Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования

АМУРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Факультет математики и информатики

Галаган Т. А.

Учебная практика

Учебно-методическое пособие

Благовещенск

2017

ББК
Г

Печатается по решению редакционно-издательского совета факультета математики и информатики Амурского государственного университета

Т.А. Галаган

Учебная практика. Учебно-методическое пособие для бакалавров направления подготовки 09.03.01 – «Информатика и вычислительная техника». – Благовещенск: Амурский гос. ун-т, 2017, 45 с.

Пособие предназначено для студентов первого курса, впервые проходящих учебную практику. Оно знакомит их с целями и задачами прохождения практики, с требованиями, предъявляемыми к отчету по практике и правилами его оформления, а также содержит необходимые теоретические сведения, необходимые для выполнения задания практики.

Рецензенты:

Е.Ф Алутина канд. физ.-мат. наук, доцент, зав. кафедрой информатики и методики преподавания информатики ФГБОУ ВПО «Благовещенский государственный педагогический университет»;

Н.А Чалкина канд. пед. наук, доцент кафедры общей математики и информатики ФГБОУ ВПО «Амурский государственный университет».

Оглавление

ВВЕДЕНИЕ

Учебная практика является обязательным разделом образовательной программы по направлению подготовки 09.03.01 – Информатика и вычислительная техника (уровень бакалавриата) и представляет собой вид учебных занятий, непосредственно ориентированных на профессионально-практическую подготовку обучающихся.

Учебная практика у студентов первого курса является практикой по получению первичных профессиональных умений и навыков, в том числе первичных умений и навыков научно-исследовательской деятельности.

Учебная практика направлена, прежде всего, на закрепление знаний, умений и навыков, полученных при изучении дисциплин «Информатика» и «Программирование».

Данное учебно-методическое пособие содержит описание целей и задач учебной практики, этапов ее проведения, требования к содержанию и оформлению материалов отчета по практике, а также необходимый материал и краткое изложение теоретических сведений, необходимых для выполнения индивидуального задания. В качестве индивидуального задания практики выступает разработка программного приложения на языке C++.

ПРАВИЛА ПРОВЕДЕНИЯ УЧЕБНОЙ ПРАКТИКИ

Цели и задачи практики. Планируемые результаты прохождения практики

Цель практики – обеспечение непрерывности и последовательности в овладении студентами профессиональной деятельности согласно требованиям к уровню подготовки бакалавров по направлению подготовки 09.03.01 – Информатика и вычислительная техника.

Задачами практики являются:

углубление знаний по дисциплинам, полученным за время обучения на первом курсе, таких как «Программирование», «Информатика», «Математика»;

развитие практических навыков разработки прикладного программного обеспечения и применения современных инструментальных средств для их создания;

развитие практических навыков инсталляции и настройки программного обеспечения общего назначения и специализированных программ;

формирование навыков подготовки и систематизации необходимых материалов и научно-технической информации для выполнения задания практики;

создание условий для практического применения знаний в области общепрофессиональных, специализированных компьютерных и математических дисциплин,

формирование и совершенствование базовых профессиональных навыков и умений в области применения современных информационных технологий;

формирование информационной компетентности с целью успешной работы в профессиональной сфере деятельности;

приобретение навыков создания отчетов, в том числе и научно-

технических;

обеспечение успеха дальнейшей профессиональной карьеры.

Прохождение учебной практики должно способствовать развитию следующих компетенций:

способностью анализировать основные этапы и закономерности исторического развития общества для формирования гражданской позиции (ОК-2);

способностью работать в коллективе, толерантно воспринимая социальные, этнические, конфессиональные и культурные различия (ОК-6);

способностью осваивать методики использования программных средств для решения практических задач (ОПК-2);

Во время учебной практики студент должен:

знать современные технические и программные средства взаимодействия с ЭВМ; технологию разработки алгоритмов и программ; методы отладки и решения задач на ЭВМ в различных режимах; основы объектно-ориентированного подхода к программированию (ОК-2, ОПК-2);

владеть навыками разработки программного обеспечения для решения прикладных задач, использования современных технологий программирования и работы с современными инструментальными средствами разработки программ, тестирования и документирования программ, навыками работы в коллективе (ОК-6, ОПК-2);

уметь проводить анализ предметной области, проводить анализ информационно-образовательных ресурсов; самостоятельно осуществлять поиск необходимой научной и учебной литературы для решения задач профессиональной деятельности (ОК-1, ОК-6, ОПК-2).

Структура и содержание практики

Практика проводится в соответствии с ПУД СМК 48-2016 ПОЛОЖЕНИЕ о практике обучающихся, осваивающих образовательные программы высшего образования – программы бакалавриата, программы специалитета,

программы магистратуры.

Разделы практики и их содержание представлены в таблице 1.

Таблица 1 – Содержание учебной практики

Разделы практики	**Содержание разделов практики**
Подготовительный этап	Заключение договора на практику. Прохождение инструктажа по технике безопасности при работе с оборудованием. Организационное собрание, проводимое руководителем практики от вуза.
Изучение учебно-методического обеспечения по предметной области знания.	Поиск и отбор необходимой информации по теме исследования. Формирование библиографического списка исследуемой области.
Выполнение индивидуального задания	Индивидуальное задание содержит конкретную задачу разработки прикладного программного обеспечения и включает разработку алгоритма решения, кодирование программы и ее отладку. В качестве индивидуального задания может быть научно-исследовательская работа по выбранной теме исследования.
Составление тестовых материалов для проведения вычислительного эксперимента	Тестовые значения должны быть подобраны с учетом граничных условий, рассчитаны прогнозируемые числовые значения.
Индивидуальные консультации с руководителем практики от вуза	Консультации заключаются в регулярном информировании руководителя практики от вуза о проделанной работе, методическая помощь при выполнении индивидуального задания.
Ведение дневника практики	Выполнение ежедневных кратких рабочих записей о проделанной работе.
Подготовка отчета по практике	Сбор необходимой информации, ее анализ и структурирование, описание разработанного программного обеспечения оформление, оформление отчета согласно требованиям.
Заключительный этап	Сдача отчета по практике Защита отчета по практике

Форма отчетности по практике

По окончании учебной практики студент обязан предоставить руково-

дителю практики от университета следующие полностью заполненные документы: дневник практики, отчет по практике, отзыв руководителя практики от организации с оценкой работы студента по пятибалльной шкале, компьютерную презентацию. Без предоставления всех перечисленных документов студент до защиты отчета по практике не допускается.

Общий объем отчета должен составлять 20-25 страниц. В приложении – графики, схемы, фрагменты кодов программы и т.д.

Основные разделы отчета:

Введение. Краткая характеристика современного состояния предметной области.

1. Описание предметной области. Наименование данного раздела может быть иным и отражать непосредственное содержание данного раздела.

2. Выполнение индивидуального задания. Дается описание конкретной задачи выполненной студентом в ходе практики. Название данного раздела должно быть изменено в соответствии с конкретной задачей выполняемой практикантом.

Заключение. Кратко перечисляется, что сделано в результате практики.

Библиографический список. Приводятся все литературные и нормативные источники, которые оформляются согласно правилам оформления письменных работ. На каждый источник должна быть хотя бы одна ссылка в тексте пояснительной записки.

Приложения. В состав приложений включаются, например: формы входных и выходных форм, распечатки экранных форм и содержимого справочных окон, листинги программ и др. На каждое приложение должны быть ссылки в тексте пояснительной записки.

Проверенный и отрецензированный отчет студент защищает руководителю практики. По результатам защиты и оценке рецензии выставляется окончательная оценка за практику, которая заносится в зачетную книжку и экзаменационную ведомость.

В процессе защиты выявляется качественный уровень практики, обращается внимание на инициативу студента, проявленную в период ее прохождения. Учитываются деловые качества студента, умение логично, грамотно и доступно представлять и излагать информацию.

При выставлении студенту оценки по практике принимаются во внимание: отзыв руководителя от предприятия, качество доклада, оформление и содержание отчета, ответы на вопросы. Оценка по практике – зачет с оценкой – приравнивается к оценкам по теоретическому обучению и учитывается при подведении итогов общей успеваемости студентов.

Требования к оформлению текста отчета

Отчет должен быть оформлен и напечатан с использованием компьютера и принтера. Текст располагается на одной стороне листа белой бумаги формата А4. Цвет шрифта должен быть черным, размер – 14, гарнитура – Times New Roman, интервал – полуторный. Текст форматируется по ширине с включением автоматического переноса слов, интервал между абзацами не добавлять.

