Федеральное государственное бюджетное

образовательное учреждение высшего образования

«Саратовский государственный технический университет

имени Гагарина Ю.А.»

Институт прикладных информационных

технологий и коммуникаций

Кафедра Информационная безопасность

автоматизированных систем

Направление подготовки 10.03.01 Информационная безопасность

Расчётно-графическая работа

по дисциплине «Языки программирования»

«Бухгалтерская программа»

Выполнила: студентка 1 курса

учебной группы с-ИБС-11

очной формы обучения

Каталова Евгения Андреевна

Проверил: ассистент каф. ИБС

Романчук С. П.

Саратов 2020

Содержание

Аннотация 3ст

Введение 4ст

Язык разработки 5ст

Структура данных 10ст

Заключение 12ст

Приложение 13ст

Аннотация

Данная расчетно - графическая работа является реализацией бухгалтерской программы, написанной на языке С++ .

Описание программы

Планируется, что программа будет содержать 4 класса:

1. Кассовый ордер
2. Приходный кассовый ордер
3. Расходный кассовый ордер
4. Контрагент

Кассовый ордер будет являться базовым классом, от него наследуются классы приходный и расходный кассовые ордера. Класс контрагент создаются отдельно и является полем 2 и 3 класса.

Будут реализованы методы для просмотра истории операций и поиск операций по различным характеристикам.

Реализовать планируется с помощью массива, элемент которого является объектом класса

Будет создан файл, куда загрузятся истории операций.

Введение

Тема работы:

«Бухгалтерская программа».

Цель работы:

Разработать приложение эмулирующий функционал бухгалтерской программы.

Задачи работы:

1. Создать расходный и приходный кассовые ордера.
2. Разработать метод для просмотра баланса на определенную дату.
3. Разработать метод для просмотра историй и поиска операций по различным характеристикам.
4. Загрузить и сохранить истории операций в файл.

Язык разработки.

**C++** — компилируемый, статически типизированный язык программирования общего назначения.

Поддерживает такие парадигмы программирования как процедурное программирование, объектно-ориентированное программирование, обобщённое программирование, обеспечивает модульность, раздельную компиляцию, обработку исключений, абстракцию данных, объявление типов (классов) объектов, виртуальные функции. Стандартная библиотека включает, в том числе, общеупотребительные контейнеры и алгоритмы. C++ сочетает свойства как высокоуровневых, так и низкоуровневых языков. В сравнении с его предшественником — языком C, — наибольшее внимание уделено поддержке объектно-ориентированного и обобщённого программирования.

C++ широко используется для разработки программного обеспечения, являясь одним из самых популярных языков программирования. Область его применения включает создание операционных систем, разнообразных прикладных программ, драйверов устройств, приложений для встраиваемых систем, высокопроизводительных серверов, а также развлекательных приложений. Существует множество реализаций языка C++, как бесплатных, так и коммерческих и для различных платформ.

Необъектно-ориентированные особенности

Типы:

Символьные: char, wchar\_t (char16\_t и char32\_t, в стандарте C++11).

Целочисленные знаковые: signed char, short int, int, long int (и long long int, в стандарте C++11).

Целочисленные беззнаковые: unsigned char, unsigned short int, unsigned int, unsigned long int(и unsigned long long int, в стандарте C++11).

С плавающей точкой: float, double, long double.

Логический: bool.

Операции сравнения возвращают тип bool. Выражения в скобках после if, while приводятся к типу bool.

Функции могут принимать аргументы по ссылке. Функции могут возвращать результат по ссылке. Cсылки сходны с указателями, со следующими особенностями: перед использованием ссылка должна быть инициализирована; ссылка всегда указывает на один и тот же адрес; в выражении ссылка обозначает непосредственно тот объект или ту функцию, на которую она указывает, обращение же к объекту или функции через указатель требует разыменование указателя.

Объектно-ориентированные особенности

C++ добавляет к C объектно-ориентированные возможности. Он вводит классы, которые обеспечивают три самых важных свойства ООП: инкапсуляцию, наследование и полиморфизм.

