Федеральное государственное бюджетное образовательное учреждение высшего образования
«ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ ИМЕНИ ДВАЖДЫ ГЕРОЯ СОВЕТСКОГО СОЮЗА, ЛЕТЧИКА-КОСМОНАВТА А.А. ЛЕОНОВА»

Г.Н. ИСАЕВА
Н.В. ЛОГАЧЁВА
Ю.В. СТРЕНАЛЮК

«ЯЗЫКИ ПРОГРАММИРОВАНИЯ»
Практикум по курсу «Языки программирования»

Учебное пособие

Королев
2023

УДК
378.016

Исаева Г.Н., Логачёва Н.В., Стреналюк Ю.В. Практикум по курсу «Языки программирования»: учебное пособие. – Королев МО: Технологический Университет имени дважды Героя Советского Союза, лётчика–космонавта А. А. Леонова , 2023. – 108с.

Рецензент: д. т. н., проф. Артюшенко .В.М.

к. т. н, доцент Сидорова Н.П.

Учебное пособие предназначено для закрепления теоретических знаний, полученных в ходе лекций и получения практических навыков решения задач с помощью вычислительных систем (ВС) студентам, обучающимся по программе бакалавриата следующих специальностей 09.03.03 «Прикладная информатика», 09.03.02 «Информационные системы и технологии», 27.03.04 «Управление в технических системах».

Практикум включает восемь работ, позволяющих понять основные подходы и приёмы в написании алгоритмов решения задач на языках программирования высокого уровня (ЯП ВУ). В качестве базовых, выбраны языки программирования C/C++. Пособие составлено таким образом, чтобы студенты сначала познакомились с парадигмой структурного программирования, поэтому программные коды ориентированы на лексику языка программирования С, а затем, после освоения практических работ данного пособия, перешли к кодированию решений задач в рамках вычислительной модели ООП и более детальному изучению ЯП C++.

Пособие составлено в соответствии с требованиями федерального государственного образовательного стандарта высшего образования (ФГОС ВО) для указанных специальностей, а также может быть использовано при подготовке студентов других направлений бакалавриата, таких как: 10.03.01 «Информационная безопасность», 01.03.02 «Прикладная математика и информатика», 09.03.04 «Программная инженерия» при изучении дисциплин, связанных с построением алгоритмов решения задач в современных системах программирования на ЯП ВУ.

© ТУ, 2023
© Исаева Г.Н.,
Логачёва Н.В.,
Стреналюк Ю.В., 2023

СОДЕРЖАНИЕ

СПИСОК СОКРАЩЕНИЙ

БД – база данных
ВС – вычислительная система
ИТ – информационные технологии
НОД – наименьший общий делитель
НОУ – национальный образовательный университет
ОЗУ – оперативное запоминающее устройство
ООП – объектно-ориентированное программирование
ОС – операционная система
ПК – персональный компьютер
ПО – программное обеспечение
ЭВМ – электронно-вычислительная машина
ЭОР – электронный образовательный ресурс
ЯП ВУ – язык программирования высокого уровня
ASCII - American standard code for information interchange
IBM - International Business Machines
IDE- Integrated Development Environment
ISO - International Organization for Standardization –Международная организация по стандартизации

ВВЕДЕНИЕ

В настоящее время сфера деятельности человечества постепенно переходит в цифровое пространство, поэтому умение понимать язык вычислительной системы (ВС) является очень важным аспектом в обучении современной молодёжи; кроме того, темп жизни задаёт вектор *эффективного решения* задач объективного мира, что без ЭВМ практически неосуществимо.

Главным инструментом при решении прикладных задач на ЭВМ являются современные языки программирования, количество которых на сегодняшний день превышает цифру восемь с половиной тысяч [8].

Согласно основным компетенциям, определённым при подготовке бакалавров по направлениям ИТ-специальностей - в рамках курса «Языки программирования», обучаемые должны освоить базовые элементы языков программирования высокого уровня(ЯП ВУ), познакомиться с современными системами программирования, уметь выбирать ЯП ВУ и среды программирования для решения поставленных задач в прикладных областях.

