СОФИЙСКИ УНИВЕРСИТЕТ
“СВ. КЛИМЕНТ ОХРИДСКИ”

ФАКУЛТЕТ ПО МАТЕМАТИКА
И ИНФОРМАТИКА

ДЪРЖАВЕН ИЗПИТ

ЗА ПОЛУЧАВАНЕ НА ОКС “БАКАЛАВЪР ПО КОМПЮТЪРНИ НАУКИ”

ЧАСТ I (ПРАКТИЧЕСКИ ЗАДАЧИ)

Драги абсолвенти:

- Попълнете факултетния си номер в горния десен ъгъл на всички листове.
- Пишете само на предоставените листове, без да ги разкопчавате.
- Решението на една задача трябва да бъде на същия лист, на който е и нейното условие (т.е. може да пишете отпред и отзад на листа със задачата, но не и на лист на друга задача).
- Ако имате нужда от допълнителен лист, можете да поискате от квесторите.
- На един лист не може да има едновременно и чернова, и белова.
- Черновите трябва да се маркират, като най-отгоре на листа напишете “ЧЕРНОВА”.
- Ако решението на една задача не се побира на нейния лист, трябва да поискате нов бял лист от квесторите. Той трябва да се защипе с телбод към листа със задачата.
- Всеки от допълнителните листове (белова или чернова) трябва да се надпише най-отгоре с вашия факултетен номер.
- Черновите също се предават и се защипват в края на работата.
- Времето за работа по изпита е 3 часа.

Изпитната комисия ви пожелава успешна работа!

Задача 1. Задачата да се реши на езика C++.

1) Да се довърши кода на `power` така, че тя да пресмята повдигането на `base` на степен `power`.

```
int power(int base, unsigned int power)
{
  int result = ____________________;

  while(power > ________________) {
    if(power % 2 == __________) {
      ___________ = result * base;
    }

    __________ /= 2;
    __________ *= base;
  }

  return result;
}
```

2) Да се посочи какво ще изведе на екрана даденият по-долу фрагмент.

```
int a=10, b=010, c=0x10;
cout << a << ", " << b << ", " << c;
```

_____

3) Под всеки от фрагментите да се посочи какво ще изведе той на екрана.

```
int var = 0;
while(var++ < 5)
  cout << var;
```

_____

```
int var = 0;
do {
  cout << var;
} while(++var < 5);
```

_____

4) Да се посочи какво ще изведе фрагментът. Допускаме, че `sizeof(unsigned int) == 4`.

```
unsigned int x = 0x11335577;
unsigned char* ptr = (unsigned char*)&x;
++*ptr;
++*(ptr+1);
++ptr[2];
++3[ptr];
cout << hex << "0x" << x;
```

_____

5) Нека `a` и `b` са променливи от тип `char`. Да се посочи какъв е проблемът в дадения по-долу фрагмент.

```
char result = a + b;
```

_____

Да се посочи възможен начин, чрез който може да се реши този проблем.

_____

6) Нека са дадени следните дефиниции:

```
const size_t size = 3;
char box[size][size] = {
  'a', 'b', 'c',
  'd', 'e', 'f',
  'g', 'h', 'i'
};
```

Да се довършат фрагментите по-долу така, че да извеждат съответните последователности от букви.

```
// Трябва да извежда aei
for (int i = 0; i < size; i++)
  cout << box[_______________][_______________];

// Трябва да извежда ceg
for (int i = 0; i < size; i++)
  cout << box[_______________][size__________];

// Трябва да извежда gec
for (int i = 0; i < size; i++)
  cout << box[size__________][_______________];

// Трябва да извежда abcfihgd
for(int i = 0; i < size; ++i)
  cout << box[0][i];
for(int i = __________; i < size; __________)
  cout << box[i][size-1];
for(int i = size_______; i >= 0; --i)
  cout << box[size-1][i];
for(int i = size-2; i _____ 0; --i)
  cout << box[i][0];
```

Критерии за оценяване

- Точки се дават само за напълно коректно посочени отговори.
- В подточките, в които се изисква да се посочи какво ще се изведе, точки се дават само ако отговорът напълно съвпада с това, което извежда съответният код. В противен случай се дават 0 т.
- В задачата за довършване на кода на функцията, ако написаното не е синтактично или логически коректно или е различно от коректния отговор, се дават 0 т.
- Сумата от точките се закръгля до цяло число.

Максималната оценка за всяка подточка е както следва:

1. 3 точки (по 0,5 за всяко коректно попълнено място);
2. 1 точка (по 0,3 за всяка коректно посочена стойност и още 0,1 ако са посочени и запетаите);
3. 1 точки (по 0,5 за всеки фрагмент);
4. 1 точка;
5. 1 точка (по 0,5 за всеки подвъпрос);
6. 3 точки (по 0,3 за всяко коректно попълнено място);

Примерно решение:

1)

```
int power(int base, unsigned int power)
{
  int result = 1;

  while(power > 0) {
    if(power % 2 == 1) {
      result = result * base;
    }

    power /= 2;
    base *= base;
  }

  return result;
}
```

2) `10, 8, 16`

3) `12345` за първия фрагмент и `01234` за втория фрагмент.