Размеры полей устанавливаются равными: правое – 10 мм, верхнее и нижнее – 20 мм, левое – 30 мм.

Нумерация страниц – внизу по центру страницы.

Допускается исправление опечаток, ошибок и графических неточностей, обнаруженных в процессе подготовки отчета. Исправления наносятся закрашиванием белой краской, с нанесением на том же месте исправленного текста (графики) машинописным способом или черной пастой рукописным способом. Количество исправлений на одном листе – не более трех.

Наименования таких структурных элементов работы как: содержание, введение, заключение, библиографический список, приложения, следует писать по центру и прописными буквами.

Каждую структурную часть отчета следует начинать с нового листа и

отделять от последующего текста двумя одинарными интервалами.

Основная часть работы делится на разделы, подразделы. Разделы, подразделы нумеруются арабскими цифрами без точки и записываются с абзацного отступа (1,25 см). Разделы должны иметь порядковую нумерацию в пределах всего текста, за исключением приложений. Номер подраздела включает номер раздела и порядковый номер подраздела, разделенные точкой (например, 2.1).

Названия разделов пишутся прописными буквами, выравнивание – по левому краю. Названия подразделов оформляются жирными строчными буквами. Заголовок подраздела не отделяется от последующего текста пустой строкой.

Все иллюстрации отчета (чертежи, графики, схемы, компьютерные скриншоты, диаграммы, фотоснимки) называются рисунками. Рисунки следует располагать в работе непосредственно после текста, в котором они упоминаются впервые, или на следующей странице. На все иллюстрации в работе должны быть даны ссылки.

Иллюстрации должны иметь наименование. Слово «Рисунок» и наименование помещаются после самой иллюстрации с выравниванием по центру следующим образом:

Рисунок 1 – Блок-схема алгоритма программы

Иллюстрации каждого приложения обозначают отдельной нумерацией арабскими цифрами с добавлением перед цифрой обозначения приложения. Например – Рисунок А.3.

При ссылках на иллюстрации следует писать «... в соответствии с рисунком 1» или «на рисунке 1 приведены …».

Цифровой материал, как правило, оформляют в виде таблиц.

Таблицы применяют для лучшей наглядности и удобства сравнения показателей. Название таблицы, при его наличии, должно отражать ее содержание, быть точным, кратким. Название таблицы следует помещать над

таблицей слева без абзацного отступа в одну строку с ее номером, через тире.

Пример оформления таблицы:

Таблица 1 – Основные переменные программы

Идентификатор	Тип	Хранимые данные
n	int	Размерность массива
X[10]	float	Одномерный массив, содержащий исходные данные
i	int	Счетчик цикла

Таблицу, в зависимости от ее размера, помещают под текстом, где впервые дана ссылка на нее, либо на следующей странице, а при необходимости – в приложении к документу. На все таблицы должны быть ссылки в работе. При ссылке следует писать слово «таблица» с указанием ее номера. Таблицу с большим количеством строк допускается переносить на другой лист (страницу). При этом слово «Таблица» и ее номер указывают один раз слева над первой частью таблицы, над другими частями пишут справа слово «Продолжение» и указывают номер таблицы. Например – «Продолжение таблицы 1». При переносе части таблицы на следующую страницу название помещают только над первой частью таблицы, нижнюю горизонтальную черту, ограничивающую таблицу, не проводят.

Библиографическое описание составляется в соответствии с ГОСТ 7.1-2003, ГОСТ 7.82-2001, ГОСТ 7.80–2000. Список нумеруется арабскими цифрами без точки. Наименования в нем располагаются в алфавитном порядке.

Примеры оформления библиографических описаний:

1 Михайлов, М. Н. Имитационное моделирование / М. Н. Михайлов, Т. В. Первозванская, Л. П. Вьюненко. – СПб: Юрайт, 2017. – 284 с.

2 Смоленцев, Н. Г. MATLAB. Программирование на C++, C#, Java и VBA / Н. Г. Смоленцев. – Москва: ДМК Пресс, 2015. – 498 с.

3 Рекомендации по выбору оборудования для работы 1С: Предпри-

ятие 8 [Электронный ресурс]. – Режим доступа: http://v8.1c.ru/. – 10.03.2017.

Приложения оформляют как продолжение отчета, включаются в общую нумерацию страниц. В тексте документа на все приложения должны быть ссылки. Каждое приложение следует начинать с новой страницы с указанием наверху по центру страницы слова «ПРИЛОЖЕНИЕ», его обозначения. Приложение может быть с названием и без названия. Если приложение имеет заголовок, то он записывается симметрично относительно текста, с прописной буквы отдельной строкой. Приложения обозначают заглавными буквами русского алфавита.

Требования к содержательной части отчета

Текст отчета должен быть кратким, четким и не допускать различных толкований. В тексте не допускается наличие грамматических и синтаксических ошибок. Рекомендуется включать проверку правописания.

Форма титульного листа отчета приведена в приложении А.

Вторым листом отчета является индивидуальное задание практики, выданное руководителем практики от вуза.

Третий лист – календарный график прохождения практики.

На первых трех страницах отчета номер страницы не проставляется, но они входят в общую нумерацию документа.

Четвертым листом является содержание. Оно включает все разделы отчета: введение, наименование всех разделов, подразделов, заключение, библиографический список и наименование приложений с указанием номеров страниц, с которых начинаются эти элементы работы.

Слово «СОДЕРЖАНИЕ» записывается первой строкой страницы, выравнивание – по центру, заглавными буквами. Образец оформления содержания приведен в приложении Б.

Первым разделом является введение. Поскольку введение является самостоятельным разделом, его объем должен быть не менее одной страницы.

Во введении дается краткая характеристика современного состояния изучаемой проблемы, содержатся цели и задачи практики, а также исходные данные для выполнения задания.

Следующим раздел отчета должен содержать краткое изложение теоретических сведений, необходимых для выполнения индивидуального задания. Допускается деление раздела на подразделы. Наименование данного раздела и его подразделов соответствует излагаемому в нем материалу.

Второй раздел – выполнение индивидуального задания. Раздел разбивается на следующие подразделы: Описание программы, Описание основных переменных и функций исполняемого файла, Содержание проекта, Тестирование программы, Руководство пользователя и др.

Описание программы выполняется на основе ГОСТа 19.402 – 78. Здесь приводятся: общие сведения (обозначение и наименование программы, программное обеспечение, необходимое для функционирования программы, языки программирования, на которых написана программа, функциональное назначение, описание логической структуры программы, входные и выходные данные). Описание алгоритма может быть выполнено блок-схемой, оформленной в виде рисунка в соответствии с ГОСТом 19.701 – 90, и вынесено в приложение.

В подразделе «Тестирование программы» приводятся подготовленные тестовые значения исходных данных, а также прогнозируемые и полученные результаты работы приложения. Полученные результаты приводятся в виде скриншотов (снимков экрана).

Руководство пользователя (оператора) должно содержать: условие выполнение программы, запуск программы, пример ее выполнения и возможные сообщения, действия в случае сбоя.

Заключение должно быть кратким и лаконичным с перечислением всех этапов выполнения задания и результатов прохождения учебной практики.

Событийно-управляемое программирование

В основу работы операционных систем семейства Windows положен принцип *событийного управления.* Это означает, что все программные приложения, а также сама операционная система работают в режиме ожидания действий пользователя и реагируют на них заранее заданным образом. *Событием* называется любое действие пользователя, например, щелчок кнопкой мыши, перемещение ее курсора или нажатие клавиши на клавиатуре.

Событие воспринимается операционной системой, а затем преобразуется в *сообщение* – запись, содержащую необходимую информацию о произошедшем событии (какая клавиша была нажата, в каком месте экрана произошел щелчок мышью и т.п.). Сообщения могут поступать от пользователя, от самой операционной системы, от активного приложения или от других приложений.

Все сообщения поступают в общую очередь, затем распределяются по очередям приложений. Каждое приложение содержит *цикл обработки сообщений*, которое выбирает сообщение из очереди и через операционную систему вызывает подпрограмму, предназначенную для его обработки – *обработчик события*.

Таким образом, Windows-приложение состоит из главной программы и набора обработчиков событий.

Среда Visual Studio.NET содержит удобные средства разработки Windows-приложений, выполняющие вместо программиста рутинную работу создание шаблонов и заготовок обработчиков событий.