Методы класса — это функции, которые смогут применяться к экземплярам класса. Грубо говоря, метод — это функция объявленная внутри класса и предназначенная для работы с его объектами. Методы объявляются в теле класса. Описываться могут там же, но могут и за пределами класса (внутри класса в таком случае достаточно представить прототип метода, а за пределами класса определять метод поставив перед его именем — имя класса и оператор ::). Методы и поля входящие в состав класса называются членами класса. При этом методы часто называют функциями-членами класса.

Наследование

В C++ при наследовании одного класса от другого наследуется реализация класса, плюс класс-наследник может добавлять свои поля и функции или переопределять функции базового класса. Множественное наследование разрешено. Конструктор наследника вызывает конструкторы базовых классов, а затем конструкторы нестатических членов-данных, являющихся экземплярами классов. Деструктор работает в обратном порядке. Наследование бывает публичным, защищённым и закрытым.

Полиморфизм

Целью полиморфизма, применительно к объектно-ориентированному программированию, является использование одного имени для задания общих для класса действий. Выполнение каждого конкретного действия будет определяться типом данных. Преимуществом полиморфизма является то, что он помогает снижать сложность программ, разрешая использование того же интерфейса для задания единого класса действий. Выбор же конкретного действия, в зависимости от ситуации, возлагается на компилятор. Полиморфизм может применяться также и к операторам.

Инкапсуляция

Основным способом организации информации в C++ являются классы. В отличие от структуры (struct) языка C, которая может состоять только из полей и вложенных типов, класс (class) C++ может состоять из полей, вложенных типов и функций-членов. Инкапсуляция в С++ реализуется через указание уровня доступа к членам класса: они бывают публичными (public), защищёнными (protected) и закрытыми (private). В C++ структуры отличаются от классов тем, что по умолчанию члены и базовые классы у структуры публичные, а у класса — собственные.

Стандартная библиотека

В языке программирования C++ термин Стандартная Библиотека означает коллекцию классов и функций, написанных на базовом языке. Стандартная Библиотека поддерживает несколько основных контейнеров, функций для работы с этими контейнерами, объектов-функции, основных типов строк и потоков (включая интерактивный и файловый ввод-вывод), поддержку некоторых языковых особенностей, и часто используемые функции для выполнения таких задач, как, например, нахождение квадратного корня числа. Стандартная Библиотека языка C++ также включает в себя спецификации стандарта ISO C90 стандартной библиотеки языка Си. Функциональные особенности Стандартной Библиотеки объявляются внутри пространства имен std.

Достоинства:

* Высокая совместимость с языком Си
* Вычислительная производительность
* Поддержка различных стилей программирования: структурное, объектно-ориентированное, обобщённое программирование, функциональное программирование, порождающее метапрограммирование.
* Автоматический вызов деструкторов объектов (в порядке обратном вызову конструкторов) упрощает и повышает надёжность управления памятью и другими ресурсами (открытыми файлами, сетевыми соединениями, т. п.).
* Перегрузка операторов
* Шаблоны (дают возможность построения обобщённых контейнеров и алгоритмов для разных типов данных)
* Возможность расширения языка для поддержки парадигм, которые не поддерживаются компиляторами напрямую
* Доступность. Для С++ существует огромное количество учебной литературы, переведённой на всевозможные языки

Недостатки:

* Плохо продуманный синтаксис сужает спектр применимости языка
* Язык не содержит многих важных возможностей
* Язык содержит опасные возможности
* Производительность труда программистов на языке оказывается неоправданно низка
* Громоздкость синтаксиса
* Тяжелое наследие
* Необходимость следить за памятью

**Code::Blocks** — свободная кроссплатформенная среда разработки. Code::Blocks написана на С++ и использует библиотеку wxWidgets. Имея открытую архитектуру, может масштабироваться за счёт подключаемых модулей. Поддерживает языки программирования С, С++, D (с ограничениями), Fortran.

