Задачи от контролни, домашни, малки контролни, текущ контрол, примерни задачи и задачи за самоподготовка

Задача 1. Нека е дадена биекция c:char -> int. Кодиране на символния низ $a_1,...,a_k$ наричаме редицата от стойности от тип int $n_1,...,n_k$, където $n_i = c(a_i)$. Да се създаде клас Encoding, която реализира съответствието "символ – код" за единичен символ.

Да се реализира на клас Досие (Dossier), който съдържа КОДИРАН текст с произволна дължина. Досиетата съдържат и информация за кодиране (декодиране) на текста. Информацията представлява динамичен масив от съответствия "символ – код", за всички участващи в текста символи. Да се реализират:

- канонично представяне на клас Dossier
- конструктор с параметри - текст(char*) и таблица за кодиране (Encoding*), който кодира текста (добавете допълнителни параметри, ако са необходими)
- булев оператор &&, който проверява дали кодовете на две досиета си противоречат (т.е. дали дават различно кодиране за един и същи символ)
- оператор за събиране (+), който събира два кодирани текста от досието, чиито кодове не си противоречат. Ако кодовете си противоречат, се създава празно досие
- printText, който извежда оригиналния текст на екрана

Задача 2. Да се дефинира клас Relation, който съдържа два обекта от тип int, наречени subject и object, и низ с произволна дължина relation, описващ връзката между тези обекти.

Пример:
Relation r1(2,6,"is smaller than"),r2(6,3,"is divisible by");

За класа да се реализират голямата четворка и оператор за изход operator<<

Пример:
r1.print(): 2 is smaller than 6.

За класа Relation реализирайте и оператор за композиция * по следния начин.

Ако r = r1 * r2, то r.subject = r1.subject, r.object = r2.object,

r.action = r1.action r1.object ", which is" r2.action

Пример:
(r1*r2).print(): 2 is smaller than 6, which is divisible by 3

Композицията се допуска само ако r1.object == r2.subject, в противен случай резултатът е r1.

Упътване: Обръщението към ф-ята itoa(val,str,10) записва числото val в низа сочен от str.

<u>Вариант 1</u>

Задача 1. Реализирайте клас `Vector2D`, описващ вектор в евклидова равнина.
Реализирайте и:

- конструктор по подразбиране, инициализиращ го като нулев вектор.

- оператор, реализиращ умножение на вектора с число.
- оператор, реализиращ сума на два вектора. Да се реализира + и +=.
- оператор, реализираш скаларно произведение на вектора с друг подаден вектор.
- оператори == и !=.

Задача 2. Да се реализира клас, представящ динамичен масив от точки в равнината.
Всяка точка се реализира с двете си реални координати (X и Y). За класа реализирайте подходящи конструктори; метод за транслация на точка от масива с подаден индекс на подаден като двойка реални числа вектор; метод за определяне на броя елементи на масива. Да се реализира външен за класа метод, който по два масива с точки връща нов масив съдържащ елементите от двата дадени.
* Да се реализира добавяне и премахване на елемент по указан индекс.

Задача 3. Отстранете грешката в програмата. Намерете резултата от изпълнението на поправената програма.

```
#include <iostream.h>
class A {
  public:
   A(int a = 2);
   void f();
   void g();
   int h() const;
   void k() const;
  private:
   int n;
};
A::A(int a) {
  std::cout << "A(int)\n";
  n = a-1;
}
void A::f() {
  std::cout << "f()\n";
  n++;
}
void A::g() {
  std::cout << "g()\n";
  f();
  n = 2*h() + 3;
  f();
}
int A::h() const {
  std::cout << "h()\n";
  n--;
  return n;
}
void A::k() const {
  std::cout << "k()\n";
  std::cout << n << std::endl;
}
```

```
int main() {
  A a;
  A b(5);
  A c = b;
  A d = A(13);
  a.f(); b.g(); c.f(); d.g();
  d.k();
  A e(a.h()+b.h()+c.h());
  e.k();
  return 0;
}
```

Вариант 2

Задача 1. Реализирайте клас Circle, описващ кръг в равнината.