4) `0x12345678`

5) Възможно е резултатът от събирането на двете променливи да не се побира в променлива от тип `char`. Например, ако допуснем `sizeof(char) == 1` и също, че двете променливи съдържат стойността 100.

Проблемът може да се реши като за резултата се използва променлива от тип, който може да побере всевъзможните стойности, които биха се получили при събирането.

6)

```
// Трябва да извежда aei
for (int i = 0; i < size; i++)
  cout << box[i][i];

// Трябва да извежда ceg
for (int i = 0; i < size; i++)
  cout << box[i][size-i-1];

// Трябва да извежда gec
for (int i = 0; i < size; i++)
  cout << box[size-i-1][i];

// Трябва да извежда abcfihgd
for(int i = 0; i < size; ++i)
  cout << box[0][i];
for(int i = 1; i < size; ++i)
  cout << box[i][size-1];
for(int i = size-2; i >= 0; --i)
  cout << box[size-1][i];
for(int i = size-2; i > 0; --i)
  cout << box[i][0];
```

Задача 2. Задачата да се реши на езика C++.

Документ наричаме символен низ с произволна дължина, който има име, също символен низ. Текстът в документа е съставен от редове, разделени със символа `'\n'`.

Папка наричаме контейнер, който може да съдържа както документи, така и други папки. Казваме, че `subfolder` е **вложена папка** във `folder`, ако `subfolder` е елемент на контейнера `folder` или е вложена в някоя папка, която е вложена във `folder`.

В следната програмата на езика за програмиране C++ липсват части.

Класът `Document` описва документ, а класът `Folder` описва папка. Класът `File` е абстрактен базов клас за `Document` и `Folder`, който дефинира операцията `search` за търсене на символен низ в йерархия от документи и папки.

Резултат от търсенето на низа `str` в даден документ `doc` наричаме такава тройка `(name,N,line)`, където `name` е името на документа `doc`, `line` е съдържанието на някой ред в документа, съдържащ `str` като подниз, а `N` е поредния номер на `line` в документа.

Резултатът от `doc.search(str)` е вектор с всички резултати от търсенето на `str` в `doc`.

При търсене в папка `folder`, `folder.search(str)` е вектор с всички резултати от търсенето на `str` в документите, съдържащи се във `folder` или в нейните вложени папки.

Да се попълнят липсващите части в програмата. При правилно заместване на празните места, програмата ще изведе следното на стандартния изход:

```
employees.txt, line 1: John Smith
employees.txt, line 2: Jane Smith
```

Да се приеме, че класът `Folder` не е нужно да прави копие на вложените обекти. Можете да използвате всякакви библиотечни функции като допишете съответните `#include` директиви.

```
#include <vector>
#include <iostream>


struct SearchResult
{
    std::string fileName;
    unsigned line_number;
    std::string line;
};
class File
{   public:

    ____________ std::vector<SearchResult> search(____________) const ____________

    virtual ~File(){}
};
class Document: public File
{   std::string name;
    std::string contents;
    public:
    Document(____________ _name,____________ _contents):_____________________________

    std::vector<SearchResult> search(____________ str) const
    {
        std::vector<SearchResult> result;
```

```
        return result;
    }
};
class Folder: public File
{
    ____________________  files;
    std::string name;
    public:
    Folder(____________ _name):____________________

    void addFile(____________ f)
    {files.push_back(f);}

    std::vector<SearchResult> search(____________ str) const
    {
        std::vector<SearchResult> result;









        return result;
    }
};
int main()
{   Document d1("employees.txt","John Smith\nMaryia Ivanova\n"),
             d2("inventory.txt","Computers: 3\nPrinters: 1"),
             d3("employees.txt","Ivan Petrov\nJane Smith"),
             d4("inventory.txt","Computers: 5, 3D Printers: 1");
    Folder root("root"), acme("ACME Soft, Inc."), best("Best Soft, OOD");
    acme.addFile(&d1); acme.addFile(&d2);
    best.addFile(&d3); best.addFile(&d4);
    root.addFile(&acme); root.addFile(&best);
    std::vector<SearchResult> results = root.search("Smith");
    for(unsigned i = 0; i < results.size(); ++i)
    {
        std::cout << results[i].fileName
                  << ", line " << results[i].line_number << ": "
                  << results[i].line << std::endl;
    }
}
```