Code::Blocks разрабатывается для Windows, Linux и Mac OS X. Среду можно собрать из исходников практически под любую Unix-подобную систему, например FreeBSD[2], PC-BSD

Возможности

Возможности компиляции

* Поддержка множества компиляторов (не включены)
* MinGW / GCC C/C++
* GNU ARM GCC Compiler
* GNU AVR GCC Compiler
* GNU GCC Compiler for PowerPC
* Microsoft Visual C++ 6[3]
* Microsoft Visual C++ Toolkit 2003
* Microsoft Visual C++ 2005/2008 (с некоторыми ограничениями)
* Microsoft Visual C++ 2010 (без поддержки отладчика, требуется DDK)

Возможности интерфейса

* Подсветка синтаксиса
* Сворачивание блоков кода
* Автодополнение кода
* Браузер классов
* Планировщик для нескольких пользователей
* Система проверки правописания (только для комментариев)
* Автоформатирование кода AStyle настраиваемый Code Style
* Утилита для разработки регулярных выражений (использует wxWidgets regexp parser, синтаксис pcre)
* Поиск по проекту с подсветкой найденных совпадений. Поддерживает регулярные выражения
* Поиск места декларации или реализации идентификатора (функции, макроса, класса и т. д.), включая подключенные заголовки из внешних библиотек
* Переход \*.h <-> \*.cpp (F11)

Структура данных.

В данной работе я использую библиотеку “cstring.h” и <fstream>, так как программа подразумевает работу с строками и с файлами

Переменные используются двух типов : int и string.

Программа состоит из 9 файлов: main.cpp, kontragent.h, kontragent.cpp, order.h, order.cpp, cashIN.h, cashIN.cpp, cashOUT.h, cashOUT.cpp.

Класс kontragent:

Этот класс подразумевает собой создание объекта, который будет выполнять роль контрагента в коде.

В этом классе используются:

int account- поле, которое несет в себе информацию о личном счете

string title- поле, которое несет в себе информацию о названии организации

kontragent()-конструктор нужен для того, чтобы объект класса kontragent можно было использовать в качестве поля в другом классе и чтобы можно было создавать объект

void setAccount(int);

void setTitle(string);

int getAccount()const;

string getTitle() const;

~kontragent();

Также здесь используются функции

CIN ввод, Print вывод и перегрузка оператора вывода

Объявление полей и методов хранятся в kontragent.h,а их описание в kontragent.cpp,

Класс Order

Этот класс используется в качестве базового, от которого в дальнейшем будет происходить наследование

kontragent \*P объект другого класса используется в качестве поля

int pay поле, которое несет в себе информацию о сумме платежа

int date поле, которое несет в себе информацию о дате проведения операции

string purpose поле, которое несет в себе информацию о назначении платежа

order();

void setPay(int);

void setDate(int);

void setPurpose(string);

int getPay() const;

int getDate()const;

string getPurpose()const;

void CIN (); функция для ввода элементов

void print(); функция для вывода элементов

~order();

Класс cashIN

Этот класс наследуется от класса Order и выполняет роль приходного кассового ордера

Класс cashOUT

Этот класс наследуется от класса Order и выполняет роль расходного кассового ордера

Сам код реализуется в файле main.cpp. Для того, чтобы записанная в объекты информация не удалялась и не терялась, реализуется так называемая база данных с помощью массивов. Весь итог программы сохраняется в отдельный текстовый файл.

Заключение

В данной практической работе было приведено описание приложение эмулирующее функционал бухгалтерской программы.

Все поставленные задачи были выполнены.

В современном мире практически все компьютеризировано, поэтому бухгалтерия ведет электронный учет своей работы. Следовательно, можно сделать вывод, что данное приложение очень актуально на сегодняшний день.