Цель данного пособия – помочь обучаемым получить элементарные навыки решения прикладных задач в среде программирования и постепенно перейти от методов и подходов структурной парадигмы (основанной на алгоритмических структурах и принципе декомпозиции кода), к более сложным задачам объектно-ориентированной методологии решения задач с помощью ЭВМ.

В качестве базовых, выбраны языки программирования С(Си) и С++, наиболее значимые и проверенные временем: Си–подобный синтаксис присутствует сейчас во всех видах и уровнях современных программных продуктов. Средой программирования может быть выбрана любая, которая поддерживает указанные языки программирования, но на взгляд авторов, предпочтение нужно отдать интегрированным средам разработки(IDE- Integrated Development Environment). Одной из таких является Visual Studio от Microsoft, работающая под управлением ОС линейки Windows. Также, выбранные языки программирования позволяют решать прикладные задачи и в другой линейке операционных систем – ОС

Linux, как в интерактивном «on-line», так и в пакетном режимах, используя редакторы для Си-подобного кода.

Решение задач на ЭВМ включает в себя несколько этапов:

1) *Постановка задачи*. На данном этапе происходит сбор информации о задаче, формулируются условия, при которых должна функционировать задача, определяются конечные цели решения задачи и описывается форма выдачи результата. Также даётся описание типам и диапазонам данных, используемых при решении.

2) *Анализ и исследование задачи, модели*. Здесь главным является изучение и исследование найденных аналогичных решений подобных задач, анализ технических и программных средств, позволяющих выполнить задачу с помощью ПК или другой вычислительной системы. Проводится разработка математической модели решения задачи, которая является самым простым объектом, реализуемым на ЭВМ, подбираются структуры данных.

3) *Разработка алгоритма*. Этот этап включает: выбор метода проектирования и формы записи алгоритма; выбор способа записи алгоритма в виде блок-схемы, псевдокода и др., если это необходимо. Формируются тесты и проектируется алгоритм;

4) *Программирование*. Как раз на этом этапе разработчик проявляет своё умение и понимание подходов и программных методов решения прикладной задачи – он должен правильно подобрать ЯП ВУ и провести кодирование алгоритма на выбранном языке программирования; здесь же уточняются формы представления данных, если их затруднительно реализовать на выбранном языке программирования.

5) *Тестирование и отладка*. Необходимо различать отладку и тестирование. Отладка – это поиск синтаксических ошибок, не обязательно всех. Тестирование – проверка ветвей алгоритма на правильность входа-выхода и семантику. Таким образом, на данном этапе происходит синтаксическая отладка, отладка семантики и логической структуры программы, выполняются тестовые расчеты и осуществляется анализ результатов тестирования; программа совершенствуется и приобретает простоту и «элегантность».

6) *Анализ результатов решения задачи* и уточнение, в случае необходимости, математической модели с повторным выполнением этапов 2)-5).

7) *Сопровождение программы*. Необходимо, как правило, доработать программу под конкретную задачу, так как одно из главных свойств любого алгоритма – массовость; составить документацию и пояснения к выполненной задаче, описать математическую модель и алгоритм, пояснить действие набора тестов.

Перечисленные этапы позволяют понять, как много должен знать разработчик программного кода в сфере программных технологий, чтобы создавать читаемые, простые, безопасные и, самое главное, расширяемые программы. Язык программирования в этом процессе занимает важное место.

Практикум включает необходимые краткие начальные сведения по основам программирования для выбранных ЯП ВУ и состоит из восьми практических работ, выполнение которых поможет в приобретении практических навыков тестирования и отладки программного кода на С, усвоению лекционного теоретического материала по курсу, понимания концепции решения задач с помощью ЭВМ. Для выполнения практических работ по данному курсу необходимы минимальные знания обучаемых по основам алгоритмизации и информатики, полученные на первом курсе и в школьных программах.

Пособие предназначено для студентов, обучающихся по направлению «Прикладная информатика», «Информационные системы и технологии», «Управление в технических системах» и может быть полезно бакалаврам других направлений подготовки по ИТ-специальности при изучении дисциплин, связанных с приобретением навыков программирования и алгоритмизации на языках программирования высокого уровня.