Примерно решение

```
#include <vector>
#include <iostream>
#include <sstream>
#include <iostream>
struct SearchResult
{
    std::string fileName;
    unsigned line_number;
    std::string line;
};
class File
{
    public:
    virtual std::vector<SearchResult> search(const std::string&) const=0;
};
class Document: public File
{
    std::string name;
    std::string contents;
    public:
    Document(const std::string& _name,const std::string& _contents)
              :name(_name),contents(_contents){}
    std::vector<SearchResult> search(const std::string& s) const
    {
        std::istringstream doc(contents);
        std::string line;
        unsigned lineCount=0;
        std::vector<SearchResult> result;
        while(std::getline(doc,line))
        {
            ++lineCount;
            if (line.find(s) != std::string::npos)
            {
                result.push_back({name,lineCount,line});
            }
        }
        return result;
    }
};
class Folder: public File
{
    std::vector<File*> files;
    std::string name;
    public:
    Folder(const std::string &_name):name(_name){}
    void addFile(File *f)
    {files.push_back(f);}
    std::vector<SearchResult> search(const std::string& s) const
    {
        std::vector<SearchResult> result;
        for(const File * f:files)
```

```
        {
            std::vector<SearchResult> found = f->search(s);
            result.insert(result.end(),found.begin(),found.end());
        }
        return result;
    }
};
int main()
{
    Document d1("employees.txt","John Smith\nMaryia Ivanova\n"),
             d2("inventory.txt","Computers: 3\nPrinters: 1"),
             d3("employees.txt","Ivan Petrov\nJane Smith"),
             d4("inventory.txt","Computers: 5, 3D Printers: 1");
    Folder root("root"), acme("ACME Soft, Inc."), best("Best Soft, OOD");
    acme.addFile(&d1); acme.addFile(&d2);
    best.addFile(&d3); best.addFile(&d4);
    root.addFile(&acme); root.addFile(&best);
    std::vector<SearchResult> results = root.search("Smith");
    for(unsigned i = 0; i < results.size(); ++i)
    {
        std::cout << results[i].fileName
                  << ", line " << results[i].line_number << ": "
                  << results[i].line << std::endl;
    }
}
```

Критерии за оценяване

Задачата се оценява по долната схема, като сумата на точките се дели на 3 и се закръгля до цяло число.

```
#include <vector>
#include <iostream>
struct SearchResult
{
    std::string fileName;
    unsigned line_number;
    std::string line;
};
class File
{
    public:

    ____________ std::vector<SearchResult> search(____________) const ____________
    [*] virtual ...(std::string&) = 0;       +2 т.
    [*] друго вярно                          +1 т.
};
class Document: public File
{
    std::string name;
    std::string contents;


    public:
```

```
    Document(_____ _name,_____ _contents):_____
    [*] const std::string&   ИЛИ            +2 т.
    [*] друго вярно                          +1 т.

    [*] правилна инициализация               +2 т.

    std::vector<SearchResult> search(_____ str) const
    {
        std::vector<SearchResult> result;









    [*] вярна реализация                     +8 т.
        return result;
    }
};
class Folder: public File
{
    _____  files;
    [*] std::vector<File*>                   +2 т.

    std::string name;
    public:
    Folder(_____ _name):_____
    [*] const std::string&   ИЛИ            +2 т.
    [*] друго вярно                          +1 т.

    [*] правилна инициализация               +2 т.

    void addFile(_____ f)
    [*] File*                                +2 т.
    {files.push_back(f);}

    std::vector<SearchResult> search(_____ str) const
    {
        std::vector<SearchResult> result;
```

```
    [*] вярна реализация                    +8 т.
        return result;
    }
};
int main()
{
    Document d1("employees.txt","John Smith\nMaryia Ivanova\n"),
             d2("inventory.txt","Computers: 3\nPrinters: 1"),
             d3("employees.txt","Ivan Petrov\nJane Smith"),
             d4("inventory.txt","Computers: 5, 3D Printers: 1");
    Folder root("root"), acme("ACME Soft, Inc."), best("Best Soft, OOD");
    acme.addFile(&d1); acme.addFile(&d2);
    best.addFile(&d3); best.addFile(&d4);
    root.addFile(&acme); root.addFile(&best);
    std::vector<SearchResult> results = root.search("Smith");
    for(unsigned i = 0; i < results.size(); ++i)
    {
        std::cout << results[i].fileName
                  << ", line " << results[i].line_number << ": "
                  << results[i].line << std::endl;
    }
}
```

Задача 3. Разглеждаме дърво на директориите на файлова система. Всяка директория може да има произволен брой наследници. Не се допускат две директории с едно и също име и общ пряк родител (в една директория не може да има две директории с едно и също име). Всички възли имат име – символен низ с дължина не по-голяма от 12 символа, съставено от малки или главни латински букви, цифри, както и символа точка. В тази задача не се изисква да се разглеждат други характеристики на директориите.

Корен на дървото винаги е главната директория. Тя е единственият елемент с празно име.

