МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«Курганский государственный университет»

Кафедра автоматизации производственных процессов

# ИЗУЧЕНИЕ ОСНОВ ПРОГРАММИРОВАНИЯ НА ЯЗЫКЕ VC++ В СРЕДЕ WINDOWS

МЕТОДИЧЕСКИЕ УКАЗАНИЯ
к комплексу лабораторных работ по дисциплине
«Технология программирования»
для студентов специальностей 220700. 62 «Автоматизация технологических процессов и производств» и 220400.62 «Управление в технических системах»

Курган 2013

Кафедра автоматизации производственных процессов

Дисциплина: «Технология программирования»

Составил: ст. преподаватель Скобелев И.В.

Утверждены на заседании кафедры «25» декабря 2012 г.

Рекомендованы методическим советом университета «2» апреля 2013 г.

**СОДЕРЖАНИЕ**

## ВВЕДЕНИЕ

В настоящее время программирование трансформировалось в целую индустрию производства программных изделий. Поэтому уже мало знать только язык программирования и операционный подход к составлению алгоритмов. Профессиональный разработчик программных изделий должен владеть теорией проектирования, методами активизации мышления. Ему необходимо умение оперирования моделями, методами генерации решений и выбора их оптимальных вариантов. Создание программных изделий коллективом разработчиков предопределило необходимость умения планирования работ и их распределения между отдельными участниками проекта.

На основе анализа математических моделей и алгоритмов решения экономических и других задач, программист обязан:

— разрабатывать программы, обеспечивающие возможность выполнения алгоритма и соответственно поставленной задачи средствами вычислительной техники; проводить их тестирование и отладку;

— разрабатывать технологию решения задачи по всем этапам обработки информации;

— осуществлять выбор языка программирования для описания алгоритмов и структур данных;

— определять информацию, подлежащую обработке средствами вычислительной техники, ее объемы, структуру, макеты и схемы ввода, обработки, хранения и вывода, методы ее контроля;

— выполнять работу по подготовке программ к отладке и проводить отладку;

— определять объем и содержание данных контрольных примеров, обеспечивающих наиболее полную проверку соответствия программ их функциональному назначению;

— осуществлять запуск отлаженных программ и ввод исходных данных, определяемых условиями поставленных задач;

— проводить корректировку разработанной программы на основе анализа выходных данных;

— разрабатывать инструкции по работе с программами, оформлять необходимую техническую документацию;

— определять возможность использования готовых программных продуктов;

— осуществлять сопровождение внедренных программ и программных средств;

— разрабатывать и внедрять системы автоматической проверки правильности программ, типовые и стандартные программные средства;

— составлять технологию обработки информации;

Данные методические указания призваны помочь студентам получить необходимые практические навыки для самостоятельной работы на языке программирования.

## Лабораторная работа №1
## Реализация технологии визуального программирования в среде Microsoft Visual

Продолжительность 2 часа.

**Цель работы: Научиться использовать средства визуального программирования Microsoft Visual Studio для разработки диалоговых форм.**

**Задание:** Разработать форму диалога для заполнения и сохранения данных в счет-фактуре. В форме должны присутствовать поля: 1). наименование товара ( реализовать виде выбора из списка выпускаемых предприятием изделий, или оказываем услуг.)
2) цена единицы – заполняется автоматически при выборе товара. 3) количество – заполняется оператором. 4) сумма – вычисляется автоматически. 5) процент налога заполнен по умолчанию, но может изменяться оператором. 6) сумма налога вычисляется автоматически. 7) Сумма к уплате – вычисляется автоматически. 8) наименование и реквизиты покупателя. 9) Дата оплаты. 10) дата отгрузки.