Практическая работа 1. ОСНОВНЫЕ СВЕДЕНИЯ О ЯП ВУ И СИСТЕМЕ ПРОГРАММИРОВАНИЯ

ЦЕЛЬ РАБОТЫ: Научиться элементарным навыкам написания кода программы на C/C++ в среде разработки, освоить интерфейс системы программирования для выполнения практических работ по курсу «Языки программирования».

1. Основные понятия

Для того, чтобы решить задачу с помощью ВС, как было указано во Введении, на четвёртом этапе данного процесса, необходимо выбрать язык программирования и иметь среду разработки.

Применительно к написанию кода, среду разработки принято называть *системой программирования*. Система программирования состоит из комплекса программных модулей, которые позволяют записать алгоритм решения задачи на выбранном ЯП ВУ и получить решение задачи в требуемом виде с выводом результата в окно (консоли, приложения) или во внешний источник хранения данных (файл или БД).

Система программирования включает следующие программные модули:

— редактор текста;

— транслятор (который может быть представлен в виде компилятора, интерпретатора или конвертора);

— компоновщик (редактор связей);

— отладчик.

Более подробные теоретические сведения о языках программирования, средах программирования и современным методологиям программирования учащиеся должны изучить во время лекций на портале Университета (https://ies.unitech-mo.ru/) или воспользоваться учебной литературой, электронными образовательными ресурсами - ЭОР, (например, НОУ «ИНТУИТ», рекомендованными министерством образования РФ)[9].

Язык программирования – это формальная знаковая система, предназначенная для записи программного кода.

Язык программирования однозначно определяет, как будут представлены данные объективного мира и какие именно операции будут выполнены над ними при использовании аппаратных и программных методов для реализации модели решения задачи в вычислительной среде. Как и у естественного языка, у языка программирования имеется набор лексических, синтаксических и семантических правил.

По степени детализации предписаний(конструкций языка) языки программирования делятся на:

— языки *низкого уровня* – степень детализации элементарных конструкций очень высока. Это машинные ЯП и машинно-ориентированные ЯП (ассемблеры).

— языки *высокого уровня* – степень детализации конструкций очень низкая и приближена к естественному и математическому языку. Принято делить ЯП ВУ на процедурные(императивные), логические, функциональные, объектно-ориентированные ЯП.

