**Контролно 1.**  
  
Задача 1. Открийте, обяснете и коригирайте грешките в следните програмни фрагменти:  
a) class Example{

public: Решение:

Example(int y=10)

{data = y;}

int getIncrementData() const 1.

{ return ++data;}

static int getCount() 2.

{ cout << "Data=" << data << endl;

return count;}

private:

int data;

static int count;

} 3.

b) char \*string;

string = new char[20];

free(string); 4.  
  
  
Решение:  
1. Не трябва да има const защото в тялото има промяна на данната data   
2. Не трябва да има static, защото ако има data не може да се използва така   
3. Трябва да има ; накрая   
4. На горния ред е заделена динамично памет посредством new. За да се освободи тази памет трябва да се използва delete, не free   
  
  
  
Задача 2. Да се открият, обяснят и коригират грешките в следния фрагмент от програма  
#include<iostream.h>  
class Time{  
public:  
Time()  
{hour=minute=second=0;)  
private:  
int hour;  
int minute;  
int second;  
};  
void main()  
{Time t;  
t.hour= 7; 1.   
cout<<t.minute; 2.  
}  
  
Решение:   
1,2. hour и minute са private т.е. не са достъпни пряко (т.е. както е в програмата), те могат да бъдат използвани само във функции.   
  
Задача 3. Какво прави (извежда) тази програма?  
#include <iostream.h>  
class CreateAndDestroy{  
public:  
CreateAndDestroy(int);  
~CreateAndDestroy();  
private:  
int data;  
};  
CreateAndDestroy::CreateAndDestroy (int value)

{ data = value; cout << “Обект “ << data << “ конструктор”; }

CreateAndDestroy::~CreateAndDestroy()

{ cout << “Обект “ << data << “ деструктор ” << endl; }

void Create ();

CreateAndDestroy first (1);

void main ()  
{ cout << “ глобален създаден преди main” << endl;

CreateAndDestroy second (2);

cout << “локален автоматичен в main” << endl;

static CreateAndDestroy third (3);

cout << “локален статичен в main” << endl;

Create ();

CreateAndDestroy fourth (4);

cout << “локален автоматичен в main” << endl;  
}

void Create ()

{ CreateAndDestroy fifth (5);

cout << “локален автоматичен в Create” << endl;

static CreateAndDestroy sixth (6);

cout << “локален статичен в Create” << endl;

CreateAndDestroy seventh (7);

cout << “локален автоматичен в Create” << endl;

}  
  
Решение:  
Принципът тук е следния:  
1. Това, което е създадено първо се унищожава последно  
2. Трябва да се внимава за областта на действие (т.е. кое до кога действа, може да се каже, че { } показват именно това)- ето орязан пример  
ob1  
{  
 ob2;  
 ob3;  
}  
{  
ob4;  
}  
първо ще се създаде ob1 после се създаде ob2, после ob3 след което ще се унищожи ob3 и след него ob2, после ще се създаде ob4 след това и той ще се унищожи и накрая ще се унищожи ob1.  
Т.е.1. се прилага за всяка област на действие   
3. За static обектите тези неща не важат напълно- static обект ще се унищожи накрая на програмата, но преди обектите, които са глобални за main (т.е. създадени преди main), също за различните static обекти се прилага 1. (в задачата това са обект 3 и обект 6)  
Обект 1 конструктор глобален създаден преди main  
Обект 2 конструктор локален автоматичен в main  
Обект 3 конструктор локален статичен в main  
Обект 5 конструктор локален автоматичен в Create  
Обект 6 конструктор локален статичен в Create  
Обект 7 конструктор локален автоматичен в Create  
Обект 7 деструктор  
Обект 5 деструктор  
Обект 4 конструктор локален автоматичен в main  
Обект 4 деструктор  
Обект 2 деструктор  
Обект 6 деструктор  
Обект 3 деструктор  
Обект 1 деструктор  
   
  
  
Задача 4. Да се открие грешката, да се обясни и коригира в следния клас Increment  
class Increment{

public:   
 Increment( int c=0, int i=1);   
 void addIncrement() {count+=increment;}   
 private:   
 int count;   
 const int increment;   
 }   
 Increment::Increment(int c, int i) 1.  
 { count = c;   
 increment = i;   
 }  
  
Решение:  
1. Тук проблемът е, че имаме cons int increment- const създава проблеми  
Трябва да се напише:  
Increment::Increment(int c, int i) : increment (i)   
 { count = c;   
 }   
понеже това е единствения начин да се зададе стойност на const данна  
  