Выполнение работы:

1. Разработать диалоговою форму для внесения в базу данных предприятия счет-фактур отпускаемой продукции.
2. Нарисовать блок-схему работы элементов формы и их взаимодействия.
3. Реализовать в виде работающей программы с помощью Microsoft Visual C++
4. **Написать отчет о проделанной работе.**

Контрольные вопросы:

1. Для чего нужна «технология программирования»?.
2. Какие этапы развития прошла дисциплина «технология программирования»?
3. Для чего используется визуальное программирование и с чем связано его появление?
4. Какие системы визуального программирования Вы знаете?
5. Как осуществить выбор из списка.
6. Как привязать программный код к элементам дизайна?
7. Для чего служит ClassWizard?

## Лабораторная работа №2

## Исследование возможностей структурного и модульного программирования на базе языка VC++.

Продолжительность 2 часа.

**Цель работы: Научиться разбивать программы на отдельные модули и осуществлять связь между модулями в работающей программе.**

**Задание:** Для разработанной на предыдущем лабораторном занятии формы, разработать модули: сохранения данных в файле, редактирования выбранной счет фактуры, удаление или пометки на удаление счет фактуры, выборки счет-фактур за определенный период времени, и ( или) по покупателю, нахождения покупателей с большими объемом закупок и т.д.

Выполнение работы:

1. Создать проект или использовать полученный в предыдущей работе проект.
2. Дополнить его максимально возможным количеством функций.
3. Возможна разработка отдельных функций разными учащимися и последующее объединение в один проект.
4. Нарисовать блок-схему программы..
5. **Написать отчет о проделанной работе.**

Контрольные вопросы:

1. Для чего нужно модульное программирование?
2. Какие преимущества и недостатки, по вашему мнению, имеет модульное программирование?
3. Когда появилась необходимость в модульном программировании.?
4. Понятие структурного программирования, в чем особенность структурного подхода?
5. Как осуществляется конструирование графического интерфейса?
6. Как связать переменную с элементом графического интерфейса?
7. Как добавить метод, обрабатывающий реакцию на события, связанные с элементом графического интерфейса?

## Лабораторная работа №3
## Решение задач программирования с использованием технологии Объектно-ориентированного метода

Продолжительность 4 часа.

**Цель работы: Изучить основы объектно-ориентированного программирования на примере создания – клиент- серверного приложения.**

**Задание:** Усовершенствовать созданную в предыдущих лабораторных работах программы, сделав на ее основе клиент-серверное приложение. В данной работе необходимо разработать клиент, который должен передавать сведения в центр.

Выполнение работы:

1. Переделываем или создаем новое приложение, используя знания полученные в лабораторной работе №13 из курса программирование и алгоритмизация приложение типа «клиент»
2. Добавьте в проект класс MySocket как производный от стандартного класса CSocket с помощью ClassWizard. Добавьте во вновь образованный класс виртуальные функции для передачи сообщения о разрыве связи OnClose, функцию для приема сообщений от сервера OnReceive и указатель для связи с формой диалогового.
3. В класс формы диалогового окна добавьте необходимый код для работы кнопок установки связи с сервером и кнопки передачи сообщений. Объявите все необходимые переменные для реализации работы программы клиента
4. **Написать отчет о проделанной работе. В отчете алгоритм решения задачи изобразить в виде блок-схемы.**

**Пояснения к выполнению работы:**

Для реализации той части программы, которая, собственно, и отвечает за все взаимодействие с сетью, нам понадобится три класса: - CClientSocket, CListeningSocket и CPool. Первый создается для каждого нового клиента, подсоединяющегося к серверу. CListeningSocket, как видно из названия, - "слушающий" сокет. Его основная и единственная задача - отвечать на запросы о подключении новых клиентов. CPool хранит в себе все создаваемые сокеты, управляет ими и осуществляет связь с остальной частью программы. Теперь рассмотрим каждый класс подробно.

```
class CClientSocket : public CSocket
{
public:
  CClientSocket(CPool* ppool);
  virtual void OnReceive(int nErrorCode);
  virtual void OnClose(int nErrorCode);
protected:
  CPool* pPool;
};
```