Kontragent.cpp

#include "kontragent.h"

using namespace std;

kontragent::kontragent(){}

void kontragent::setAccount(int account)

{ this->account=account;}

void kontragent::setTitle(string title)

{ this->title=title;}

void kontragent::CIN()

{ cout<<"Enter account\t";

cin>>account;

this->setAccount(account);

cout<<"Enter title\t";

cin>>title;

this->setTitle(title);}

int kontragent:: getAccount()const

{ return account;}

string kontragent::getTitle()const

{ return title;}

kontragent::~kontragent(){}

bool operator<<(ofstream& fout, const kontragent& P)

{ fout <<"\taccount\t"<< P.getAccount() ;

fout << "\ttitle"<<P.getTitle() ;

fout<<"\n";

return fout;}

Kontragent.h

#ifndef KONTRAGENT\_H\_INCLUDED

#define KONTRAGENT\_H\_INCLUDED

using namespace std;

#include<iostream>

#include<string.h>

#include <fstream>

class kontragent {

protected;

int account;

string title;

public:

kontragent();

void setAccount(int);

void setTitle(string );

int getAccount()const;

string getTitle() const;

void CIN();

~kontragent();

friend bool operator<<(ofstream& fout, const kontragent& obj);

};

Order.cpp

#include "order.h"

using namespace std;

order::order(){}

void order::setPurpose(string purpose)

{ this->purpose=purpose;}

void order::setDate(int date)

{ this->date=date;}

void order::setPay(int pay)

{ this->pay=pay;}

int order:: getPay()const

{ return pay;}

int order::getDate()const

{ return date;}

string order::getPurpose()const

{ return purpose;}

void order:: CIN (kontragent \*P)

{ int pay, date;

string purpose;

cout<<"Enter pay\t";

cin>>pay;

this->setPay(pay);

cout<<"Enter date\t";

cin>>date;

this->setDate(date);

cout<<"Enter purpose\t";

cin>>purpose;

this->setPurpose(purpose);

this->P=P;}

void order:: print()

{cout<<endl<<"date\t"<<this->getDate()<<"\ttitle\t"<<this->P->getTitle()<<"\taccount\t"<<this->P->getAccount()<<"\tpurpose\t"<<this->getPurpose()<<"\tpay\t"<<this->getPay()<<endl;}

order::~order(){}

bool operator<<(ofstream& fout, const order& P)

{ fout <<"\tdate\t"<< P.getDate() ;

fout <<"\tpay\t"<< P.getPay() ;

fout <<"\tpurpose\t"<<P.getPurpose();

fout<<"\n" ;

return fout;}

Order.h

#define ORDER\_H\_INCLUDED

using namespace std;

#include<iostream>

#include<string.h>

#include"kontragent.h"

#include <fstream>

class order {

protected:

kontragent \*P;

int pay;

int date;

public:

string purpose;

order();

void setPay(int);

void setDate(int);

void setPurpose(string );

int getPay() const;

int getDate()const;

string getPurpose()const;

void CIN (kontragent\*P);

void print();

friend bool operator<<(ofstream& fout, const order& obj);

~order();

};#endif // ORDER\_H\_INCLUDED

#endif // KONTRAGENT\_H\_INCLUDED

cashIN.cpp

#include "cashIN.h"

cashIN.h

#ifndef CASHIN\_H\_INCLUDED

#define CASHIN\_H\_INCLUDED

#include "order.h"

class cashIN:public order{};

#endif // CASHIN\_H\_INCLUDED

CashOUT.cpp

#include "cashOUT.h"

cashOUT.h

#ifndef CASHOUT\_H\_INCLUDED

#define CASHOUT\_H\_INCLUDED

#include "order.h"

class cashOUT:public order{};

#endif // CASHOUT\_H\_INCLUDED

Main.cpp

using namespace std;

#include<iostream>

#include<string>

#include"kontragent.h"

#include"order.h"

#include"cashIN.h"

#include"cashOUT.h"

#include <fstream>

int main()

{int i,x,k,j, date1, pay1=0, pay2=0, account1, account2, balance=0,n = 0;

string purpose1, purpose2, title1,title2;

ofstream fout;

fout.open("information.txt", ios::app);

cout<<"enter the number of elements"<<endl;

cin>>x;

kontragent kin[x];

kontragent kout[x];

cashIN in[x];

cashOUT out[x];

cout<<"\nenter elements of cashIN\n\n"<<endl;

fout<<"Elements of cashIN\n";

for(i=0; i<x; i++)