Среда разработки Visual Studio.NET

Платформу .Net корпорации Microsoft включает в себя средства для разработки программ на четырех языках программирования C++, J#, C#, VT.

Платформа является открытой, т.е. компиляторы для нее могут поставляться сторонними разработчиками.

Приложения в процессе разработки называются проекты. Проект объединяет в себе: файлы, папки, ссылки и прочие ресурсы.

Среда Visual Studio.NET позволяет создавать различные проекты:

- консольные приложения, осуществляющие ввод-вывод в окно командного процессора;

- Windows-приложения, использующие элементы интерфейса Windows: кнопки, формы, события и пр.;

- веб-приложения, формирующие веб-страницу, доступ к которой осуществляется через браузер;

- библиотеки классов, объединяющие классы для дальнейшего их использования в других приложениях;

- веб-сервисы – компоненты, методы которых вызываются через Интернет.

Компоненты Windows Forms

Windows Forms – интерфейс программирования приложений (API), отвечающий за графический интерфейс и упрощающий доступ к элементам интерфейса Microsoft Windows. Создаваемый с их помощью управляемый код не зависит от языка разработки, т.е. программист одинаково может использовать Windows Forms в приложениях на других языках программирования: C#, VB.Net, J# и др.

Краткие описания основных компонент:

Action List (список действий) осуществляет управление взаимодействием между интерфейсными элементами и логикой всей программы.

Button (кнопка) служит для создания в приложение различных прямоугольных кнопок с текстовой однострочной надписью.

Checkbox (ячейка состояния) позволяет создавать на экранной форме

приложения ячейку с двумя состояниями (без галочки и с галочкой) и со строкой, содержащей их названия. Щелчок левой кнопкой мыши по этому компоненту во время работы программы вызывает изменение состояния данного компонента на противоположное. В программе можно узнать состояние компонента и в зависимости от его значения задавать выполнение различных действий.

ComboBox (комбинированный список) позволяет создавать элемент, являющийся комбинацией строки ввода и выпадающего списка для выбора, т.е. объединяя в себе компоненты редактируемого поля (ListBox) и списка (Edit).

Edit (редактирование) задает однострочное редактируемое окно для ввода-вывода данных.

GroupBox (окно группы) служит для создания области, визуально объединяющей на форме несколько интерфейсных элементов.

Label (этикетка) создает на форме текстовую метку или надпись.

ListBox (окно списка) отвечает за создание прямоугольного поля для отображения текстовых строк с возможностью их выбора, добавления или удаления в ходе работы программы.

GroupBox (окно группы) служит для создания области, визуально объединяющей на форме несколько интерфейсных элементов.

Memo отображает на форме многострочное поле ввода-вывода данных. Обычно служит для создания редакторов и полей, для вывода блоков данных.

Panel (панель) создает пустую область, на которой размещаются другие компоненты. Как правило, она используется для создания панели инструментов.

PopupMenu (всплывающее меню) предназначено для создания контекстного меню.

RadioButton (радиокнопка) создает круглое поле с двумя состояниями (с точкой и без точки) и с текстовой строкой, поясняющей ее назначение в

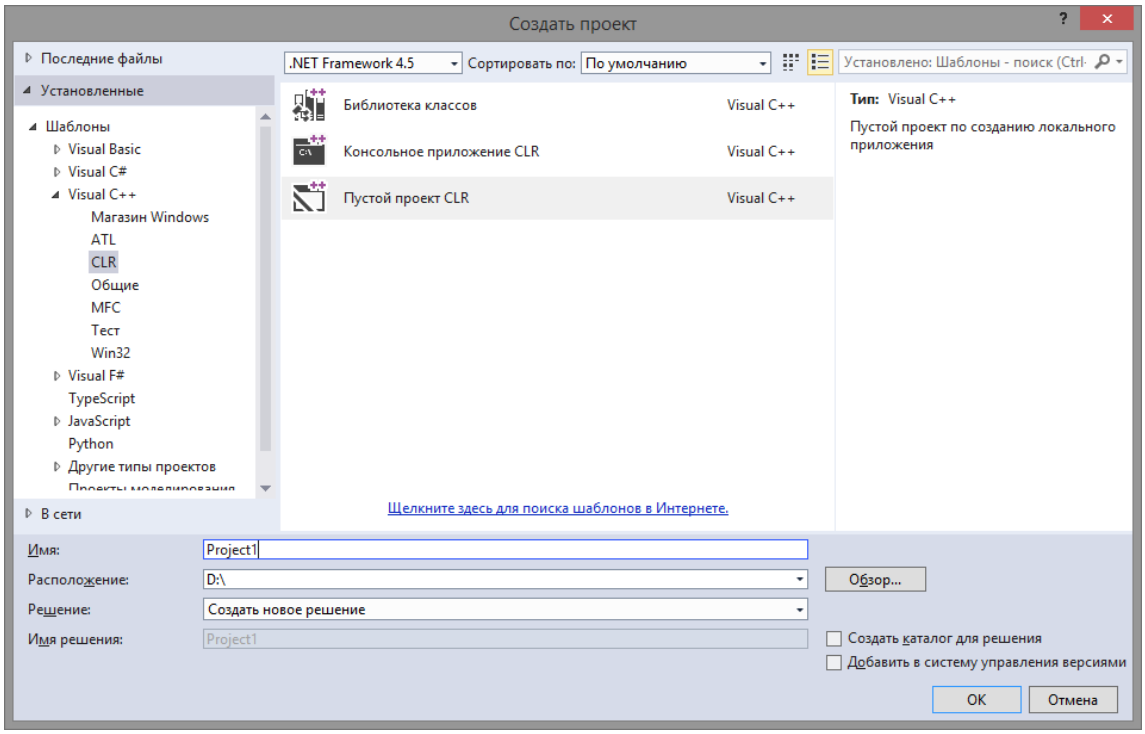
программе. Обычно несколько таких компонентов, расположенных на форме, позволяют переключить только один элемент из группы. Изменить состояние можно только для одного из этих компонентов, т. к. остальные компоненты переключают при этом свое состояние автоматически.

RadioGroup (группа радиокнопок) позволяет создавать на форме контейнер в виде прямоугольной рамки для объединения группы взаимно исключающихся радиокнопок.

ScrollBar (линейка прокрутки) создает элемент, похожий на линейку с бегунком и кнопками для прокрутки окна. С его помощью можно изменять значение какого-либо параметра в пределах некоторого заданного интервала.

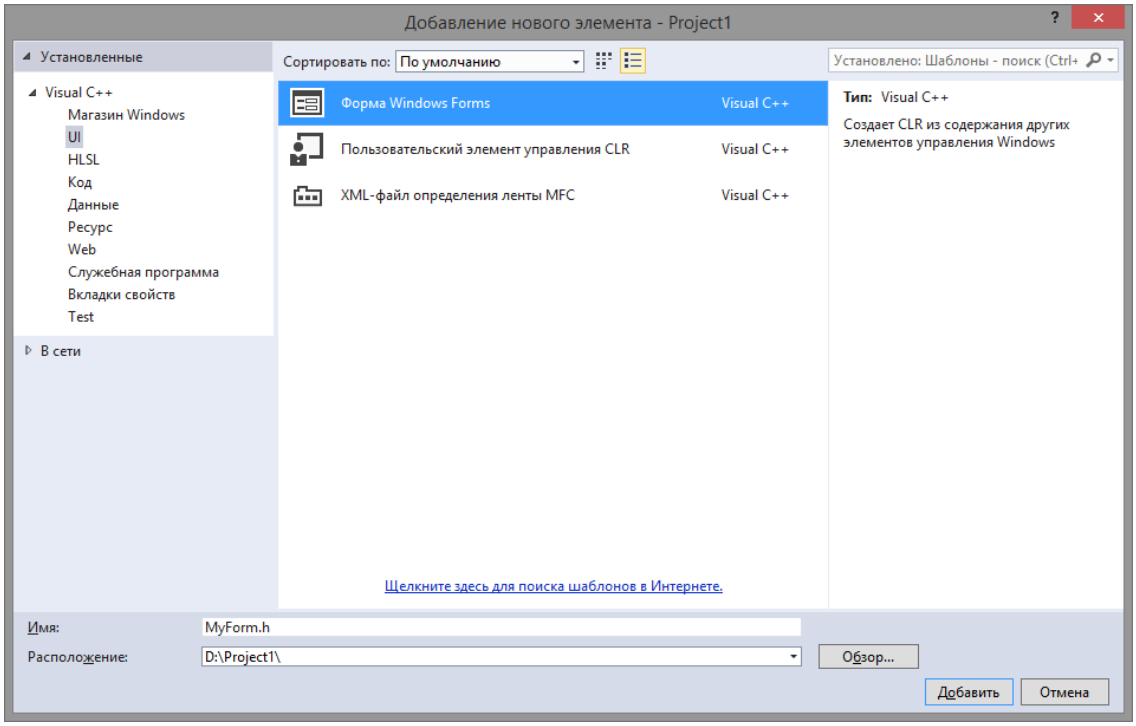
Создание проекта Windows Forms в Visual Studio на C++

Для создания Windows-приложения в среде Visual Studio на языке C++ необходимо выполнить последовательность *Файл → Создать → Проект*. В открывшемся окне осуществить выбор типа проекта: в разделе *Visual C++* подраздел *CLR*, а затем – пункт *Пустой проект CLR*.