Этот класс - наследник CSocket, который, в свою очередь, является наследником CAsyncSocket. Последние два класса принадлежат библиотеке MFC. В них выполнена вся черная работа по упаковке WinSock API в объектно-ориентированную обертку. Заголовки классов находятся в файле afxsock.h, а особо любопытные могут заглянуть в их исходный текст sockcore.cpp, находящийся в том же каталоге, что и остальные исходные материалы библиотеки. Строго говоря, основная нагрузка лежит на CAsyncSocket, и можно было бы свой класс наследовать от него, а не от CSocket, но, несмотря на то, что классы выглядят очень похожими, в их устройстве, а значит и использовании есть существенные различия.

Некоторые функции, принадлежащие CAsyncSocket, такие, как Receive, Send, Accept, всегда возвращают ошибку с кодом WSAEWOULDBLOCK. Это связано с тем, что они не дожидаются своего полного выполнения, и поэтому результат их вызова приходится определять в других местах программы. В CSocket те же самые функции ожидают, пока все необходимые действия не завершатся, и лишь тогда возвращают управление программе. Сокет с такими свойствами называется блокирующим. Для взаимодействия в локальной сети такие задержки в большинстве случаев не будут заметны. Но при использовании Интернета время ожидания может оказаться слишком большим. Тогда существует два варианта: один - воспользоваться CAsyncSocket и самостоятельно создать механизм обработки подтверждений. Его можно реализовать следующим образом: после вызова упомянутых функций устанавливаем таймер и, если по прошествии некоторого времени не получаем уведомление об удачном выполнении затребованных действий, констатируем неудовлетворительный результат. Другой - выделить для CSocket отдельный поток.

Теперь вернемся к нашему классу. Единственная необходимая его переменная - указатель на объект типа CPool, т.е. указатель на хозяина. Его инициализация совершенно необходима, поэтому конструктор по умолчанию заменяем на следующий.

```
CClientSocket::CClientSocket(CPool* ppool)
{
  pPool=ppool;
}
```

Кроме того, нам потребуется переопределить две виртуальные функции. Кстати, ClassWizard поддерживает классы сокетов, так что требуется просто выбрать название нужной функции из списка. Вообще, если им пользоваться, то про сокеты он еще много чего напишет, предложит переопределить копирующий конструктор и т.д., но на это творчество можно спокойно закрыть глаза.

```
void CClientSocket::OnReceive(int
  nErrorCode)
{
char buff[4096];
  CString buffer;
int nRead;
nRead=Receive(buff, 4096);
switch (nRead)
{
case 0:
    Close();
    break;
case SOCKET_ERROR:
    AfxMessageBox ("Error occurred");
    Close();
  break;
default:
  buff[nRead]='\0';
  buffer=buff;
pPool->ProcessRead(this,buffer);
  }
  CSocket::OnReceive(nErrorCode);
}
```

Эта функция вызывается средой, чтобы сообщить сокету о получении новых данных. Их мы можем получить, используя функцию Receive, которой в качестве параметров передается указатель на массив памяти с записанными принятыми байтами и размером массива. В случае успешного выполнения функция возвращает все полученные байты. При возникновении ошибки возвращается SOCKET_ERROR. Кроме того, получение нулевого значения трактуется как сигнал об обрыве соединения, такое может произойти, если передающий сокет был закрыт во время транзакции. Для того чтобы известить о произошедшем "вышестоящие органы", вызываем ProcessRead(this), передавая себя самого через аргумент.

И последняя функция

```
void CClientSocket::OnClose(int
  nErrorCode)
{
  pPool->ProcessClose(this);
CSocket::OnClose(nErrorCode);
}
```

Она вызывается при закрытии текущего соединения. Переопределять ее нужно только для того, чтобы сообщить о закрытии CPool. Обратите внимание на вызов функций, принадлежащих предкам. Microsoft настаивает, что это необходимо. На

самом деле функции пустые, но не исключено, что в последующих версиях там может появиться какой-либо код.

