрива двоично-нареденото дърво, зададено чрез указателя p
    void DeleteTree(node_bin<T>* &p) const;
  // CopyTree копира указаното в неявното двоично-наредено дърво
      void CopyTree(BinOrdTree<T> const&);
  // Copy копира указаното от указателя p двоичо наредено дърво на ново място
  // в паметта, указано от указателя q
     void Copy(node_bin<T>* &q, node_bin<T>* const&p) const;
};
```

фиг. 4

Задача 20. (3 т.) Реализирайте член-функциите: **T RootTree()** const, **BinOrdTree LeftTree() const** и **BinOrdTree RightTree() const** за фиг.4

Задача 21. (3 т.) Към шаблона на класа **`BinOrdtree`** (фиг. 4) добавете и конструктор, който създава идеално балансирано двоично-наредено дърво с n елемента (Двоично дърво е идеално балансирано ако всеки негов връх има ляво и дясно поддърво, в които броят на възлите се различава най-много с 1).

Задача 22. (3 т.) Дефинирайте шаблон на външна функция, която извежда върховете от n-то ниво на непразното двоично-наредено дърво d (коренът се счита за връх от 0-во ниво). Де се използва дефиницията от фиг. 4.

Задача 23. (4 т.) Предефинирайте оператора **>>**, така че да извежда двоично-наредено дърво, обхождайки го в реда: ляво-корен-дясно. Да се използва дефиницията от фиг. 4.

Задача 24. (4 т.) Даден е стек от цели числа. Да се напише програмен фрагмент, който сортира елементите на стека като за целта използва двоично-наредено дърво. Да се използват дефинициите от фиг. 3 и фиг. 4.

ЧЕРНОВА ЗА ЗАДАЧИ 20, 21, 22, 23, и 24