Реализирайте и:

- конструктор по подразбиране, инициализиращ го като кръг с център началото на координатната система и радиус 1.
- оператор за събиране на два кръга, който създава нов кръг с радиус сумата на радиусите на двата дадени и център с координати сума от координатите на центровете на дадените кръгове. Да се реализира и съответния оператор +=.
- метод bool isIn(double, double), проверяващ дали дадена точка лежи в кръга.
- оператори == и !=.
- операторите за сравнение на два кръга (< и >). Един кръг е по-малък от друг ако се съдържа изцяло в него.

Задача 2. Да се реализира клас Vector.

Класът трябва да съдържа динамичен масив от цели числа. Реализирайте подходящи конструктори; методи за извличане на стойност от масива по нейния индекс и промяна на стойността на елемент с определен индекс; метод за определяне на броя елементи на масива и външен за класа метод, който по два вектора връща конкатенацията им.

*Да се реализира добавяне и премахване на елемент по индекс.

Задача 3. Отбележете и обяснете грешките в програмата. Поправете ги, така че да се получи работеща програма. Намерете резултата от изпълнението й.

```
#include <iostream.h>
class A {
  public:
   A(int, int = 0);
   void print();
   int f_Aa();
   int f_Ab();
  private:
   int a, b;
};
A::A(int x, int y) {
  a = x;
```

```
  b = y;
}
void A::print() {
  std::cout << a << " " << b << std::endl;
}
int A::f_Aa() const {
  return a;
}
int A::f_Ab() const {
  return b;
}
class B {
  public:
   B(double = 3, A);
   void print() const;
   double f_Bb();
   A f_Ba() const;
  private:
   A a;
   double b;
};
B::B(double d, A e) {
  a = e;
  b = d;
}
void B::print() const {
  a.print();
  std::cout << b << std::endl;
}
double B::f_Bb() const {
  std::cout << a.f_Aa() << " " << a.f_Ab() << std::endl;
  return b;
}
A B::f_Ba() {
  return a;
}
int main() {
  A a(4), b;
  a.print();
  std::cout << b.a << " " << b.b << std::endl;
  B b1(4.5, a);
  b1.print();
  B b2(2.5);
  b2.print();
  return 0;
}
```

Вариант 3

Задача 1. Реализирайте клас ArithmeticProgression, описващ аритметична прогресия. Реализирайте:

- Конструктор по подразбиране, инициализиращ прогресията като съвпадаща с множеството на естествените числа.
- Оператор за събиране на прогресии. Под сума на прогресии разбираме прогресия, представляваща тяхната поелементна сума. Да се реализира и съответния оператор +=.
- Оператор за изваждане на прогресии. Под разлика на прогресии разбираме прогресия, представляваща тяхната поелементна разлика. Да се реализира и съответния оператор -=.
- Оператор за добавяне на цяло число без знак към прогресия, който създава прогресия, получена от началната с прескачане на дадения брой елементи от началото (т.е. да се измества началото).

Задача 2. Да се реализира клас, представящ динамичен масив от точки в пространството.
Всяка точка се реализира с трите си реални координати (X, Y и Z). За класа реализирайте подходящи конструктори; метод за транслация на точка от масива с подаден индекс на подаден като тройка реални числа вектор; метод за определяне на дължината на масива. Да се реализира и външен за класа метод, който по два масива с точки връща нов масив съдържащ елементите от двата дадени.
***** Да се реализира добавяне и премахване на елемент по индекс.