Контрольные вопросы:

1. В чем смысл объектно-ориентрованного подхода в программирование?
2. Что такое объект и что такое класс?
3. Что значит наследование?
4. Как создаются классы?.
5. Как создаются объекты.
6. Как добавить класс работающий с Socket?

## Лабораторная работа №4
## Создание программ в среде Microsoft Visual Studio. Применение средств автоматизации разработки приложений

Продолжительность 4 часа.

**Цель работы: Научиться создовать классы и объекты в VC++. Получить практические навыки по разработке приложения на основе классов и объектов в Microsoft Visual Studio VC++ .**

**Задание:** Создать серверную часть программы для получения сообщений от клиентов и сохранения результатов в базе данных центра.

Выполнение работы:

1. Создать приложение которое будет выполнять роль сервера и принимать данные от клиентов в центре.
2. Добавьте в проект класс CClientSocket как производный от стандартного класса CSocket с помощью ClassWizard. Добавьте во вновь образованный класс виртуальные функции для передачи сообщения о разрыве связи OnClose , функцию для приема сообщений от сервера OnReceive и указатель для связи с формой диалогового окна.
3. Добавьте в проект класс CListeninngSocket как производный от стандартного класса CSocket с помощью ClassWizard. Добавьте во вновь образованный класс виртуальную функцию OnAccept(int nErrorCode) для реакции на подключение очередного клиента и указатель для связи с формой диалогового окна.
4. Добавьте в проект класс CPool, в котором и будем производить всю работу, связанную с созданием сокетов и передачей информации между сервером и клиентами. Добавьте в него необходимые для обработки и передачи информации функции. BOOL Send(int index,CString data); void ProcessRead(CClientSocket *pSocket,CString msg); int ProcessClose(CClientSocket *pSocket); int ProcessAccept();BOOL Send2All(CString data); BOOL Init(void *phost,int port);
5. Добавьте в класс диалогового окна функцию для отображения сообщений от клиентов UpdateWiew(), а также методы для передачи сообщений
6. **Написать отчет о проделанной работе. В отчете алгоритм решения задачи изобразить в виде блок-схемы.**

**Пояснения к выполнению работы:**

```
class CListeningSocket : public CSocket
{
public:
CListeningSocket(CPool* ppool);
  virtual void OnAccept(int nErrorCode);
protected:
CPool* pPool;
}
```

Класс также хранит указатель на своего владельца и имеет такой же конструктор, какой и CClientSocket. Для добавления функциональности достаточно переопределить всего одну функцию.

```
void CListeningSocket::OnAccept(int
  nErrorCode)
{
pPool->ProcessAccept();
CSocket::OnAccept(nErrorCode);
}
```

Она вызывается системой при запросе нового соединения. В отличие от предыдущего класса информирующая функция ProcessAccept() не передает указатель this. Но это и не требуется, поскольку объект данного типа существует в единственном экземпляре.
Как можно увидеть, вся деятельность описанных классов связана в основном с информированием CPool о том, что с ними происходит важного. Рассмотрим, наконец, сам класс CPool. Он не имеет предков и наделен широкими полномочиями.

```
class CPool
{
public:
  CPool();
  virtual ~CPool();
  BOOL Init(void *phost, int port);
  BOOL Send2All(CString data);
  void ProcessClose(CClientSocket *pSocket);
  void ProcessRead(CClientSocket* pSocket,CString msg);
  int ProcessAccept();
protected:
  CListeningSocket *m_pSocket;
  CPtrList connectionList;
  void *pHost;
};
```