{ kin[i].CIN();

in[i].CIN(&kin[i]);

cout<<"\n\t\t\*\*\*\*\*\*\*\*\*\*\n";}

for(i=0; i<x; i++)

{ in[i].print();

fout<<in[i];}

fout<<"Elements of cashOUT\n";

cout<<"\nenter elements of cashOUT\n\n";

for(j=0; j<x; j++)

{ kout[j].CIN();

out[j].CIN(&kout[j]);

cout<<"\n\t\t\*\*\*\*\*\*\*\*\*\*\n";}

for(j=0; j<x; j++)

{ out[j].print();

fout<<out[j];}

cout<<"\n information purpose 1\n"<<"\n information account 2\n"<<"\n information title 3\n"<<"\n information balance 4\n";

cin>>k;

switch(k){

case 1:

cout <<"purpose1"<<endl;

cin>>purpose1;

n=-1;

for ( i = 0; i < x; i++)

{if (in[i].getPurpose() == purpose1)

n= i;}

if (n!= -1)

{ in[ n].print();

fout<<"information (purpose)\n";

fout<<in[n];

} else

{ cout << " purpose it's not find" << endl;

fout<<"information (purpose)\n";

fout<< " \npurpose it's not find\n";

}cout <<"purpose2"<<endl;

cin>>purpose2;

n=-1;

for ( j = 0; j < x; j++)

{if (out[j].getPurpose() == purpose2)

n= j;

}if (n!= -1)

{cout <<"purpose2"<<endl;

out[ n].print();

fout<<"information (purpose)\n";

fout<<out[n];

}else

{ cout << " purpose it's not find\n" ;

fout<<"information (purpose)\n";

fout<<"purpose it's not find\n" ;

} break;

///////////////////////////////////////////////////////

case 2:

cout <<"account1"<<endl;

cin>>account1;

n= -1;

for ( i = 0; i < x; i++)

{ if (kin[i].getAccount() == account1)

n= i;}

if (n!= -1)

{ in[ n].print();

fout<<"information (account)\n";

fout<<in[n];

}else

{ cout << " account it's not find\n" ;

fout<<"information (account)\n";

fout<<"account it's not find\n" ;

} cout <<"account2"<<endl;

cin>>account2;

n= -1;

for ( j = 0; j < x; j++)

{ if (kout[j].getAccount() == account2)

n= j;

}if (n!= -1)

{ out[ n].print();

fout<<"information (account)\n";

fout<<out[n];

} else

{ cout << " account it's not find\n";

fout<<"information (account)\n";

fout<<"account it's not find\n" ;

} break;

case 3 :

cout <<"title1"<<endl;

cin>>title1;

n= -1;

for ( i = 0; i < x; i++)

{ if (kin[i].getTitle() == title1)

n= i;

} if (n!= -1)

{ in[ n].print();

fout<<"information (title)\n";

fout<<in[n];

} else

{ cout << " title it's not find\n" ;

fout<<"information (title)\n";

fout<<"title it's not find\n" ;

} cout <<"title2"<<endl;

cin>>title2;

n= -1;

for ( j = 0; j < x; j++)

{ if (kout[j].getTitle() == title2)

n= j;

} if (n!= -1)

{ out[ n].print();

fout<<"information (title)\n";

fout<<out[n];

} else

{ cout << " title it's not find\n";

fout<<"information (title)\n";

fout<<"title it's not find\n" ;

} break

case 4:

cout <<"\n\t\t\tdate for balance\n"<<endl;

cin>>date1;

for ( i = 0; i < x; i++)

{ if (in[i].getDate() <= date1)

{ pay1=pay1+in[i].getPay();

} }for ( j=0; j<x; j++)

{ if (out[j].getDate() <= date1)

{ pay2=pay2+out[j].getPay();} }

if ((pay1!=0)&&(pay2!=0))

{ balance=pay1-pay2;

cout<<"balance = "<<balance<<"pay1 - "<<pay1<<"pay2 "<<pay2;

fout<<"information (balance)\n";

fout<<"balance= "<<balance;

} break; }fout.close();

return 0;

}