Задача 3. Намерете, обяснете и поправете грешките в програмата. Какъв е резултатът от изпълнението на поправената програмата?

```
#include <iostream.h>
#include <string.h>
#include <assert.h>
class first {
  public:
   first(char* ="first", int = 0);
   ~first();
   first(const first&);
   first& operator=(const first&);
    void print() const;
  private:
   char* str;
   int x;
};
first::first(char* s, int y) {
  std::cout  << "first(" << s << "," << y << ")\n";
  str = new char[strlen(s)+1];
  assert(str != 0);
  strcpy(str, s);
  x = y;
}
first::~first() {
  std::cout << "~first()\n";
  delete str;
}
first::first(const first& s) {
  std::cout << "first(const s)\n";
  delete str;
```

```
  str = new char[strlen(s.str)+1];
  assert(str != 0);
  strcpy(str, s.str);
  x = s.x;
}
first& first::operator=(const first& s) {
  std::cout << "first::operator=()\n";
  if(this != &s) {
    str = new char[strlen(s.str)+1];
    assert(str != 0);
    strcpy(str, s.str);
    x = s.x;
  }
  return *this;
}
void first::print() const {
  std::cout << str << " " << x << std::endl;
}
int main() {
  first a("***"), b("+++", 12), c(b);
  a.print(); b.print();
  c.print();
  c = a;
  c.print();
  first* p = new first("---", -5);
  assert(p != 0);
  p.print();
  delete p;
  return 0;
}
```

Задача 1.

Първа част: Речник (3 т.)

Да се реализира шаблон на клас `Dictionary` с два типови параметъра K и V. Класът да представя речник, който съдържа редица от ключове от тип K и на всеки ключ съпоставя по една стойност от тип V. Едно примерно представяне би могло да бъде с два масива с една и съща дължина — един с елементи от тип K и един с елементи от тип V. За типа K и V може да се приеме, че са налични:

- голяма четворка
- операция за изход <<
- операции за сравнение == и !=
- за типа K допълнително са налични операции за наредба <, <=, >, >=

За класа да се реализират:

- голяма четворка
- добавяне на ключ и съответна на него стойност
- търсене на стойност по ключ
- извежда речника на стандартния вход подреден по ключ

- разширяване на речника при изчерпване на капацитета му

Втора част: Книга с контакти (3 т.)
Да се създаде клас `Contact`, който съдържа име, телефонен номер (до 10 цифрен), и идентификатор с подходящи конструктор, селектори и мутатори.
Да се създаде клас `ContactBook`, който представя бележник с контакти като набор от три речника, чиито ключове са съответно име, телефонен номер и идентификатор, а стойностите са указатели към контакти (`Contact*`).
За този клас да се реализират:

- добавяне на контакт
- намиране на контакт по даден критерий (име, телефонен номер или идентификатор). *Внимание: върнатият контакт да няма възможност да бъде променян!*
- извеждане на книгата с контакти подредена по даден критерий (име, телефонен номер или идентификатор)
- премахване на контакт

Упътване 1: Може да се използва наготово класът от първото домашно `String` или стандартния клас`std::string`.
Упътване 2: Може да се предефинира операцията за изход **<<** за тип `Contact*`, за да работи правилно извеждането на речник.
Забележка: за тази част от задачата не се изискват никакви усилия по управление на динамичната памет.

Трета част: Динамична памет (2 т.)
За класа `ContactBook` да се реализира допълнителна логика, която да се грижи правилно за управлението на подаваните контакти. Считаме, че `ContactBook` "притежава" контактите, т.е. при добавяне на контакт създава негово копие, а при изтриване унищожава контакта.

Задача 2. Формули (12 т.)

Част първа: Йерархия от класове (5 т.)
Да се дефинира абстрактен базов клас `Formula`, дефиниращ операцията `double value() const`. Класът да представя формула с неуточнено представяне, чиято стойност може да бъде изчислена чрез метода `value`. Да се дефинира също и операция `void print() const` за извеждане на явния вид на формула на стандартния изход.
а) Да се дефинира производен клас `Constant`, който описва дробни константи. Обекти от клас `Constant` се конструират със стойността на съответната константа.
Пример:

```
Constant a (0), b (1);
std::cout << b.value();    // извежда 1
b.print();             // също извежда 1
```

б) Да се дефинира клас `BinaryOperation`, описващ приложение на двуместна аритметична операция върху други две формули. Допустимите операции са `+`, `-`, `*`, `/`, както и операциите за сравнение `<` и `=`, като последните имат стойност 1 при удовлетворено (не)равенство и стойност 0 иначе. Обекти от клас `BinaryOperation` се конструират със символ, описващ операцията и два указателя към формули, задаващи операндите.
Пример:

```
BinaryOperation test ('*',
           new Constant (2),
           new BinaryOperation ('+',
                     new Constant (1),
                     new Constant (3));
std::cout << test.value(); // извежда 8, т.е. стойността на 2 * (1
+ 3)
test.print();          // извежда "( 2 * ( 1 + 3 ) )"
```