Для хранения клиентских сокетов, используем CPtrList. Это достаточно удобно и в нашем случае эффективнее, чем, скажем, CPtrArray. Для связи с остальными частями программы CPool имеет указатель на некоторый внешний объект pHost.
Конечно, в классе объявлен указатель на слушающий сокет; как и другие его "двоюродные братья", этот объект также будет создан динамически. Перейдем к функциям.

```
BOOL CPool::Init(void *phost,int port)
{
  pHost=(CChildView*)phost;
```

```
  m_pSocket = new CListeningSocket(this);
  if (m_pSocket->Create(port))
  {
    if(m_pSocket->Listen())
      return TRUE;
  }
  return FALSE;
}
```

Нет смысла записывать производимые здесь действия в конструктор, так как можно к этому моменту не знать номера порта, а кроме того, до начала активных действий с сокетом необходимо вызвать AfxSocketInit(), функцию, которая может вернуть ошибку. Если вам не хочется самим набирать эту часть кода, воспользуйтесь компонентом из Component Gallery. Он добавит инициализацию сокетов в функцию InitInstance() класса приложения. То же самое будет сделано, если в AppWizard попросить поддержку сокетов. Запуск сокета на прослушивание "эфира" осуществляет функция Listen(), а номер порта, на котором сокет будет это делать, передается через функцию Create().

Следующая функция

```
int CPool::ProcessAccept()
{
CClientSocket* pSocket =
  new CClientSocket(this);
if (m_pSocket->Accept(*pSocket))
{
connectionList.AddTail(pSocket);
    pHost->UpdateView();
}
else
    delete pSocket;
  return 0;
}
```

Добавляет созданный CClientSocket в хранилище (функция AddTail) и сигнализирует о необходимости обновить информацию об имеющихся подключениях. Вообще все вызовы функций объекта pHost не относятся непосредственно к механизму взаимодействия сокетов и добавлены исключительно ради придания жизни исходному коду. Передача соединения от слушающего сокета клиентскому происходит в m_pSocket->Accept(*pSocket).

Обработка отключения клиента реализована в

```
void CPool::ProcessClose(CClientSocket *pSocket)
{
POSITION pos = connectionList.Find(pSocket);
connectionList.RemoveAt(pos);
delete pSocket;
  pHost->UpdateView();
}
```

На функции ProcessRead, по идее, должна лежать основная нагрузка по разбору полученного, но, поскольку обмен происходит только строковыми данными, здесь больше и делать нечего.

```
void CPool::ProcessRead(CClientSocket *pSocket,CString msg)
{
  pHost->UpdateView();
  pHost->editbox.AddText(": "+msg);
}
```

В большинстве реальных приложений может оказаться необходимой возможность посылки одного сообщения всем подключенным на данный момент клиентам. Эту задачу выполняет функция Send2All(CString data).

```
BOOL CPool::Send2All(CString data)
{
  if(data.GetLength()==0)
    return FALSE;
  for(POSITION pos=connectionList.
    GetHeadPosition();pos != NULL;)
  {
CClientSocket* pSock = (CClientSocket*)
  connectionList.GetNext(pos);
    pSock->Send(data,data.GetLength());
  }
  return TRUE;
}
```

Из connectionList поочередно берется указатель на клиентский сокет, и, чтобы послать данные, вызывается функция Send, которой необходимо передать указатель на источник данных и его размер в байтах. В приведенном выше коде data - это переменная типа CString. Здесь стоит отметить, что если объем данных больше, чем можно передать за один раз, то исходный блок будет разбит и передан по частям. В этом случае функция OnRecieve вызывается несколько раз, а при использовании CAsyncSocket будет вызвана еще и OnOutOfBandData, извещающая о фрагментации данных. Максимальный размер блока в SDK и прочих документациях указывать не любят. Еще наполним содержанием деструктор, где освободим всю выделенную память.

```
CPool::~CPool()
{
  while(!connectionList.IsEmpty())
  {
  CClientSocket*pSocket=(CClientSocket*)
    connectionList.RemoveHead();
  if(pSocket!=NULL)
  {
    pSocket->Close();
    delete pSocket;
```

```
  }
  }
  m_pSocket->Close();
  delete m_pSocket;
}
```

Теперь, когда мы рассмотрели все необходимые классы, проследим за последовательностью действий, выполняемых при подключении нового клиента. При получении требования на соединение вызывается функция CListeningSocket::OnAccept, она, в свою очередь, вызывает ProcessAccept, принадежащую CPool. В ProcessAccept создается новый объект класса CClientSocket. С этого момента он и отвечает за взаимодействие с клиентом, а слушающий сокет снова готов обрабатывать запросы о новых подключениях. Вот, пожалуй, и все, что касается сервера.