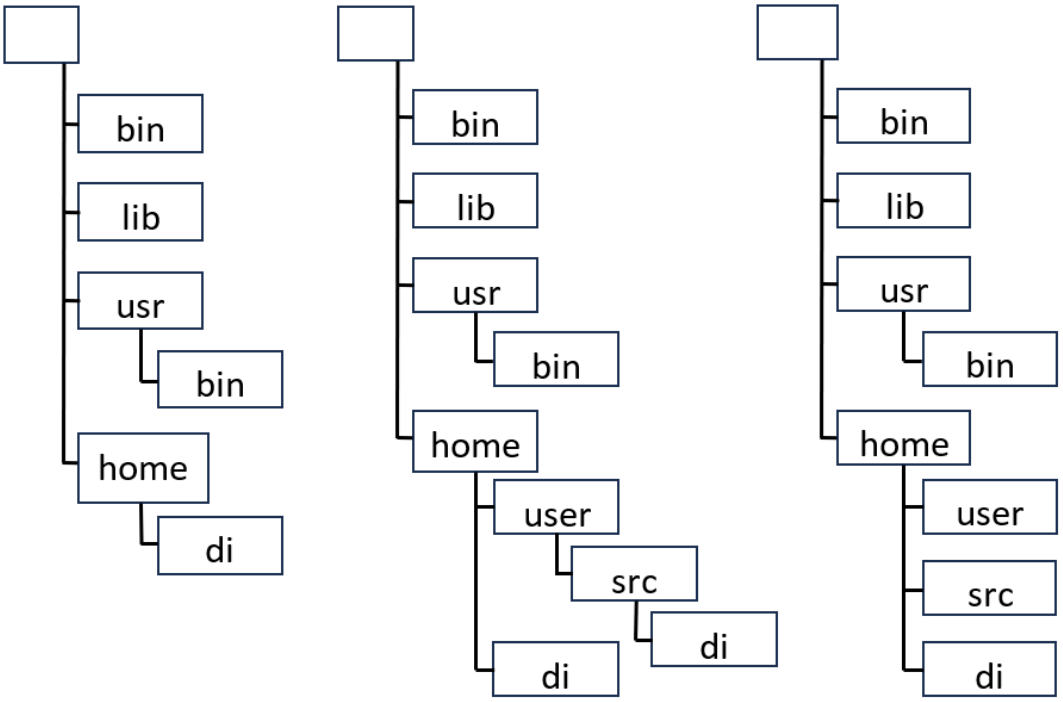
1. Да се опише структура, с която можете да се представи такова дърво и да се напише нейната дефиниция на C++.

2. Да се дефинира функция `insert`, която получава като аргументи дърво на файлова система и символен низ, описващ пълен път към директория, като за разделител се използва символът `'/'`. Функцията трябва да добави тази директория в дървото, като при нужда създава всички липсващи родителски директории от пълния път до нея. Например, ако е зададен пълен път `"/home/user/src"`, след приключване на работата на функцията в дървото със сигурност трябва да има елемент с име `"home"`, който е наследник на корена; елемент с име `"user"`, който е наследник на елемента `"home"`, и елемент с име `"src"`, който е наследник на елемента `"user"`.

3. Да се дефинира функция `flatten`, която получава като аргумент дърво на файловата система и пълен път към елемент в него. Функцията трябва да преобразува всички непреки наследници на този елемент в негови преки наследници, т.е. всички елементи на поддървото на елемента трябва да станат преки наследници на корена му. Ако има повтарящи се имена, функцията трябва да остави само един пряк наследник с това име.

Задачата трябва да се реализира на езика C++. Позволено е използването на класовете и методите от стандартната библиотека, отговаряща на стандарта C++14 или по-стар.

Пример: На първата фигура е показано примерно дърво на директориите. На следващата към това дърво е извършено добавяне на директорията `/home/usr/src/di`. На последната е показан резултатът от прилагане на `flatten` към директорията `/home` на предходното дърво.



Примерно решение

```
#include <string>
#include <vector>
#include <map>
#include <iostream>

using std::string;
using std::vector;
struct node
{
    std::map<string, node*> children;
};
using dir_map = std::map<string, node*>;

struct tree
{
    node root;
};

std::vector<string> split(const string& path) {
    const char delimeter = '/';

    std::vector<string> result;
    size_t pos = 0;
    while (pos < path.length()) {
        size_t end = path.find(delimeter, pos);
        if (end == string::npos) {
            end = path.length();
        }
        if (end-pos > 1) result.push_back(path.substr(pos, end - pos));
        pos = end + 1;
    }
    return result;
}


bool insert(node& n, const vector<string>& dirs, int index)
{
    if (index == dirs.size()) return true;
    if (dirs[index].empty()) return false;
    dir_map::iterator it = n.children.find(dirs[index]);
    if (it == n.children.end()) {
        n.children.insert({ dirs[index], new node });
    }
    return insert(*(n.children[dirs[index]]), dirs, index + 1);
}

bool insert(tree& tree, const string& path)
{
    if (path.empty() || path[0] != '/') return false;
```

```
    std::vector<string> dirs {split(path)};
    return insert(tree.root, dirs, 0);
}

void clear(node& n)
{
    for (auto& v : n.children) {
        clear(*v.second);
        delete v.second;
}
}
void clear(tree& t)
{
    clear(t.root);
}


void print(const node* n, int depth)
{
    for (const auto& val : n->children) {
        for (int i = 0; i < depth; ++i)
            std::cout << "  ";
        std::cout << val.first << '\n';
        print(val.second, depth + 1);
    }
}
void print(const tree& t)
{
    print(&t.root, 0);
    std::cout << std::endl;
}



void add_all(dir_map& children, node* root) {
    for (auto& val : root->children) {
        add_all(children, val.second);
        children[val.first] = val.second;
        val.second->children.clear();
    }
}

void flatten(node* n, const vector<string>& path)
{
    for (const string& dir : path) {
        dir_map::iterator it = n->children.find(dir);
        if (it == n->children.end()) {
            std::cout << "Path not found!\n";
            return;
        }
```

```
        n = it->second;
    }
    add_all(n->children, n);
}
void flatten(tree& t, const string& path)
{
    flatten(&t.root, split(path));
}

int main()
{
    tree t;
    insert(t, "/lib");
    insert(t, "/usr");
    insert(t, "/bin");
    insert(t, "/home/di");
    insert(t, "/usr/bin");
    print(t);

    insert(t, "/home/usr/src/di");
    print(t);

    flatten(t, "/home");
    print(t);

    clear(t);
    return 0;
}
```

Критерии за оценяване

1. За коректно дефинирана структура – 1 т.
   Допускат се различни коректни дефиниции.