в) Да се дефинира клас `Read`, описващ стойност, въведена от потребителя.
Пример:

```
BinaryOperation test ('*', new Constant (2), new Read);
std::cout << test.value();  // изисква от потребителя да въведе
                        // число и го връща умножено по 2
test.print();           // извежда "( 2 * Read )"
```

г) Да се дефинира клас `Conditional`, описващ формула, която зависи от дадено условие. Обекти от клас `Conditional` се конструират с три указателя към `Formula` — условие ("if"), формула при удовлтеворяване на условието ("then") и формула при неудовлетворяване на условието ("else").
Пример:

```
Conditional test (new BinaryOperation ('<', new Read,
                                            new Constant (5)),
            new Constant (1),
           new Constant (2));
std::cout << test.value(); // извежда 1, ако поребителят въведе
                      // число, по-малко от 5
                      // и 2 в противен случай
test.print(); // извежда "if ( Read < 5 ) then 1 else 2"
```

Забележка: за тази част от задачата не се изискват никакви усилия по управление на динамичната памет.

Втора част: Управление на динамичната памет (3 т.)
За така изградената йерархия да се дефинират необходимите специални методи (конструктори, деструктори, оператор за присвояване) така, че динамичната памет да се управлява коректно. Следните операции трябва да могат да протичат без проблеми, свързвани със споделяне на памет, изтичане на памет или други неправилни манипулации на динамичната памет. След приключването на съответния им блок, заетата динамична памет трябва да бъде освободена.

```
Constant c1 (1);
Constant* c2 = new Constant (2);
BinaryOperation f1 ('+', &c1, c2);
BinaryOperation f2 = f1;    // копиране
f2 = f1;                    // присвояване
delete c2;
```

Упътване 1: В класа `Formula` да се добави виртуален метод `Formula* clone() const`. Всеки клас да предефинира метода така, че обектите да могат да построяват свои идентични копия.
Упътване 2: Да се променят класовете така `BinaryOperation` и `Conditional`, така че да "притежават" формулите, които ги съставят. В частност конструкторите да бъдат построени така, че да запомнят в член-данните адресите на копия на параметрите си, а не оригинално подадените адреси. Дестуркторите, от своя страна, да изтриват копията, записани в съответните член данни.

Трета част: Десериализация (4 т.)
Показаните в примерите резултати от метода `print` за всеки клас от описаната йерархия демонстрират едно еднозначно представяне на всички формули, представими с нея. Да се дефинира рекурсивна функция `Formula* readFormula()`, която по въведено от стандартния вход правилно построено представяне на формула построява съответното ѝ представяне чрез класовете от изградената йерархия.
Пример:

```
Formula* f = readFormula(); //от стандартния вход се въвежда
                            // “if ( Read < 5 ) then 1 else 2”
std::cout << f->value(); // извежда 1, ако поребителят въведе
                    // число, по-малко от 5
                      // и 2 в противен случай
delete f;
```

Забележка: По ваше желание можете да изберете друго, по-просто представяне на формула.

Задача 1 (Поправителен изпит 2018-2019)
Наближава седмицата на технологичните иновации, където ще бъде представен прототип на нов **куфар**. Той има цвят, на него са написани име, адрес и телефон на притежателя му, но най-важното е, че в него могат да се побират определен брой **дрехи**, като **панталони**, **рокли** и **костюми**. За всички дрехи се знае, че имат свой цвят, размер и цена. Панталоните си имат дължина на крачола, а роклите - дължина на полата.