После создания проекта в обозревателе решений требуется кликнуть по нему правой кнопкой мыши. В открывшемся контекстном меню выбрать:

Добавить → Создать элемент, а затем в разделе *UI* открывшегося меню: *Форма→Windows→Forms.*



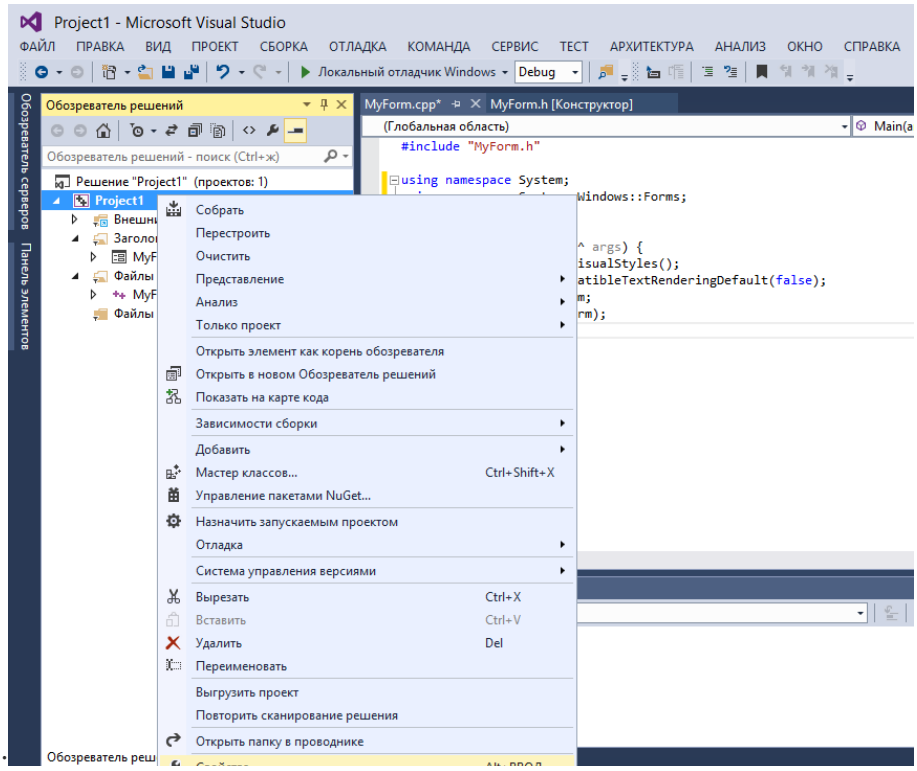
Когда форма будет добавлена, в обозревателе решений нужно выбрать файл MyForm.cpp, откроется новая вкладка с единственной строчкой кода:

```
#include "MyForm.h"
```

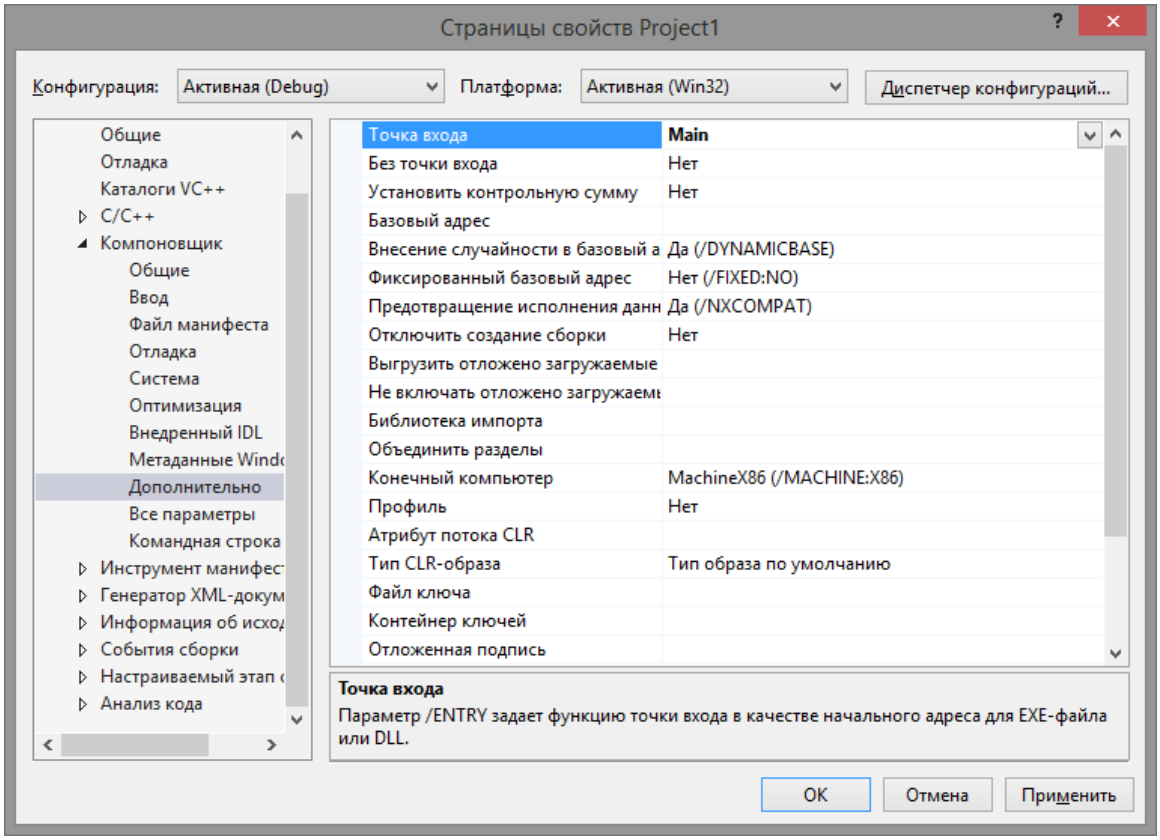
К этому коду следует добавить:

```
using namespace System;
using namespace System::Windows::Forms;
[STAThread]
void Main(array<String^>^ arg) {
    Application::EnableVisualStyles();
    Application::SetCompatibleTextRenderingDefault(false);
    Project1::MyForm form; //Project1 - имя вашего проекта
    Application::Run(%form);
    }
```

После этого в свойствах проекта выбрать подраздел *Система раздела →Компоновщик* и в строке *Подсистема* из выпадающего меню: *Windows (/SUBSYSTEM:WINDOWS),* нажать *Применить.*



Не закрывая окно свойств проекта, нужно перейти в подраздел *Дополнительно* и в строке *Точка входа* ввести **Main** и нажать **ОК.**