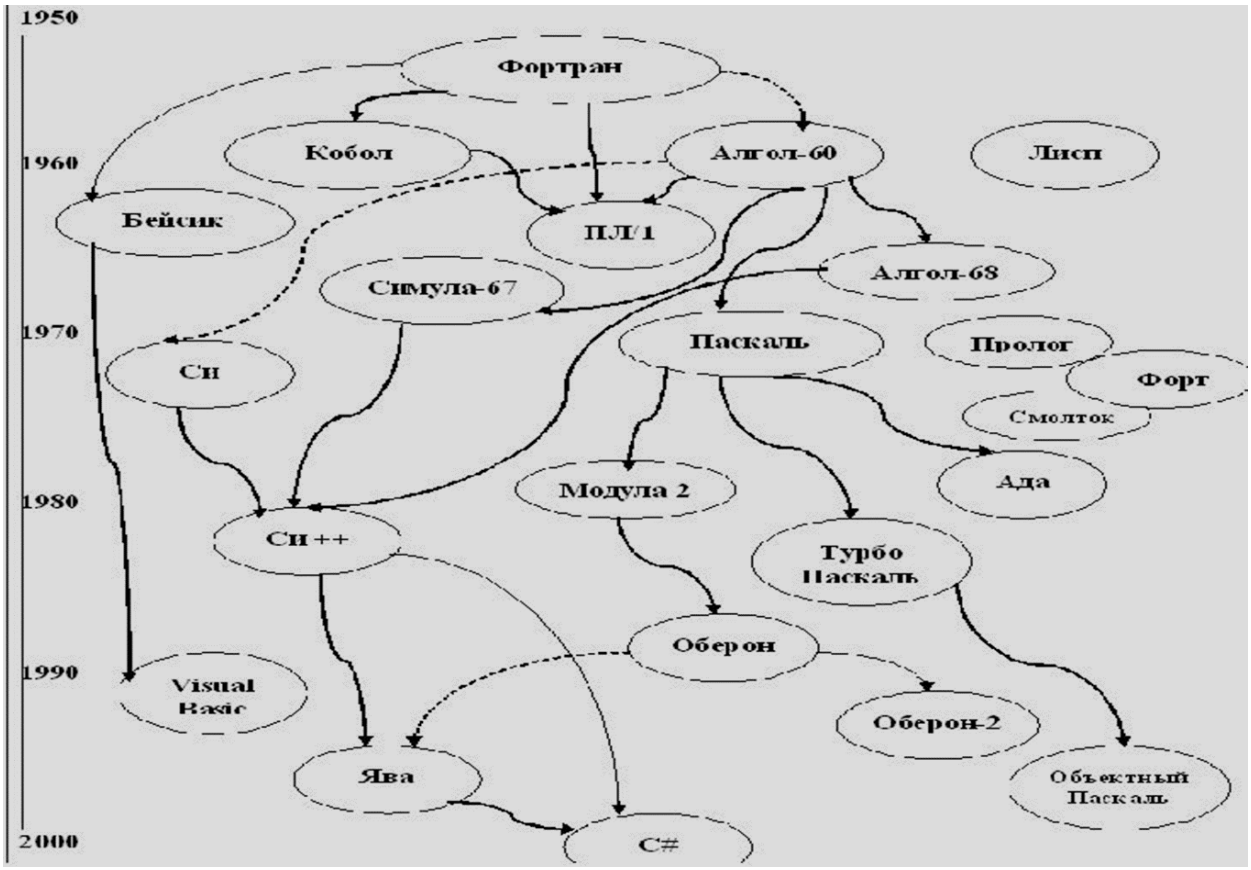

Рисунок 1 – Связи языков программирования на временной шкале

Первым языком программирования высокого уровня является Фортран (Fortran - сокращение от FORmula TRANslator), который в 1954-57гг. прошлого столетия был разработан в корпорации IBM (International Business Machines). Начиная с этого времени, семейство ЯП ВУ расширяется и развивается, согласно требованиям решаемых человечеством задач. На рисунке 1 представлен фрагмент иерархического дерева ЯП ВУ, в привязке ко времени появления Фортрана и до 2000 года. Родственные связи между языками нанесены основной линией, если выше стоящий по уровню язык программирования оказал на ниже стоящий на временной шкале ЯП ВУ непосредственное влияние, и пунктирной линией, если языки находятся в косвенном «родстве». Несмотря на то, что современные ЯП ВУ тяготеют к мультипарадигмальности, исторически, семейство языков программирования делят на четыре основных ветви [6].

Императивное программирование возникло во второй половине прошлого века – позже в нём зародились основы структурного и модульного программирования; языки программирования, относящиеся к данной парадигме: Фортран, Алгол, Паскаль, Бейсик, Си. Кроме того, языки императивного программирования являются и алгоритмическими: они ориентированы на запись алгоритмов — детерминированных последовательностей действий над структурами данных [1].

К языкам *объектно-ориентированной парадигмы* относятся Java, C++, Visual Basic, Eiffel, C#, Kotlin. Объектно-ориентированное программирование (ООП) — это технология, ставшая ответом на кризис программирования конца прошлого века: задачи усложнились, появилось множество слабо структурированных экономических и социальных данных, структурное программирование в промышленных масштабах стало затратным и дорогостоящим. Данная технология основывается на понятии объекта и класса; программный код записывается в виде взаимодействия совокупности объектов, каждый из которых является экземпляром определенного типа (класса), а классы образуют иерархию с наследованием свойств[5].