Важна част от функционалността е куфарът да казва дали има място за още една дреха; трябва да може да добавя такава (разбира се, ако има място); да изважда дадена дреха от определено място и да разбере дали има (и ако да – на кои места) дреха с определен цвят и/или тип (например дали има черен панталон, дали има някаква червена дреха, дали има някакви рокли т.н.). Трябва да може да записва в текстов поток и пълна информация за куфара и всичко в него, включително колко струва съответният модел. От така записаната информация трябва и да може да се поръча(създаде) еквивалентен нов куфар. Прототипът на новия куфар впечатлява с това, че той може да се клонира, но не може да сменя притежателя си. Колкото и негови копия да бъдат направени, те ще имат същите данни написани на тях. Новият куфар ще има **отделни** дрехи, които са същите като тези в оригиналния.

Реализирайте подходящи класове, организирани в правилна йерархия, които да описват ситуацията на прототипа на куфара, който ще бъде представен на изложението. Покажете използването на тези класове в подходяща кратка програма.

В решението на задачата може да използвате колекциите от STL (list, vector и т.н.), но употребата им трябва да е коректна. В решението **не можете** да използвате класа std::string.

(2019) Писмен изпит по Обектно-ориентирано програмиране.

** Всички операции, свързани със споделяне и освобождаване на памет, трябва да могат да протичат без проблеми.*

Задача 1. Да се напише програма, която позволява създаване и работа с множество от ястия.
За целта да се създаде клас ястие (**Dish**).
Всяко ястие (**Dish**) се характеризира със:

- ★ Списък от продукти (**ingredientsNames**) - всеки продукт се задава чрез неговото име с произволна дължина. Списъкът съдържа максимум 100 продукта.
- ★ Време за приготвяне (**cookTime**) - измерва се в минути.

Да се реализира функция **print**, която извежда информацията за едно ястие на екрана.

За целта на нашата задача различаваме два основни типа ястия:

- ★ Ястие с месо (**MeatDish**) - съдържа допълнителна член-данна месо (**meat)**, която съдържа името на основния протеин в ястието.
  Член-данната **meat** може да приема една от следните стойности: **“chicken”, “beef”, “pork”, “duck”.**
  Да се реализира функция **print**, която извежда собствените и наследените данни за ястието.

- ★ Ястие с морска храна (**SeafoodDish**) - съдържа допълнителна член-данна морска храна (**seafood)**, която съдържа името на основния протеин в ястието. Член-данната **seafood** може да бъде име на риба или друг морски дар с произволна дължина.
  Валидация на стойността на член-данната не е необходима.
  Да се реализира функция **print**, която извежда собствените и наследените данни за ястието.

Да се реализира клас **SurfAndTurf**, представляващ ястие, което комбинира морски дарове и месо.
Да се напише функция **print**, която извежда **<u>цялата</u>** информация за ястието.

Да се реализира главна функция (**main**), която:

- ★ Създава обект от тип **SurfAndTurf**,
- ★ Създава указател към **Dish** и го свързва с динамично създаден обект **MeatDish**. (*Използвайте оператор **new***)
- ★ Освобождава заделената памет, където е нужно.

Задача 2. Да се напише шаблон на абстрактен клас множество (**Set**). Класът описва следните операции:

- ★ **bool member(<тип> x) const -** проверява дали x е елемент на множество**,**
- ★ **оператор [] -** връща i-тия елемент на множество или грешка,
- ★ **int length () const -** връща броя на елементите в множеството**.**

Да се реализира клас множество с точно 1 елемент (**Singleton**).
Да се създадат подходящи конструктори.

Да се реализира клас множество от максимум n елемента (**ArraySet**). Броят на елементите се задава при създаването на обекта. Класът да поддържа следните операции:

- ★ Добавяне на елемент към множеството (**addItem**) - ако капацитетът е изчерпан или този елемент се съдържа в множеството, методът връща стойност лъжа.

Да се реализира клас Меню (**Menu**).

Класът включва:

- ★ Произволно множество (**Singleton** или **ArraySet**) от ястия (**dishes**).
  Всяко ястие в множеството може да отговаря на произволен тип от задача 1.
- ★ Съответстващо множество от цени (**prices**).

Да се реализира конструктор с два параметъра от подходящ тип.
Да се реализира метод **print**, който извежда информация за всяко ястие и съответстващата му цена.
Приемете, че данните, които се подават са валидни и съвпадат по размер и тип.