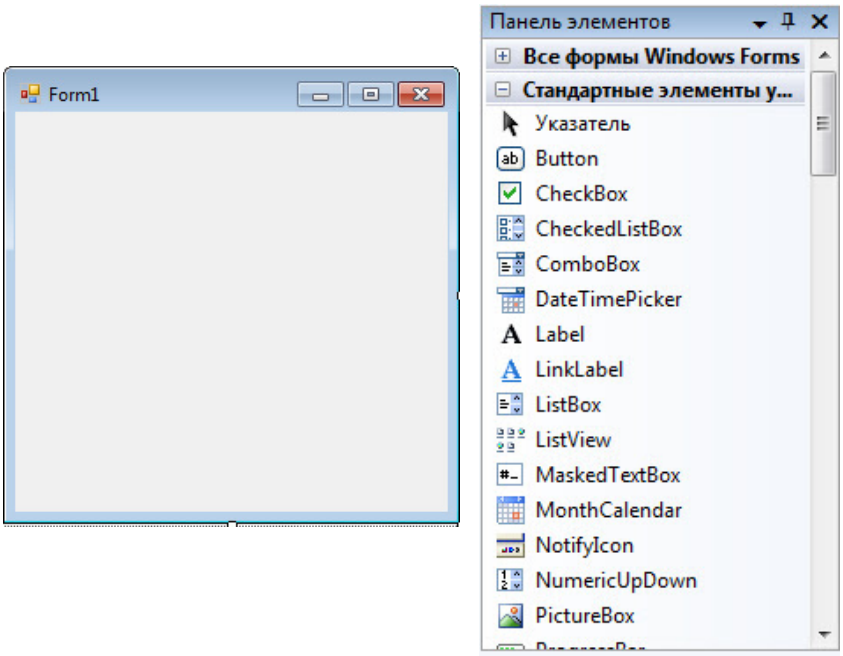
На этом настройки проекта завершены. Для редактирования внешнего

вида формы, необходимо перейти во вкладку *MyForm.h* [Конструктор], кликнув дважды по файлу *MyForm.h* в обозревателе решений.

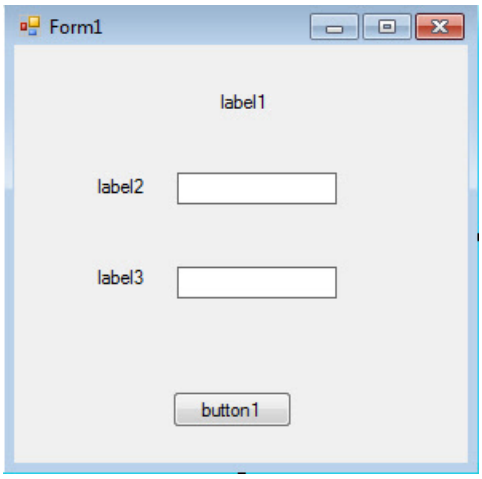
Пример создания Windows-приложения

После создания проекта на экране должна появиться пустая форма: Справа от нее располагается панель элементов. В случае ее отсутствия включение возможно с помощью меню **Вид → Панель Элементов** или комбинацией клавиш: **Ctrl + Alt + X**.

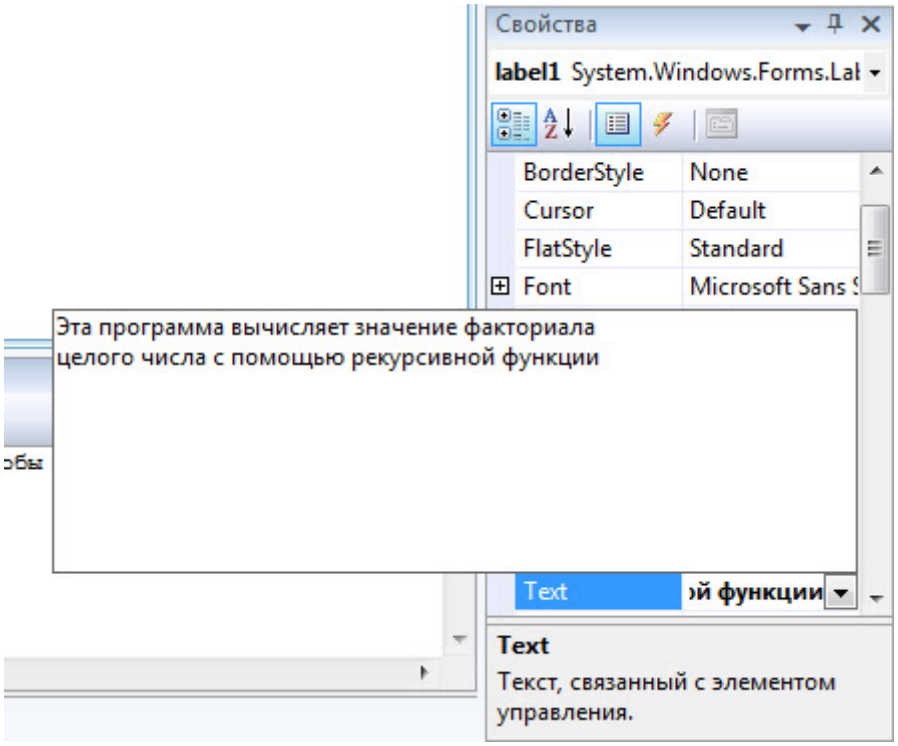
На панели элементов расположены элементы Windows Forms.



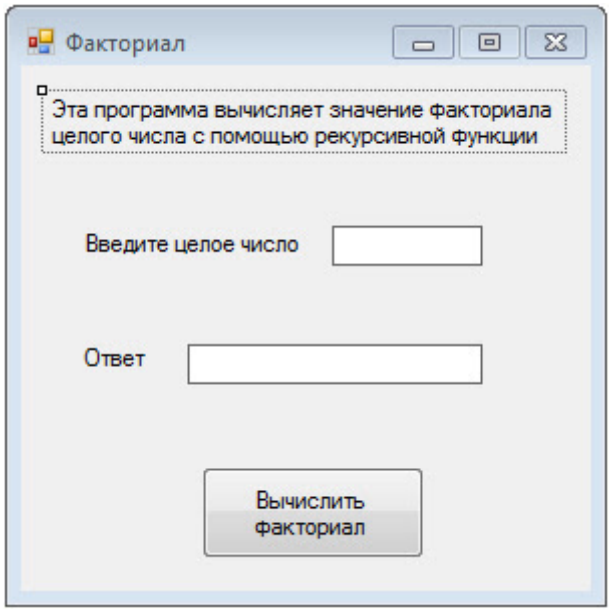
В данном случае понадобятся: надпись **(Label),** два текстовых поля (**TextBox)** и кнопка (**Button)**. Перетащив перечисленные элементы на форму, можно получить ее в виде:



Для изменения текста надписей элементов нужно выделить элемент и перейти в **Панель свойств**, которая, как правило, располагается под **Панелью элементов.** Если панель выключена, ее можно включить, выбрав в меню **Вид** -> **Диспетчер свойств** и задать для данного элемента значение атрибута **Text,** как показано на рисунке.



Задав атрибут **Text** для всех элементов формы, можно получит ее в виде:



Затем следует задать атрибут **Name** в **Панели Свойств** для текстовых полей (**TextBox**) . Для первого поля пусть будет **x**, а для второго – **y**. Форма готова.

Для добавления заголовочного файла – **fact.h** – в проект, нужно кликнуть правой кнопкой мыши в **Обозревателе решений** на папке **Заголовочные файлы**, далее в меню: **Добавить** -> **Создать Элемент** и вписать название файла – **fact.h (вкладка код-> заголовочный файл)**, затем выполнить **Добавить**.

В файле будет содержаться прототип функции вычисления факториала:

```
long double fact(int N);
```

Далее требуется подключить этот файл к проекту с помощью директивы **#include** – открыть файл **fact.cpp** и добавить строку кода:

```
#include "fact.h"
```

Затем в файл исходного кода **fact.cpp** добавляется и сама функция вычисления факториала:

```
#pragma once
long double fact( int N)
{
  // если пользователь ввел отрицательное число
   if (N < 0)           return 0;      // возвращаем ноль
   // если пользователь ввел ноль
   else  if (N == 0)  return 1;  // возвращаем факториал нуля – единицу
   // во всех остальных случаях выполняем рекурсию
    else   return N * fact(N - 1);
}
```

После компиляции кода, можно приступить к написанию обработчика событий кнопки **Button**. Открыв файл **Form.h**, щелкнуть дважды по кнопке **Button** в визуальном представлении формы и перейти к исходному коду.

Необходимо помнить, что любой набор символов, вводимый с клавиатуры в текстовое поле программы, является строкой. Но программа работает с числами. Поэтому значение из текстового поля нужно привести к целочисленному типу, воспользовавшись одной из функций конвертирования(преобразования) типов данных. Для этого нужно объявить и инициализировать переменную для хранения значение числа, для которого будет вычисляться значение факториала:

int number = System :: Convert :: ToDouble (x–>Text);

Затем вызвать функцию **fact** с переменной **number** в качестве фактического параметра. Результат работы функции передается в переменную **factor** следующим кодом:

double factor = fact(number);

Затем следует выполнить обратное преобразование – полученное значение факториала в строку – и присвоить его второму текстовому полю:

y –> Text = System :: Convert :: ToString (factor);

В итоге, обработчик события для компонента Button готов. Скомпилировав и запустив программу, можно проверить ее работоспособность.