2. `insert` – общо 4 т., от които:

   а) за разделяне на пътя на коректни части по разделител – 1 т.

   б) за проследяване на съществуващ път – 1 т.

   в) за коректно създаване на елемент – 0,5 т.

   г) за добавяне на нов елемент на правилното място при построяване на новата част от пътя – 1,5 т.

3. `flatten` – общо 5 т., от които:

   - за намиране на възела, който ще бъде корен на трансформацията – 1 т.
   - за намиране на всички възли в наследниците (обхождане) – 2 т.
   - за преместването им като наследници на родителя и избягване на повторения – 1 т.
   - за превръщането на всички непреки наследници в листа (т.е. премахването на наследниците им) – 1 т.

Задача 4. Моделът „ChatFOS 5“ може да генерира прости функционални изрази, съдържащи една променлива `x` и извиквания на двуместни числови функции от предварително зададена „библиотека“, които моделът познава и може да използва. Библиотеката се представя със списък от наредени двойки: символ на функцията и самата функция. Изразите на Haskell са представени с дървета от следния алгебричен тип: `data Expr = N Integer | X | F Char Expr Expr`, където конструкторът `N` представя числова константа, `X` представя променливата на израза, а `F` представя функция със зададен символ и два израза за аргументи. Изразите на Scheme използват синтаксиса за S-изрази.

Да се попълнят празните полета по-долу на един от езиците Haskell или Scheme така, че:

а) функцията `correct` да проверява дали даденият израз е коректно построен само чрез функциите от библиотеката `lib`, цели числа и символа `x`;

б) функцията `calc` да връща едноместната функция, пресмятана от даден израз при библиотека от функции `lib`, като допуска, че изразът е коректно построен;

в) функцията `check` да проверява дали генерираният израз `expr` правилно пресмята функцията `f`, като проверява дали те връщат еднакви стойности над всички цели числа в интервала $[a; b]$;

г) функцията `score` намира точността на модела (число в интервала $[0; 1]$) като по дадени списъци `exprs` от генерирани изрази и `fs` от функции, които те би трябвало да пресмятат, чрез `check` намира отношението на броя на правилно пресметнатите функции към броя на всички функции.

Упътване: *могат да се използват наготово всички функции в R5RS за Scheme и в Prelude за Haskell.*

Haskell

```
correct lib (F f l r) = ____________________ && ____________________ && ____________________
correct lib _         = ______
calc lib (N n)  ____________
calc lib X      ____________
calc lib (F c l r) ____________________________________________________________________
  ____________________________________________________________
check a b lib f expr = ____________ (\x -> ____________________ && ____________________) [a..b]
score a b lib fs exprs =   fromIntegral (length (filter ____________________________________))
                         / fromIntegral (length ______)
```

Пример:

```
exprs = [F '-' (F '*' X X) (N 1), F '+' (F '*' X (N 2)) (N 3),
         F '*' (F '*' X X) (F '+' X (N 2)), F '+' (F '^' X (N 3)) (N 1)]
lib = [('+',(+)),('-',(-)),('*',(*))]                  -- calc lib (head exprs) 3 ⟶ 8
score (-10) 10 lib [\x -> x^2-1, \x -> x*3+2, \x -> x^3+2*x^2, \x -> x^3+1] exprs ⟶ 0.5
```

Scheme

```
(define (correct lib expr)
  (if (pair? expr) (and ____________________ ____________________ ____________________)
      (or ____________________ ____________________)))
(define (calc lib expr) (____________________
    (cond ((number? expr) ______)
          ((eqv? expr 'x) ______)
          (else ____________________________________________________________________))))
(define (check a b lib f expr) (if (> a b) ______
  (and ____________________ ____________________________________ (check (+ a 1) b lib f expr))))
(define (score a b lib fs exprs) (/ (length (filter ____________________________________))
                                    (length ______)))
```

Пример:

```
(define lib (list (cons '+ +) (cons '- -) (cons '* *)))   ; ((calc lib '(- (* x x) 1)) 3) ⟶ 8
(define fs (list (lambda (x) (- (expt x 2) 1)) (lambda (x) (+ 2 (* x 3)))
                 (lambda (x) (+ (expt x 3) (* 2 x x))) (lambda (x) (+ 1 (expt x 3)))))
(score -10 10 lib fs '((- (* x x) 1) (+ (* x 2) 3) (* (* x x) (+ x 2)) (+ (^ x 3) 1))) ⟶ 0.5
```