К непроцедурному программированию относятся: *функциональное* (языки программирования: Лисп, Haskell, Elm и др.) и *логическое* программирования (языки программирования: Пролог,

Mercury,Oz и др.). В функциональном программировании программа рассматривается как суперпозиция функций, применяемых к исходным данным; основным приёмом для построения программного кода является – рекурсивный(когда функция вызывает самоё себя, но с другими аргументами), программный код при таком программировании напоминает математическую запись решения задачи. Логическое программирование основано на формальной логике: формируются база данных и база знаний, а также подсистема логического вывода; решение задачи состоит в получении целевого утверждения, согласно исходным данным и реализованной логической модели.

Конечно, современное программирование в промышленных масштабах отличается от программирования учебного, пользователь может не видеть инкапсулированный код, который находится в библиотеке классов или модулей, ему предоставляется лишь листинг с написанными интерфейсными строками: вызов методов классов или функций. Но к достижению основного принципа ООП (современного программирования XXI века): «Наследуй и изменяй» лежит долгий путь в познании базовых алгоритмических структур, декомпозиции кода, различных типов данных и их правильного использования.

2. Язык программирования C++

Сведения о ЯП ВУ C++ впервые появились в 1983 году в компании Bell Laboratories, его разработчиком был Бьерн Страуструп, но работа над языком продолжалась до конца 80х. В своей работе Страуструп опирался на опыт создателей языков Симула, Модула 2 (см. Рис.1), на язык С, который является непосредственным предшественником C++ и с которым он полностью совместим (библиотеки С работают и в C++).

В 1997 году был принят международный стандарт C++, который обеспечил единообразие всех реализаций языка C++, стандарт утвержден Международной организацией по стандартизации ISO(International Organization for Standardization). Его номер ISO/IEC 14882, стандарт по заявке, за плату можно получить на узле американского национального комитета по стандартам в информационных технологиях: www.ncits.org.

C++ компилируется непосредственно в машинный код, что позволяет ему быть одним из самых быстрых в мире языков, является строго типизированным языком (поддерживает статические и динамические типы данных).

Надо отметить наличие препроцессора – мощного инструмента языка C++, при помощи которого происходит обработка кода перед компиляцией, и затем уже с помощью промежуточного файла, полученного препроцессором, ведётся дальнейшая сборка решения. Директивы препроцессора (неисполняемые операторы в записи кода) начинаются с символа #, наиболее востребованными являются *#include* и *#define*. Директивой *#include* присоединяются файлы(библиотек, заголовочных модулей, и т.п.), присущие ЯП, реализованному в данной версии системы программирования. Директива #define используется для определения макросов и замены одних конструкций на другие, например, макрос __DATE__ заменится на текущую дату, __TIME__ - на текущее время, __LINE__ - на текущую строку. По определению, макрос представляет собой логически завершённый код, подобный действию программной функции. Макросы часто используются в Си, в C++ их использование стало реже.

Язык C++ является объектно-ориентированным, мультипарадигмальным, универсальным языком, востребованным на сегодняшний день у разработчиков ПО, позволяющим решить практически любую задачу программирования. C++, как преемник языка C, широко используется в системном (низкоуровневом) программировании. Но самое главное его преимущество: на C/C++ разработаны все библиотеки современных ОС, систем программирования, пользовательских приложений различного назначения, в том числе мобильных и веб-приложений. Это позволяет на протяжении многих лет занимать первые строчки в рейтингах данным языкам программирования [2-3]. Кроме того, из рисунка 1 видно, что данный язык стал родителем для таких двух мощных современных языков программирования высокого уровня, как Java и C#!

3. Методические указания

Для выполнения практических работ в вычислительной системе, работающей под управлением ОС Windows, необходимо

наличие приложения Visual Studio от Microsoft, не ниже VS 2015. Различные версии Visual Studio (2015, 2017, 2019, 2022) можно установить параллельно с помощью базовых средств компилятора версии 142 для Visual Studio 2019 [11]. Для создания отчётов по выполненным работам – текстовый процессор MS Word, для получения «скриншотов» с результатами работы программных кодов - приложение Paint(Paint 3D) или «Ножницы».