Обработка исключительных ситуаций

Исключительная ситуация (исключение) – возникновение непредвиденного или аварийного события, которое может порождаться некорректным использованием аппаратуры. Например, деление на нуль или обращение по несуществующему адресу памяти. C++ позволяет восстанавливать программу после таких ошибок и продолжить ее выполнение.

В C++ исключения не поддерживают обработку асинхронных событий, например обработку прерываний или сбои работы оборудования. Механизм исключений предназначен только для событий, происходящих в самой программе.

Исключения позволяют разделить процесс вычислений на две части:

обнаружение исключительной ситуации, и ее обработка. Это позволяет лучше структурировать программу. Другое достоинство использование исключений заключается в передачи информации об ошибке в вызывающую функцию без возвращающего значения, параметров или глобальных переменных.

Место, в котором может произойти ошибка, должно входить в контролируемый блок – составной оператор, перед которым указывается ключевое слово try.

Общий механизм обработки исключительной ситуации заключается в следующих шагах:

1. Появление ошибки. Функция, в которой она возникла, генерирует исключение. Для этого используется ключевое слово throw с параметром, определяющим вид исключения. Параметр может быть константой, переменной или объектом. Он используется для передачи информации об исключении его обработчику.

2. Отыскивается соответствующий обработчик исключения и ему передается управление программой.

3. Если обработчик не найден, вызывается стандартная функция terminate, которая вызывает функцию abort, аварийно завершающую текущий процесс. Можно также установить собственную функцию завершения процесса.

При вызове каждой функции в C++ в стеке создается область памяти, предназначенная для хранения локальных переменных и адреса возврата в вызывающую функцию. Термин стек вызовов обозначает последовательность вызванных, но еще не завершившихся функций. Раскручиванием стека называется процесс освобождения памяти из-под локальных переменных и возврата управления вызывающей функции. Тот же механизм используется и при обработке исключений. Поэтому после обнаружения исключения исполнение может быть продолжено с точки его генерации.

Синтаксис исключения:

try {

контролируемый блок

}

Все функции, прямо или косвенно вызываемые из try-блока, также считаются принадлежащими ему.

Генерация исключения:

throw [выражение];

тип выражения определяет тип порождаемого исключения.

При генерации исключения выполнения текущего блока прекращается, и происходит поиск соответствующего обработчика и передача ему управления.

Не всегда исключение, возникшее во внутреннем блоке, может быть сразу обработано правильно. В этом случае используются вложенные контролируемые блоки, и исключение передается на более высокий уровень с помощью throw.

Обработчики исключений начинаются с ключевого слова catch, за которым в скобках следует тип обрабатываемого исключения. Они должны непосредственно располагаться за try-блоком. Их синтаксис аналогичен определению функции с одним параметром.

В качестве параметра может стоять многоточие, что означает, что обработчик перехватывает все исключения.

После обработки исключения управление передается первому оператору, находящемуся непосредственно за обработчиками исключений. Туда же, минуя все код всех обработчиков, передается управление, если в try-блоке не было сгенерировано

Обработчики производных классов следует располагать до обработчиков базовых, иначе им никогда не будет передано управление.

Обработчик типа void автоматически скрывает указатель любого другого типа, поэтому его также следует размещать после обработчиков указа-

телей конкретного типа.

```
#include <fstream.h>
#include<iostream.h>
class Hello{
//информирует о своем создании и уничтожении
public:
Hello( ) {cout<<”Hello!”<<endl;}
~Hello( ) {cout<<”Bye!”<<endl;}
};
void f1( ){
ifstream ifs( “name”);
if (!ifs) {
cout<< “генерируем исключение”<<endl;
throw “Ошибка при открытии файла”;
}
void f2( ) {
Hello h; //создаем объект
f1( ); // вызываем функцию, генерирующую исключение
}
int main( ) {
try {
cout<<”входим в try-блок”<<endl;
f2( );
cout<<”выходим в try-блок”<<endl;
}

catch (int i) {
cout<< “вызван обработчик int, исключение -”<< i << endl;
return -1;
```

```
}
catch ( const char *p) {
cout<< “вызван обработчик const char*, исключение -”<< p << endl;
return -1;
}
catch ( …) {
cout<< “вызван обработчик всех исключений”<<  endl;
return -1;
}
```

Результат выполнения

входим в try-блок

Hello!

Генерируем исключение

Bye!

вызван обработчик const char*, исключение - Ошибка при открытии файла.

Таким образом, механизм исключений позволяет корректно уничтожать объекты при возникновении ошибочных ситуаций.

Поэтому выделение и освобождение некоторого ресурса полезно оформлять в виде классов, конструктор которых выделяет ресурс, а деструктор освобождает. Например, конструктор может открывать файл, а деструктор его закрывать. В этом случае будет гарантия, что при возникновении ошибки чтения файла, он будет корректно закрыт, и информация не будет утеряна.

Класс для представления исключения можно описать внутри класса, при работе с которым оно может возникнуть. Конструктор копирования в этом случае обязательно должен быть public, иначе будет невозможным создать копию объекта при генерации исключений.

Стандартная библиотека C++ содержит ряд функций обработки исключений. Все они являются производными от библиотечного класса exprection, описанного в заголовочном файле stdexpect.

Для обработки ошибок при выделении памяти в библиотеке описан класс bad_alloc, тип функций-обработчиков ошибок new_handler и функция установки нового обработчика set_new_handler.

Механизм обработки ошибок выделения памяти следующий: если операция new не может выделить требуемое количество динамической памяти, она вызывает функцию обработчик типа new_handler, который должен либо попытаться освободить память, либо породить исключение в виде объекта класса bad_alloc или производного от него, либо вызвать функцию abort или exit для завершения программы.

Другие функции

invalid_argument – попытка вызова функции с неверным параметром

ranger_error – неверный результат выполнения

overflow_error – арифметическое переполнение.

Шаблоны классов

Шаблоны классов поддерживают парадигму обобщенного программирования, т.е. программирования с использованием типов в качестве параметров. Механизм шаблонов в C++ допускает применение абстрактного типа в качестве параметра при определении класса или функции.

В дальнейшем шаблон класса может быть использован для создания классов. Процесс генерации компилятором определения конкретного класса по шаблону класса и аргументам шаблона называется инстанцированием (актуализацией) шаблона.

Определение шаблона класса имеет следующий синтаксис:

template <параметры шаблона> class имя класса {тело} ;

Параметры шаблона перечисляются через запятую. В их качестве мо-

гут использоваться не только типы и переменные, но и шаблоны.

Типы, используемые в шаблонах, могут быть как встроенными, так и определенные пользователем. Внутри шаблона параметр типа может применяться в любом месте, где допустимо в дальнейшем использовать спецификацию типа.

Пример. Для представления точки на плоскости разработан класс Point, в котором координаты задаются двумя числами типа double. А в другом приложении требуется задать точки для целочисленной системы координат (тип int). Поэтому сначала объявлен шаблон класса.

```
template <class T> class Point {
private:    T x, y;
public:     Point (T a, T b) : x(a), y(b) {}
            void show( ) {cout<< "("<<x<<","<<y<<")"<<endl; }
};
```

Префикс template <class T> означает, что объявлен шаблон класса, в котором T – некоторый абстрактный тип. T – параметр шаблона. Вместо T может использоваться любое имя типа. После своего объявления T используется внутри шаблона, аналогично именам обычных типов.

Вместо template <class T> class Point можно писать конструкции вида template <typename T> class Point, но первый вариант считается более распространенным.

При создании шаблона класса в программе генерация классов не происходит немедленно. Для этого нужно создать экземпляр шаблонного класса, который создается либо объявлением объекта Либо объявлением указателя на актуализированный шаблонный тип с присваиванием ему адреса, возвращаемого операцией new.

```
Point <int> anyPoint(-13, 5);
Point <double> otherPoint=new Point<double>(-13.56789, 5.65478);
```

Возможно вынесение определений шаблона в отдельный файл, а затем

его подключение к программе директивой препроцессора.