Примерни решения

Haskell

```
correct lib (F f l r) = elem f (map fst lib) && correct lib l && correct lib r
correct lib _         = True

calc lib (N n)   _ = n
calc lib X           x = x
calc lib (F c l r) x = f (calc lib l x) (calc lib r x)
  where (f:_) = [ f | (fc, f) <- lib, fc == c ]

check a b lib f expr = all (\x -> correct lib expr && calc lib expr x == f x) [a..b]

score a b lib fs exprs = fromIntegral (length (filter id (zipWith (check a b lib) fs exprs)))
                       / fromIntegral (length fs)
```

Scheme

```
(define (correct lib expr)
  (if (pair? expr)
      (and (memv (car expr) (map car lib))
           (correct lib (cadr expr)) (correct lib (caddr expr)))
      (or (number? expr) (eqv? expr 'x))))

(define (calc lib expr) (lambda (x)
    (cond ((number? expr) expr)
          ((eqv? expr 'x) x)
          (else ((cdr (assv (car expr) lib))
                 ((calc lib (cadr expr)) x) ((calc lib (caddr expr)) x))))))

(define (check a b lib f expr) (if (> a b) #t
  (and (correct lib expr) (= ((calc lib expr) a) (f a)) (check (+ a 1) b lib f expr))))

(define (score a b lib fs exprs)
  (/ (length (filter (lambda (x) x) (map (lambda (f expr) (check a b lib f expr)) fs exprs)))
     (length fs)))
```

Критерии за оценяване

Ако е писано и по двата езика, се взима по-високият резултат. Сумата от точките се закръгля до цяло число.

Haskell

По ред на празните слотове:

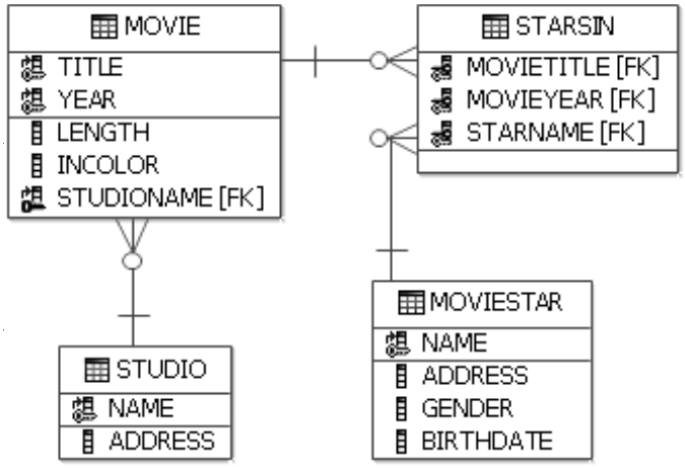
1. **0,5 т.** за `elem f (map fst lib)` или еквивалентен израз, който проверява дали символът `f` се среща сред първите компоненти на наредените двойки в списъка `lib`;
2. **0,5 т.** за едното от двете рекурсивни извиквания `correct lib l` и `correct lib r`;
3. **0,5 т.** за другото рекурсивно извикване;
4. **0,5 т.** за константата `True`;
5. **0,5 т.** за добавяне на параметър, последван от знака за равенство и променливата `n`;
   - признава се и знак за равенство, последван от `const n` или едноместна λ-функция, която връща `n` без значение от стойността на параметъра си;
6. **0,5 т.** за добавяне на параметър, последван от знака за равенство и името на параметъра;
   - признава се и знак за равенство, последван от `id` или едноместна λ-функция, която връща единствения си параметър;
7. **2 т.** за добавяне на параметър, последван от знака за равенство и коректна реализация, която намира в `lib` функцията, съответстваща на символа `c` и я прилага над резултатите от рекурсивните извиквания `calc lib l` и `calc lib r`;
   - признава се и знак за равенство, последван от едноместна λ-функция, чието тяло съдържа коректна реализация, както е описано по-горе;
8. този слот се разглежда и оценява съвместно с предишния;
9. **0,5 т.** за функцията `all`;
   - признава се и алтернативна реализация с `filter` и сравнение с празния списък;
10. **1 т.** за един от следните изрази: извикване на `correct lib expr` и проверка дали стойността на `f x` съвпада с `calc lib expr x`;
11. **1 т.** за другия от горните изрази;
12. **2 т.** за коректно поелементно филтриране на тези функции и съответните им изрази, които дават еднакъв резултат при подаването им на функцията `check`;
13. **0,5 т.** за параметъра `fs` или `exprs`.

Scheme

По ред на празните слотове:

1. **0,5 т.** за `(memv (car expr) (map car lib))` или еквивалентен израз, който проверява дали първият елемент на S-израза `expr` се среща сред първите компоненти на наредените двойки в списъка `lib`;
2. **0,5 т.** за едното от двете рекурсивни извиквания на функцията `correct` над втория или третия елемент на S-израза `expr`;
3. **0,5 т.** за другото рекурсивно извикване;
4. **0,25 т.** за един от двата израза `(number? expr)` и `(eqv? expr 'x)`;
   - признава се също и използване на предикатите `integer?`, `eq?` и `equal?`;
5. **0,25 т.** за другия от горните два израза;
6. **0,5 т.** за специалната форма `lambda`, последвана от списък от точно един параметър;
7. **0,25 т.** за `expr`;
8. **0,25 т.** за параметъра от слот 6;
9. **2 т.** за коректна реализация, която намира в `lib` функцията, съответстваща на първия елемент на S-израза `expr`, и я прилага над резултатите от прилагането на рекурсивните извиквания на функцията `calc` над `lib` и втория и третия елемент на S-израза `expr` над параметъра от слот 6;
10. **0,5 т.** за константата `#t`;
11. **1 т.** за един от следните изрази: извикване на `(correct lib expr)` и проверка дали стойността на `((calc lib expr) a)` съвпада с `(f a)`;
12. **1 т.** за другия от горните изрази;
13. **2 т.** за коректно поелементно филтриране на тези функции и съответните им изрази, които дават еднакъв резултат при подаването им на функцията `check`;
14. **0,5 т.** за параметъра `fs` или `exprs`.

Задача 5. Дадена е базата от данни `Movies`, в която се съхранява информация за филми, филмови студиа, които ги произвеждат, както и актьорите, които участват в тях.



Таблицата `Studio` съдържа информация за филмови студиа:

- `name` — име, първичен ключ;
- `address` — адрес.

Таблицата `Movie` съдържа информация за филми. Атрибутите `title` и `year` заедно формират първичния ключ.

- `title` — заглавие;
- `year` — година, в която е заснет филмът;
- `length` — дължина в минути;
- `incolor` — `'Y'` за цветен филм и `'N'` за черно-бял;
- `studioname` — име на студио, външен ключ към `Studio.name`.

Таблицата `MovieStar` съдържа информация за филмови звезди:

- `name` — име, първичен ключ;
- `address` — адрес;
- `gender` — пол, `'M'` за мъж (актьор) и `'F'` за жена (актриса);
- `birthdate` — рождена дата.

Таблицата `StarsIn` съдържа информация за участието на филмовите звезди във филмите. Трите атрибута заедно формират първичния ключ. Атрибутите `movietitle` и `movieyear` образуват външен ключ към `Movie`.

- `movietitle` — заглавие на филма;
- `movieyear` — година на заснемане на филма;
- `starname` — име на филмовата звезда, външен ключ към `MovieStar.name`.

1) Да се напише заявка, която извежда без повторение имената на всички актриси, които са играли във филми както през 20-и век (до 2000 г. включително), така и през 21-и век (от 2001 г. насам).

2) Да се посочи коя от следните заявки извежда без повторение имената на всички студиа, на които сумарната дължина на всички техни черно-бели филми е по-голяма от дължината на най-дългия филм на това студио (без значение дали е цветен):

А)
```
SELECT name
FROM Studio
LEFT JOIN Movie
     ON name = studioname
     AND incolor = 'N'
GROUP BY name
HAVING SUM(length) > MAX(length);
```

Б)
```
SELECT DISTINCT name
FROM Studio
RIGHT JOIN Movie
      ON name = studioname
      AND incolor = 'N'
WHERE SUM(length) > MAX(length);
```

В)
```
SELECT name
FROM Studio
WHERE (SELECT SUM(length)
       FROM Movie
       WHERE incolor = 'N'
       AND studioname = Studio.name)
   >  (SELECT MAX(length)
       FROM Movie
       WHERE studioname = Studio.name);
```

Г)
```
SELECT DISTINCT studioname
FROM Movie m
WHERE (SELECT SUM(length)
       FROM Movie
       WHERE incolor = 'N'
         AND studioname = m.studioname)
    > (SELECT MAX(length)
       FROM Movie);
```

Примерно решение на подзадача 1:

```
SELECT starname
FROM StarsIn
JOIN MovieStar ON starname = name
WHERE movieyear <= 2000 AND gender = 'F'
INTERSECT
SELECT starname
FROM StarsIn
JOIN MovieStar ON starname = name
WHERE movieyear > 2000 AND gender = 'F';
```

Критерии за оценяване:

1) Общо 5 т. за изцяло коректно решение, като:
   - точките се намаляват с 3 при комбиниране на двете условия за годините по такъв начин, че да се извеждат всички актриси, които са играли в произволен филм (например, вариант на примерното решение с използване на `UNION`, вместо `INTERSECT`);
   - за всяка сгрешена клауза броят на точките се намалява с 2 до достигане на 0 т.

2) Единственият верен отговор е В). Посочването на този отговор носи 5 т., а посочването на грешен отговор или комбинация от отговори носи 0 т.

Задача 6. В тази задача става дума за обикновени неориентирани графи. С "$d(a)$" означаваме степента на върха a. "Триъгълник в граф" означава подграф с три върха, всеки от които е съсед на всеки от другите два върха в триъгълника. Нека G е граф. Нека n е броят на върховете му. Нека (u, v) е ребро в G. Да се докаже, че за всяко $k \in \{0, 1, \ldots, n-2\}$ е вярно, че ако $d(u) + d(v) = n + k$, то (u, v) е ребро в поне k различни триъгълника.

Примерно решение

Иска се да се покаже, че u и v имат поне k общи съседи. Нека A е множеството от съседите на u. Нека v е множеството от съседите на v.