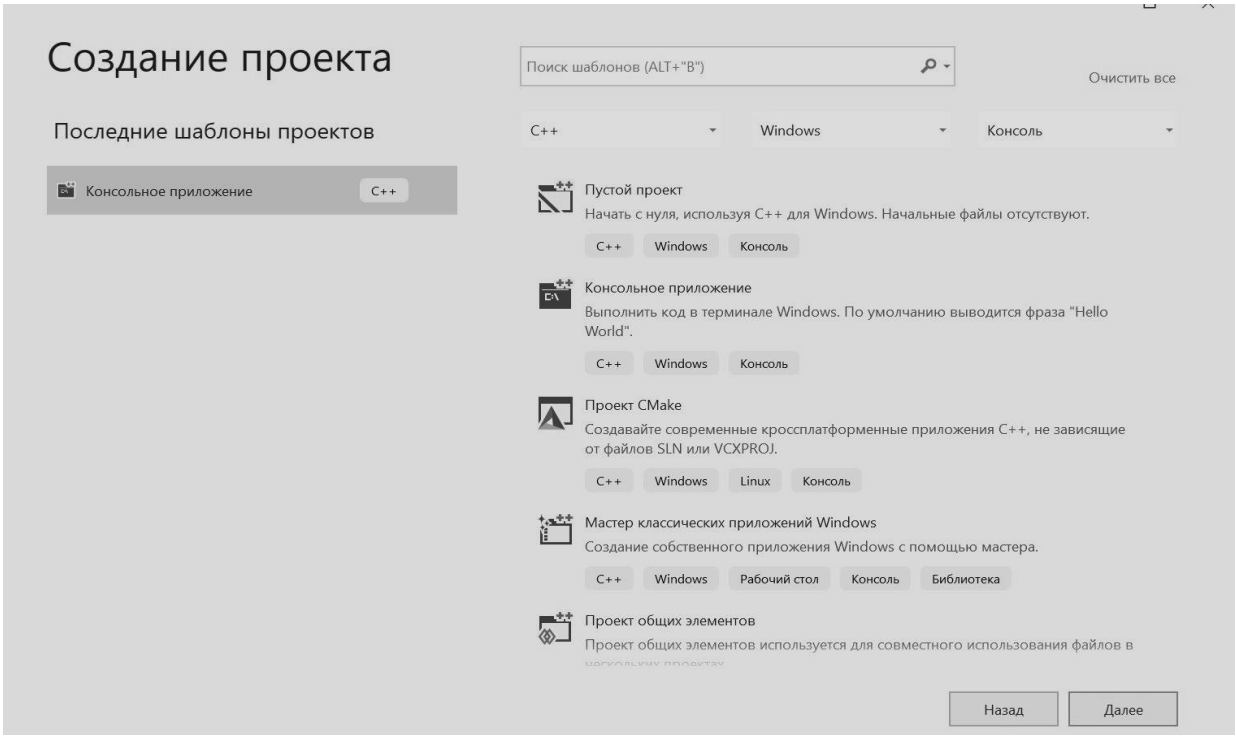

Рисунок 2. - Создание нового проекта

Чтобы начать работу с приложением Visual Studio и попасть в окно редактора, после активации приложения, создайте новый проект(рисунок 2). Далее выберите требуемую вкладку из левого столбца окна «Создание проекта», выберите <Далее> и сформируйте окончательный файл проекта (рисунок 3), набрав его имя в соответствующей строке приглашения.

После этого система откроет окно редактора и создаст пустой проект со всеми настройками и комментариями. Проверьте работоспособность предоставленного макета Вашего будущего проекта решения задачи, нажав зелёный треугольник запуска исходного модуля на компиляцию и выполнение с помощью отладчика Windows. Если решение правильно, то Вы должны получить результат выполнения программного кода в консольном

окне, а в окне вывода получить «нулевой» код завершения программы (рисунок 4).