Для создания шаблона для массивов из n элементов возможна следующая конструкция.

template <class T, int n> class Array {…}

Актуализация данного шаблона:

Array<Point, 20> array; // массив из 20 элементов

В этом случае параметры class T, int n могут рассматриваться как формальные параметры шаблона, на место которых при компиляции встанут конкретные значения.

Методы шаблона автоматически становятся шаблонами функций. Если метод описывается вне шаблона, его заголовок должен иметь следующие элементы:

template <описание параметров шаблона>

тип имя класса <параметры шаблона>:: имя функции (параметры)

Правила описания шаблонов:

Локальные классы не могут содержать шаблоны в качестве своих элементов;

шаблоны методов не могут быть виртуальными;

шаблоны классов не могут содержать статические элементы, дружественные функции и классы;

шаблоны могут быть производными как от шаблонов, так и от обычных классов, а также, в свою очередь являться базовыми для шаблонов и обычных классов;

внутри шаблонов нельзя объявлять дружественные шаблоны.

Шаблоны являются мощным и эффективным средством обращения с различными типами данных. К недостатком использования шаблонов можно отнести то, что программа с шаблонами должна содержать полный код для каждого порождаемого типа, что значительно увеличивает размер исполняемого файла.

Стандартная библиотека C++ содержит достаточно большой набор шаблонов.

Контейнерные классы

Контейнерные классы – это классы, предназначенные для хранения данных, организованных определенным образом. К ним относятся массивы, линейные списки, стеки и другие. Для каждого типа контейнера определены методы работы с его элементами, независимо типа элементов данных, хранимых в контейнере. Поэтому один и тот же вид контейнера можно использовать для хранения данных различных типов. Эта возможность реализована стандартной библиотекой шаблонов (STL – Standard Template Library) языка C++.

Использование контейнеров позволяет повысить надежность, универсальность, переносимость программ, уменьшить время их разработки, но за это приходится расплачиваться снижением их быстродействия.

Все контейнеры делятся на два класса: последовательные и ассоциативные. Последовательные – обеспечивают хранение конечного количества однотипных величин в виде непрерывной последовательности. К ним относятся векторы (vector), двусторонние очереди (deque), списки (list), стеки (stack) и очереди с приоритетами (priority_queue). Шаблоны указанных контейнеров хранятся в одноименных библиотеках языка C++.

Ассоциативные контейнеры обеспечивают быстрый доступ к данным по ключу. Они построены на основе сбалансированных деревьев. Это словари (map), словари с дубликатами (multimap), множества (set), множества с дубликатами (multiset), битовые множества (bitset).

На основе контейнеров стандартной библиотеки можно создавать собственные контейнерные классы.

Контейнерные классы обеспечивают стандартизированный интерфейс их применения. Смысл одноименных операций для различных контейнеров

одинаков. Стандарт определяет только интерфейс контейнеров, поэтому их реализации могут отличаться.

Практически в любом контейнерном классе определены следующие поля указанных типов:

value_type	тип элемента контейнера
size_type	тип индексов, счетчиков элементов и т.д.
iterator	итератор
const_iterator	константный итератор
reverse_iterator	обратный итератор
const_reverse_iterator	константный обратный итератор
reference	ссылка на элемент
const_reference	константная ссылка на элемент
key_type	тип ключа (для ассоциативных контейнеров)
key_compare	тип критерия сравнения (для ассоциативных контейнеров)

Термин «итератор» является аналогом указателя на элемент. Он может применяться для прохода по элементам контейнера в прямом и обратном направлениях. Когда значения элементов контейнера не предполагается изменять применяют константные итераторы.

В помощь для работы с итераторами определено несколько методов:

iterator begin( ) const_iterator begin( ) const	указывают на первый элемент
iterator end( ) const_iterator end( ) const	указывают на элемент за последним
reverse_iterator rbegin( ) const_reverse_iterator rbegin() const	указывают на первый элемент в обратной последовательности
reverse_iterator rend( ) const_reverse_iterator rend( ) const	указывают на элемент, следующий за последним в обратной последовательности

В каждом контейнере эти типы и методы определяются способом, зависящим от их реализации.

Во всех контейнерах определены методы, позволяющие получить сведения о размере контейнеров:

size( ) – число элементов,

max_size( ) – максимальный размер контейнера,

empty( ) – булевская функция, отвечающая на вопрос: пуст ли контейнер.

Последовательные контейнеры

Вектором называют структуру, которая позволяет обеспечить эффективный произвольный доступ к своим элементам, добавление и удаление из конца структуры.

Двусторонняя очередь в дополнение к вектору разрешает удаление и добавление с обеих ее сторон.

Список эффективно реализует вставку и удаление элементов в произвольное место своей структуры, но не обеспечивает доступа к этим элементам. Операции последовательных контейнеров представлены в таблице.

Операция	Метод	vector	deque	list
Вставка в начало	push_front	-	+	+
Удаление из начала	pop_front	-	+	+
Вставка в конец	push_back	+	+	+
Удаление из конца	pop_back	+	+	+
Вставка в произвольное место	insert	+	+	+
Удаление из произвольного места	erase	+	+	+
Произвольный доступ к элементу	[ ], at	+	+	-

Пример 1. Программа считывает и выводит на экран ряд целых чисел, хранящийся в файле.

```
#include <fstream.h>
#include <vector>
int main( ) {
ifstream in (“primer.txt”);
vector <int> v;
int x;
while (in.eof( ) )
in>>x;
v.push_back(x);
for (vector<int>::iterator i=v.begin(); i!=v.end(); ++i)  cout<<*i<< “ “;
}
```

В примере для создания вектора используется конструктор по умолчанию. Можно пользоваться и другими конструкторами:

```
explicit vector(); // конструктор по умолчанию
explicit vector(size_type n, const T& value=T()); //создается вектор длины n и заполняется одинаковыми элементами – копиями значения value
vector (const vector<T>& x); //конструктор копирования
```

Ключевое слово explicit используется, чтобы при создании объекта исключить выполнение неявного преобразования при присваивании значения другого типа. Например,

```
vector <int> v2(10, 1); // вектор из 10 элементов равных единице
vector <int> v4 (v2); // создается вектор v4 равный v2
```

Пример 2.

```
#include <vector>
using namespase std;
int main() {
```

```
double arr[]= {1.1, 2.2, 3.3, 4.4 };
int n=sizeof(arr)/sizeof(doule);
vector<double> v1(arr, arr+n);
vector<double> v2;    //пустой вектор
v1.swap(v2);          //обменять содержимым v1 и v2
while (!v2.empty() )
        {
        cout<<v2.back()<< ' '; //вывод последнего элемента
        v2.pop_back();      //удаление элемента
        }
return 0;
}
```

Пример 3

```
#include <iostream>
#include <vector>
using namespace std;
int main()
{
  vector <int> vecInt(3,10);
//создается  вектор из 3 элементов и заполняется  значением,  равным 10
  vector <int>:: iterator it;
  it = vecInt.begin();    //итератор указывает на элемент vec[0]
   //Вектор расширяется  до 4 элементов
  vecInt.insert (it, 20); // в начало  вектора  записывается 20
   // вектор расширяется до 5 элементов
  it = vecInt.begin() + 3;
//итератор  указывает на 4 элемент (нулевой  элемент+3)
  vecInt.insert(it, 30);
```

```
// четвертым элементом становится 30
   vecInt.insert(it+1, 90);
//функция вставляет значение второго параметра на место, указанное первым
//вектор расширяется  до 6 элементов и 5 элементом записывается 90
    cout << "Vector contains: ";
   for (int i=0; i<vecInt.size(); ++i)
   {
      cout << vecInt[i] << ends;                              //20 10 10 30 90 10
   }
   cout << "\nVector max_size: " << vecInt.max_size();
   cout << "\nVector size: " << vecInt.size();    // размер вектора равен  6
   vecInt.resize(10);          //увеличиваем размер вектора до 10 элементов
   cout << "\nNew vector size: " << vecInt.size() << endl;
   return 0;
}
```

ВАРИАНТЫ ИНДИВИДУАЛЬНОГО ЗАДАНИЯ

Общая часть задания: написать Windows-приложение, заголовок главного окна которого содержит: Ф. И. О., номер группы и номер варианта индивидуального задания.

Создать меню с командами Input, Calc и Exit.