Да разгледаме $|A \cup B|$. Съгласно комбинаторният принцип на включването и изключването, в сила е

$$|A \cup B| = |A| + |B| - |A \cap B|$$

Но $d(u) = |A|$, $d(v) = |B|$, така че $|A| + |B| = n + k$. Тогава

$$|A \cup B| = n + k - |A \cap B|$$

Забелязваме, че $|A \cup B| \leq n$, защото множеството $A \cup B$ е подмножество на V. Тогава

$$n + k - |A \cap B| \leq n \ \leftrightarrow \ k - |A \cap B| \leq 0 \ \leftrightarrow \ |A \cap B| \geq k.$$

Доказахме, че общите съседи на u и v са поне k на брой.

Критерии за оценяване

Всяко коректно доказателство се оценява с 10 точки.

Всяко грешно доказателство се оценява с 0 точки.

Задача 7. Даден е ориентиран граф, представен чрез поредица от факти от вида `arc(<Node1>, <Node2>, <Cost>)`, всеки от които означава дъга с начало `<Node1>`, край `<Node2>` и цена `<Cost>`:

```
arc(a,b,5).  arc(a,c,5).  arc(b,d,3).  arc(b,e,2).  arc(c,d,2). arc(c,g,14).
arc(d,f,6).  arc(e,g,9).  arc(f,g,7).  arc(f,i,3).
```

Дадена е също така поредица от факти от вида `h(<Node>,<Cost>)`, дефиниращи евристичната функция, с помощта на която се пресмята приближена стойност `<Cost>` на разстоянието от възела `<Node>` до възела `g`:

```
h(a,15).  h(b,11).  h(c,10).  h(d,10).  h(e,9).  h(f,7).  h(g,0).  h(i,4).
```

Ако се търси път в графа от възела `a` до възела `g`, да се посочи решението, намерено с използване на всеки от посочените по-долу методи за търсене.

а) Beam search при широчина на лъча `BeamSize=3`;

   Решението да се обоснове, като се посочи как beam search избира следващия връх, който да обходи (каква е неговата оценяваща функция);

б) A* search;

в) Hill climbing.

Примерно решение

Търсеният път за всеки от методите за търсене е:

а) `[a,c,g]`

Методът Beam search разширява този частичен път от фронта на търсене, който е с най-малка стойност на евристичната функция `h(n)`, която оценява оставащата част от пътя до целта. На всяка стъпка фронтът се ограничава до определения брой най-добри състояния.

б) `[a,b,e,g]`

в) `[a,c,g]`

Критерии за оценяване

а) Коректен и обоснован отговор се оценява с 5 точки.

- Коректен отговор без обосновка се оценява с 3 точки.

б) Коректен отговор се оценява с 3 точки.

в) Коректен отговор се оценява с 2 точки.

Задача 8. В пространството спрямо ортонормирана координатна система $K = O\vec{e}_1\vec{e}_2\vec{e}_3$ са дадени правите

$$m : \left| \begin{array}{l} x - y + 2z - 1 = 0 \\ x - z \quad = 0 \end{array} \right. , \qquad g : \left\{ \begin{array}{l} x = \quad 2v \\ y = -2 \\ z = \ 2 - v \end{array} \right. .$$

и равнината

$$\alpha : 5x + 2y + z - 2 = 0.$$

Да се намери тази трансверзала на правите m и g, която е перпендикулярна на равнината α.

Примерно решение

Намираме параметрично представяне на правата m:

$$m : \begin{cases} x = u \\ y = -1 + 3u \\ z = u \end{cases}.$$

Произволна точка M от правата m има координати:

$$M(u, -1 + 3u, u).$$

Произволна точка G от правата g има координати:

$$G(2v, -2, 2 - v).$$

Следователно:

$$\overrightarrow{MG} = (-u + 2v, -1 - 3u, 2 - u - v).$$

Понеже искаме трансверзалата $t \perp \alpha \implies \overrightarrow{MG} \parallel \vec{n}(5, 2, 1) \perp \alpha \Leftrightarrow \overrightarrow{MG} = k\vec{n}$. Получаваме:

$$(-u + 2v, -1 - 3u, 2 - u - v) = k(5, 2, 1).$$

$$\left| \begin{array}{l} -u + 2v = 5k \\ -1 - 3u = 2k \\ 2 - u - v = k \end{array} \right.$$

Решението на тази система линейни уравнения е

$$u = -1, \quad v = 2, \quad k = 1$$

За оста t намираме:

$$t : \begin{cases} \ni M(-1, -4, -1), \ u = -1, \\ \ni G(4, -2, 0), \ v = 2, \\ \parallel \overrightarrow{MG}(5, 2, 1) \parallel n \end{cases} \implies t : \begin{cases} x = -1 + 5s \\ y = -4 + 2s \\ z = -1 + s \end{cases}.$$

Критерии за оценяване

- За намиране параметрично представяне на правата $m \to 2$ т.
- За намиране на вектор $\overrightarrow{MG} \to 2$ т.
- За получаване на системата $\overrightarrow{MG} = k\vec{n} \to 2$ т.
- За решаване на системата $\overrightarrow{MG} = k\vec{n} \to 2$ т.
- За намиране на трансверзалата $\to 2$ т.

Чернова