Задача 5. Какво прави (извежда) тази програма?  
#include <iostream.h>  
class object {  
static int a;  
int b;  
public:  
void set(int I, int j){a=I,b=j;}  
void show();  
}  
int object::a=0;  
void object::show()  
{cout<<”This is static a: ”<<a;  
cout<<”This is non static b: ”<<b;  
cout<<”\n”;};  
int main()  
{object x,y;  
x.set(1,1);  
x.show();  
y.set(2,2);  
y.show();  
x.show(); може би y  
return 0;}  
  
Тази задача е от тези, които не знам колко добре съм успял да разчета, но нещата би трябвало да са така  
Решение:  
Тук няма какво да се обяснява, идеята на тази задача е да уплаши хората със static данната а  
  
This is static a: 1 This is non static b: 1  
This is static a: 2 This is non static b: 2  
This is static a: 2 This is non static b: 1 🡨 обърнете внимание на този ред (това е защото а е static)  
  
Ако случайно е пишело y то последния ред ще е: This is static a: 2 This is non static b: 2  
  
  
Задача 6. Какво прави (извежда) тази програма?  
#include <iostream.h>  
class Sample  
{private:  
int m\_nID;  
public:  
Sample(int nID)  
{cout<<”Constructing Sample”<<nID<<endl;  
m\_nID=nID;}  
~Sample()  
{cout<<”Destrcting Sample”<<m\_nID<<endl;}  
int GetID(){return m\_nID;}  
};  
int main()  
{ Sample cSample(1);  
 cout<<cSample.GetID()<<endl;  
 Sample \*pSample=new Sample(2); $  
cout<<pSample->GetID()<<endl;  
delete pSample; ^  
return 0;}

Тук също не съм абсолютно сигурен дали съм разчел всичко както трябва, но отново нещата би трябвало да са така  
  
Решение:  
Constructing Sample1  
1  
Constructing Sample2 редът $ е еквивалентен на създаване на обект и се вика констр.  
2  
Destructing Sample2 редът ^ еквивалентен на унищожаване на обект и се вика дестр.  
Destructing Sample 1  
  
  
  
Задача 7. Какво прави (извежда) тази програма?  
#include <iostream.h>  
class myclass{  
public:  
int who;  
myclass(int id);  
~myclass();  
}glob\_ob1(1), glob\_ob2(2); 🡨 Тук тези обекти са създадени още след   
  
 дефиницията на класа и са глобални за main   
  
  
myclass::myclass(int id)  
{cout<<”Constructing ”<<id<<”\n”;  
who=id;}  
myclass::~myclass()  
{cout<<”Destructing ”<<who<<”\n”;}  
  
int main()  
{myclass local\_ob3(3);  
cout<<”This will not be the first one displayed”<<”\n”;  
myclass local\_ob4(4);  
return 0;}  
  
За тази задача съм най-несигурен- идеята е тази, но различните неща може да не се казват точно така и обектите може да са инициализирани с други цифри  
  
Решение:  
Ситуацията е подобна на ситуацията от задача 3.  
  
Constructing 1  
Constructing 2  
Constructing 3  
This will not be the first one displayed  
Constructing 4  
Destructing 4  
Destructing 3  
Destructing 2  
Destructing 1

Задача 8. Какво прави (извежда) тази програма?  
#include <iostream.h>  
class myclass{  
int i;  
public:  
myclass (int x);   
~myclass();

void set\_i(int x) {i=x;}   
int get\_i(){return i;}  
};  
  
  
myclass::myclass(int x)  
{  
i=x;  
cout<<”Constructing ”<<i<<”\n”;  
}  
myclass::~myclass()  
{cout<<”Destructing”<<i<<”\n”;}  
  
void f(myclass ob);  
  
int main()  
{myclass o(1);  
f(o);  
cout<<”This is in main”;  
cout<<o.get\_i()<<”\n”;  
return 0;}  
  
void f(myclass ob)  
{ob.set\_i(2);  
cout<<”This is local….”<<ob.get\_i(); 🡨 Тук имаше нещо, което скриваше написаното  
cout<<”\n”;}  
  
Решение:  
В тази задача има следния трик  
  
Constructing 1  
This is local….  
Destrcting 2  
This is in main 1  
Destructing 1  
  
Как става така, че имаме един обект, а успяхме да го унищожим 2 пъти, че и изкарахме неща на екрана?  
Когато нещо се подава на фунция в нашия случай функцията f ( f(o) подаваме обекта о като аргумент на f) всъщност се подава копие на съответното нещо. Така всъщност имаме два обекта, като копието се унищожава на края на областта си на действие.  
Хубаво обаче след като се появява втори обект защо не излиза Constructing…. Отговора изисква по-задълбочени познания и е следния- ние в програмата специално сме написали обикновен конструктор, но принципно по подразбиране си имаме и готов конструктор за копиране, който се използва специално при копиране. Тъй като не сме го писали специално ние (нищо не ни пречи да си го напишем), то той не изкарва някакво съобщение когато се изпълнява.  
  