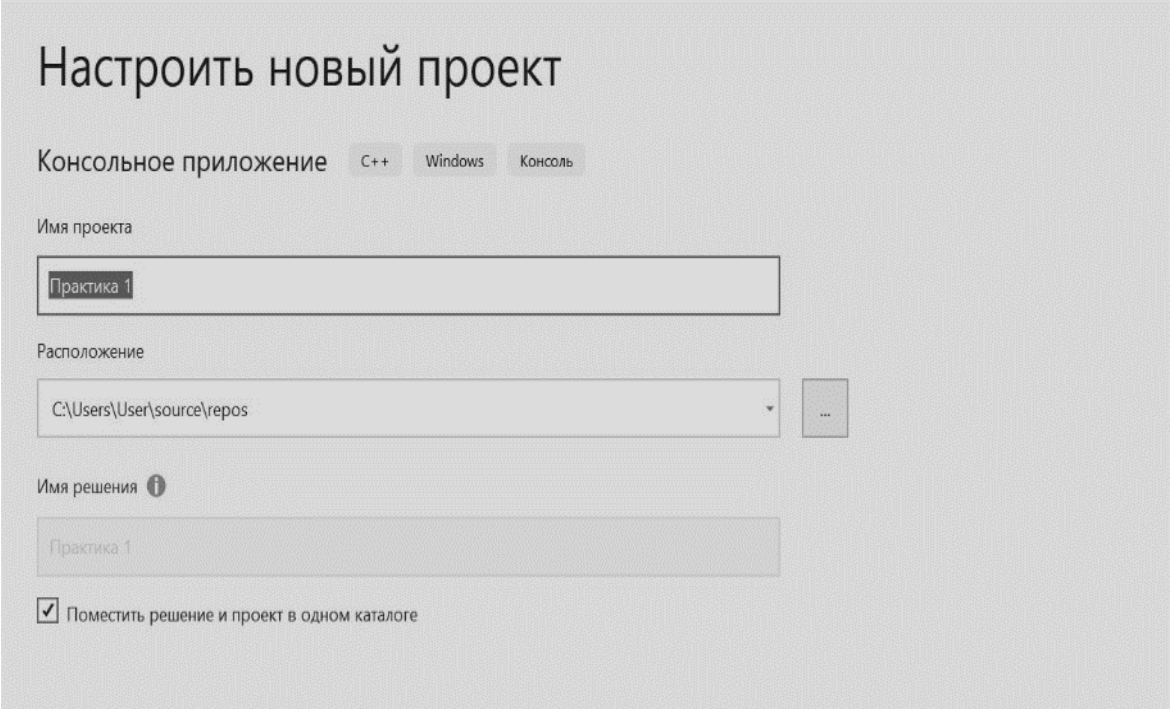

Рисунок 3. – Выбор имени и расположения проекта

До того как приступить к выполнению заданий в среде программирования – познакомьтесь с интерфейсом программного продукта Visual Studio:

— изучите вкладки главного меню: Файл, Правка, Вид, Проект, Отладка;

— изучите окна, которые открываются при запуске приложения: Окно редактора, Обозреватель решения, Вывод и, используя вкладку меню «Вид», попробуйте добавить новые окна на экран или убрать окна с экрана монитора;

— обратите внимание на панель инструментов под вкладками меню и изучите её содержимое: как быстро закомментировать код в окне редактора, как раскомментировать, как активировать локальный отладчик windows, как перейти в начальное окно создания проекта (рисунок 3);

— если возникают вопросы по освоению интерфейса системы программирования – можно использовать помощника по системе, вызвав его функциональной клавишей <F1> с клавиатуры

или обратиться к справочной системе на сайте разработчика программного продукта [10].