При выборе команды Input открывается диалоговое окно, содержащее:

- поле типа TextBox для ввода параметра начального значения для задания последующих элементов массива;

- поле типа TextBox для ввода параметра счетчика случайных чисел;

- группу из флажков, соответствующих результатам расчетов согласно варианту задания, типа CheckBox;

- поле типа TextBox для вывода элементов массива;

кнопку типа Button.

Обеспечить возможность:

ввода начальных значений;

вывода полученных значений массива;

выбора режима расчетов с помощью флажков.

При выборе команды Calc обеспечить открытие диалогового окна с результатами расчетов.

При выборе команды Exit обеспечить завершение приложения.

В программе использовать проверку возникновения следующих исключительных ситуаций: введение буквы в качестве входного параметра, отсутствие выбора в группе флажков.

При работе с элементами массива использовать последовательный контейнер Vector.

Вариант 1

Элементы массива задать по следующему правилу: введенный параметр является первым и последним элементами массива, остальные значения

задаются с помощью счетчика случайных чисел.

В массиве, содержащем 14 элементов, определить произведение его положительных элементов, расположенных до максимального элемента этого массива и отсортировать по возрастанию элементы, расположенные после максимального.

Вариант 2

Элементы массива задать по следующему правилу: введенный параметр является центральным элементом массива, каждый следующий после него в два раза меньше предыдущего, остальные значения задаются с помощью счетчика случайных чисел.

В массиве, содержащем 17 элементов, определить сумму положительных элементов, расположенных между минимальным и максимальным элементами данного массива и отсортировать по возрастанию.

Вариант 3

Элементы массива задать по следующему правилу: введенный параметр является последним и центральным элементами массива, остальные значения задаются с помощью счетчика случайных чисел.

Если положительных элементов массива, содержащего 25 элементов, больше чем отрицательных, то отсортированные по убыванию положительные элементы расположить в начале массива. Иначе в начале массива расположить отсортированные отрицательные элементы.

Вариант 4

Элементы массива задать по следующему правилу: введенный параметр является первым элементом массива, следующие семь элементов по правилу: каждый последующий в два раза больше предыдущего, остальные значения задаются с помощью счетчика случайных чисел.

Если номер минимального элемента массива, содержащего меньше 18, отсортировать по возрастанию элементы, расположенные после него, иначе отсортировать элементы с 8 по 18 номер. Минимальный элемент уменьшить

в два раза.

Вариант 5

Элементы массива задать по следующему правилу: введенный параметр является последним элементом массива, остальные элементы задаются с помощью счетчика случайных чисел.

Если номер максимального элемента массива из 25 значений больше 15, то отсортировать по убыванию элементы массива, расположенные до максимального элемента, иначе отсортировать элементы данного массива, расположенные за максимальным элементом. Максимальный элемент увеличить в три раза.

Вариант 6

Элементы массива задать по следующему правилу: введенный параметр является первым и последним элементами массива, остальные элементы задаются с помощью счетчика случайных чисел.

Три элемента, расположенные в центре массива из 25 значений, оставить без изменения, остальные отсортировать по возрастанию. В отсортированном массиве заменить первые пять элементов нулями.

Вариант 7

Элементы массива задать по следующему правилу: введенный параметр является первым и последним элементами массива, остальные элементы задаются с помощью счетчика случайных чисел.

Если номер минимального элемента массива из 20 значений меньше пяти, отсортировать по убыванию элементы, расположенные до него по возрастанию, иначе после него по убыванию. В отсортированном массиве каждый второй элемент обнулить.

Вариант 8

Элементы массива задать по следующему правилу: введенный параметр является центральным элементом массива, остальные элементы задаются с помощью счетчика случайных чисел.

В массиве из 19 элементов, определить номер первого положительного элемента из расположенных после минимального и отсортировать элементы между ними по возрастанию.

Вариант 9

Элементы массива задать по следующему правилу: введенный параметр является последним элементом массива, первые три элемента в десять раз больше своих индексов, остальные элементы задаются с помощью счетчика случайных чисел.

В массиве из 15 элементов, отсортировать их по убыванию и найти сумму элементов, расположенных после минимального и стоящих на нечетных местах.

Вариант 10

Элементы массива задать по следующему правилу: введенный параметр является первым элементом массива, последние три элемента в десять раз больше своих индексов, остальные элементы задаются с помощью счетчика случайных чисел.

В массиве, содержащем 14 элементов, определить произведение элементов, удовлетворяющих условию a<c[i]<b и расположенных до максимального элемента этого массива и отсортировать их по убыванию. Значения переменных a и b вводить с клавиатуры.

Вариант 11

Элементы массива задать по следующему правилу: введенный параметр является первым элементом массива, последние пять элементов в три раза больше своих индексов, остальные элементы задаются с помощью счетчика случайных чисел.

Массив из 20 элементов отсортировать следующим образом: сначала расположить все положительные в порядке убывания, затем отрицательные элементы в порядке возрастания, в конце – все элементы, равные нулю.

Вариант 12

Элементы массива задать по следующему правилу: введенный параметр является первым элементом массива, последние три элемента в пять раза больше своих индексов, остальные элементы задаются с помощью счетчика случайных чисел.

Если максимальный элемент вектора из 18элементов расположен до минимального отсортировать по возрастанию элементы расположенные после него, иначе до него. Минимальный элемент увеличит в десять раз.

Вариант 13

Элементы массива задать по следующему правилу: введенный параметр является первым и последним элементами массива, три центральных элемента равны нулю, остальные элементы задаются с помощью счетчика случайных чисел.

Вывести на экран сначала нулевые элементы массива из 29 элементов, а за ними отсортированные по убыванию ненулевые элементы данного массива.

Вариант 14

Элементы массива задать по следующему правилу: введенный параметр является первым и последним элементами массива, три центральных элемента равны нулю, остальные элементы задаются с помощью счетчика случайных чисел.

Первые пять элементов массива из 23значений оставить в неизменном порядке, следующие 10 отсортировать в порядке возрастания, оставшиеся отсортировать в порядке возрастания.

Вариант 15

Элементы массива задать по следующему правилу: введенный параметр является последним элементом массива, три центральных элемента равны нулю, остальные элементы задаются с помощью счетчика случайных чисел.

Элементы массива размерности 20 отсортировать следующим образом: сначала расположить все положительные в порядке убывания, затем отрицательные элементы в порядке возрастания, в конце – все элементы, равные нулю.

ПРИЛОЖЕНИЕ А

Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
АМУРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
(ФГБОУ ВО «АмГУ»)

Факультет математики и информатики

Кафедра информационных и управляющих систем

Направление подготовки 09.03.01 – Информатика и вычислительная техника

Направленность программы Автоматизированные системы обработки информации и управления

ОТЧЕТ ПО УЧЕБНОЙ ПРАКТИКЕ

Исполнитель

студент группы 653об _____ В.В. Сидоров
(подпись, дата)

Руководитель

доцент, канд. техн. наук _____ А.В. Иванов
(подпись, дата)

Благовещенск 2017

ПРИЛОЖЕНИЕ Б

СОДЕРЖАНИЕ

ЛИТЕРАТУРА

Галаган, Т.А. Алгоритмические языки и программирование. Язык С++. Курс лекций (Рек. ДВРУМЦ) / Т.А. Галаган – Благовещенск: изд-во АмГУ, 2007. – 147 с.

Галаган, Т.А. Объектно-ориентированное программирование. Язык С++ . Учебное пособие для бакалавров направления подготовки 38.03.05 – «Бизнес-информатика». – Благовещенск: Амурский гос. ун-т, 2016

Лаптев, В.В. С++ Объектно-ориентированное программирование: задачи и упражнения (Допущено Мин. обр. РФ) / В.В. Лаптев, А.В. Морозов, А.В. Бокова – СПб.: Питер, 2007. – 281 с.

Павловская, Т.А. С/С++. Программирование на языке высокого уровня (Допущено МинОбр РФ) – СПб.: Питер, 2009. – 461 с.

Павловская, Т.А. С++. Объектно-ориентированное программирование. Практикум (Допущено МинОбр РФ) / Т.А. Павловская, Ю.А. Щупак. – СПб.: Питер, 2004. 265 с.

Татьяна Алексеевна Галаган,
доцент кафедры ИиУС АмГУ

Учебная практика. Учебно-методическое пособие для бакалавров направления подготовки 09.03.01 – «Информатика и вычислительная техника»

Изд-во АмГУ.