Действия, выполняемые на стороне клиента, совсем просты. Но, на всякий случай, приведу и их. Для наглядности в класс сокета я добавил взаимодействие с окном диалога. Даже с конкретной привязкой к реализации программы код практически не увеличился в объеме. Переопределяем уже знакомые функции. Единственное поле данных - указатель на объект класса, производного от CDialog. Это конкретная реализация pHost из предыдущих классов.

```
class MySocket : public CSocket
{
public:
  MySocket(CClientDlg *pDlg)
    {m_pDlg = pDlg;};
  virtual void OnClose(int nErrorCode);
  virtual void OnReceive(int nErrorCode);
  CClientDlg *m_pDlg;
};
void MySocket::OnClose(int nErrorCode)
{
  m_pDlg->GetDlgItem(IDC_CONNECT)->
    EnableWindow(TRUE);
  delete this;
  CSocket::OnClose(nErrorCode);
}
void MySocket::OnReceive(int nErrorCode)
{
  char st[4096];
  int r=Receive(st, 4096);
  st[r]='\0';
  m_pDlg->SetDlgItemText(IDC_ANSWER,st);
  CSocket::OnReceive(nErrorCode);
}
```

IDC_ANSWER и IDC_CONNECT - идентификаторы соответствующих ресурсов, первый - окно редактирования, второй - кнопка, нажатием на которую клиент пытается соединиться с сервером. Если это получается, то кнопка блокируется. Это действие реализуется следующим образом:

```
pSocket = new MySocket(this);
pSocket->Create();
if(pSocket->Connect("127.0.0.1", port))
{
  GetDlgItem(IDC_CONNECT)->EnableWindow(FALSE);
  pSocket->Send(Name,Name.GetLength());
}
else
  delete pSocket;
```

Функции Connect необходимы следующие два параметра: LPCTSTR lpszHostAddress - адрес узла сети, он может быть числовым ip-адресом (я использовал значение 127.0.0.1, так как оно представляет собой адрес локального компьютера) или буквенным, т. е. DNS-адресом, например "someserver.ru". Второй параметр - номер порта. При разрыве соединения кнопка снова активизируется. Что мы и делаем в OnClose().

Контрольные вопросы:

1. Какие средства автоматизации разработки приложения вы знаете?
2. Для чего используются структуры? Приведите примеры использования структур.
3. Как классы вы создавали в лабораторной работе?
4. Как использовали свойства наследования?

## Лабораторная работа №5
## Использование классов библиотеки MFC при создании приложений с помощью Microsoft Visual Studio. Использование динамических и статических библиотек

Продолжительность 4 часа.

**Цель работы: Изучить приемы работы с классами и объектами библиотеки MFC. Научится создавать и отлаживать приложения работающие в сети.**
**Задание:** отладить взаимодействие серверной и клиентской частей системы разработанных в предыдущих лабораторных работах.

Выполнение работы:

1. Запустить серверную часть программы и клиентскую. Отладить их работу во взаимодействии.
2. Отладить работу серверной части с несколькими клиентами одновременно. Решить задачи записи данных в базу данных на сервере и на клиентах..
3. Решить задачу передачи данных на клиент от сервера по запросу. .
4. **Написать отчет о проделанной работе. В отчете алгоритм решения задачи изобразить в виде блок-схемы.**

**Пояснения к выполнению работы:**
Испытываем ранее разработанные программные модули во взаимодействии исправляем выявленные ошибки.