  
  
**Контролно 2.**Задача 1.  
Решение:  
Да кажем, че клас Б е наследник на клас А. И двата класа имат конструктори. Конструкторите служат за задаване на стойности на данните. Б да е наследник на А означава да има цялата информация на А. Затова когато се създава обект от клас Б се случват следните неща:  
1. Първо се извиква конструктора на класа А и се задават стойности на данните на обекта от клас А  
2. После се изпълнява конструктора на Б и се задават стойностти на данните на обекта от клас Б.   
Така обекта от Б има все едно има в себе си обект от клас А, с бонус собствени дани и информация (принципно се предават и функции, но това в момента няма значение)   
  
В задачата горенаписаното се случва повече от веднъж- създаваме в main обект от клас С и в скобичките имаме данни, които се предават. С е наследник на В значи се вика конструктора на В и някаква част от данните му се предават:  
C(int nValue, double dValue,char ch Value): B(nValue,dValue)  
Подчертаното са тези данни, които се предават на конструктора на В. В обаче е наследник на А и операцията се повтаря- преди да се изпълни конструктора на В( т.е. да се изпълнят cout<<”B”<<…. И т.н.) се вика конструктора на А и му се дава nValue.  
След като са подадени данните се изпълнява първо конструктора на А, после този на В и накрая този на С  
  
A:5  
B:4.3  
C:R  
  
Задача 2.  
Решение:  
PoweredDevice: 3  
Scanner: 1  
Printer: 2  
  
Тук се случват почти същите неща като в задача 1 с изключение на това, че е налице множествено наследяване- ако стриктно се спазват действията от задача 1 то би трябвало няколко пъти да се появи PoweredDevice. Тук обаче имаме думичката virtual в  
class Scanner:virtual public PoweredDevice и в class Printer:virtual public PoweredDevice,  
която предотвратява множественото наследяване т.е. PoweredDevice: 3 ще излезе само веднъж  
  
Задача 3.  
Тук са изпълнени всички неща от задача 3 от контролно 1 с няколко допълнения от наследяването:  
4. При създаването на обект на клас( да го наречем Б), който е наследник на друг клас(да го наречем А) първо се изпълнява конструктора на А, после конструктора на Б.  
5. При унищожаването на такъв обект първо се изпълнява деструктора на Б, после деструктора на А  
  
Решение:  
Конструктор на Point:[1.1, 2.2]  
Конструктор на Point:[1.1, 2.2]  
Конструктор на Point:[7.2, 2.9]  
Конструктор на Circle:4.5[7.2,2.9]  
Конструктор на Point:[5, 5]  
Конструктор на Circle:10[5, 5]  
Деструктор на Circle:10[5, 5]  
Деструктор на Point:[5, 5]  
Деструктор на Circle:4.5[7.2, 2.9]  
Деструктор на Point:[7.2, 2.9]  
  
Задача 4.  
С тази задача ще се занимавам друг път ☺  
  
Задача 5.  
  
Решение:  
result=drol (с едно l)  
  
Функцията list създава последователен списък, като във всеки елемент от него се съдържа по една буквичка от думата droll (с две l)  
Функцията print извиква рекурсивно себе си и ако човек разбира от последователни списъци и рекурсия би видял, че като даваме в main на print указателя first->next->next (когато го подаваме first сочи края на списъка, ние подаваме first->next->next, с което изпускаме второто l) first, който е във функцията, стига до нулев байт, вика result= и после се викат останалите букви  
(няма по-лесен начин да се обясни, но има доста по-детайлен)  
  
Задача 6.  
Решение:  
1. If(ptr->nomer==key) прави се проверка дали това е търсения елемент  
цикъла while трябва да се прекрати ако това наистина е елемента  
затова пишем break;  
2. Ако това не е елеметът, който търсим трябва да продължим напред и затова пишем  
ptr=ptr->next  
  
Задача 7.  
Решение:  
a) Трябва да се провери дали със *следващия* елемент ще се надхвърли максималния брой допустими елементо. Затова правим проверка със следващия елемент. Самата проверка е направена, ние просто трябва да дадем следващия елемент и пишем   
t++  
b) Тук ние трябва да направим проверката дали стекът е празен преди да се извършат операциите по-доло. ПРоверката представлява:  
if(s->t==0) exit(1);  
  
Задача 8.  
Решение:  
Тук се прави проверка за препълване над многоточието и някакво присвояване след многоточието. Това което ние трябва да направим е да подадем следващия елемент за да може да се извърши присвояването ( все пак сме преминали успешно проверката) т.е. :  
q->r=(q->r+1)%M