Контрольные вопросы:

1. Для чего служит библиотека .MFC?
2. Какие классы в ы использовали в своем приложении из библиотеки .MFC?
3. Чем отличаются классы от объектов?
4. Какие объекты вы создавали на основе классов из библиотеки .MFC?

## Порядок оформления отчета

Отчет оформляется по результатам выполнения лабораторной работы на стандартных листах бумаги формата А4 с помощью редактора Word. Оформление отчета является важной частью лабораторной работы. При работе над отчетом студент должен осмыслить ход выполнения работы и научиться документировать полученные результаты.

Отчет должен содержать блок-схемы алгоритмов, оформленные по ГОСТу, необходимые структуры данных и распечатки программных кодов, разработанные студентом.

В отчете могут быть приведены тестовые данные, использовавшиеся студентом для проверки правильности работы программы. По результатам проверки программы должны быть сделаны выводы. Если в работе получились не те результаты, которые ожидались, в отчете должны быть отмечены причины, по которым не удалось получить желаемые результаты.

Допускается оформление одного отчета по всему курсу, выполненных в течение семестра лабораторных работ.

Пример оформления титульного листа для лабораторной работы приведен в Приложении.

## Список рекомендуемой литературы


1. Камаев В.А. Технологии программирования: Учебник/В.А. Камаев, В.В. Костерин. — 2-е изд., перераб. и доп. — М.: Высш. шк.,2006. - 454 с: ил.
2. Жоголев Е.А. Технология программирования/Е.А. Жоголев. — М.: Научный мир, 2004. — 216 с.
3. Одинцов И.О. Профессиональное программирование. Системный подход/И.О. Одинцов. - СПб: BHV-СПб., 2002. - 512 с.
4. Давыдов В.Г Программирование и основы алгоритмизации: Учеб. пособие. - М.: Высш.шк , 2003.
5. Холзнер С. Microsoft Visual C++6 с самого начала – СПб.: Питер 2005.
6. Либерти Джесс. Освой самостоятельно C++ за 21 день.- М.: Издательский Дом "Вильяме", 2001. - 814 с.
7. Павловская Т.А. С/С++: Программирование на языке высокого уровня. Учебник. - СПб.: Питер,2001. - 460 с.
8. http://www.cetiforum.ru
9. http://www.microsoft.ru
10. http:// www.firststeps.ru
11. http://www.realcoding.net/
12. Франка П. С ++ учебный курс**:** Питер – 2005.
13. Павловская Т.А., Щупак Ю.А. **С++.** Объектно-ориентированное программирование: Практикум: Питер – 2005

**Приложение 1**

**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ**

КУРГАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Кафедра автоматизации производственных процессов

# ОТЧЕТ

по лабораторным работам
по дисциплине
«Технология программирования»

Выполнил:
студент гр. 214 Иванов И.И.

Проверил:
преподаватель Скобелев И.В.

Курган 2013

Скобелев Игорь Вадимович

**ИЗУЧЕНИЕ ОСНОВ ПРОГРАММИРОВАНИЯ НА ЯЗЫКЕ VC++ В СРЕДЕ WINDOWS**

МЕТОДИЧЕСКИЕ УКАЗАНИЯ
к комплексу лабораторных работ по дисциплине
«Технология программирования»
для студентов специальностей 220700. 62 «Автоматизация технологических процессов и производств» и 220400.22 «Управление в технических системах»

Авторская редакция

| | | |
|---|---|---|
| Подписано в печать17.06.13 | Формат 60x84 1/16 | Бумага тип. №1 |
| Печать трафаретная | Усл.печ.л. 1,25 | Уч.-изд.л. 1,25 |
| Заказ 104 | Тираж 22 | Цена свободная |

Редакционно-издательский центр КГУ.

640669, г. Курган, ул. Гоголя, 25.

Курганский государственный университет.