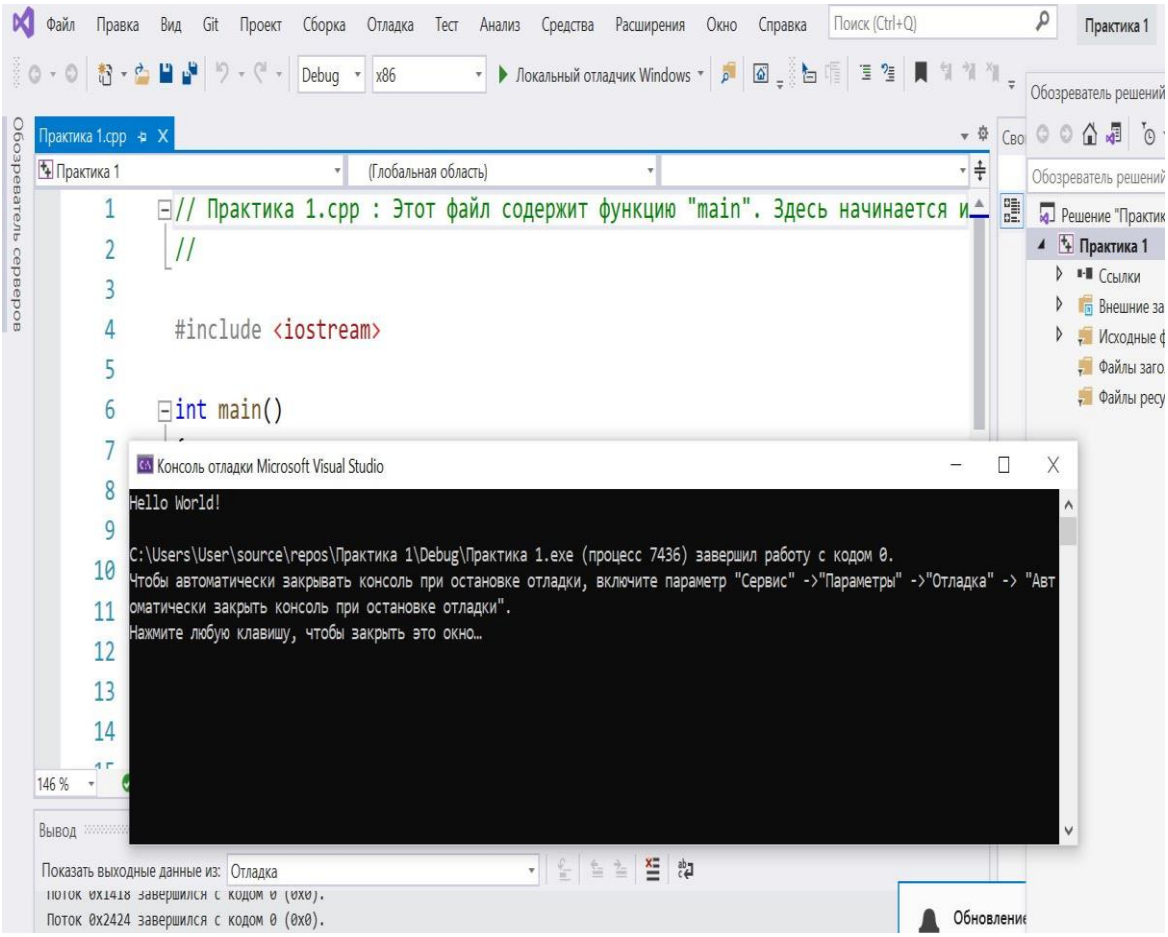

Рисунок 4. – Окна Visual Studio 2019 после выполнения программы

Visual Studio – интегрированная среда разработки, она позволяет создавать консольные приложения, современные кроссплатформенные приложения, оконные приложение; все файлы, относящиеся к Вашему решению находятся в папке, которая имеет одинаковое имя с именем Вашего проекта. Изучите её содержимое: чтобы быстро перейти в данную папку с файлами, можно на выпадающем меню, при нажатии правой кнопкой мыши на имени исходного файла(находящегося над окном редактора), выбрать вкладку «Открыть содержащую папку».

В любом алгоритмическом языке можно выделить четыре составляющих:

— алфавит языка (допустимые символы);
— лексемы (элементарные конструкции);

— выражения(задают определённое правило вычисления некоторого значения);

— операторы(задают законченное описание некоторого действия) [5].

Лексемы образуются из символов, выражения — из лексем и символов, а операторы — из символов, выражений и лексем. Используя материалы лекций и дополнительной литературы, познакомьтесь с синтаксисом и семантикой ЯП ВУ С/С++.

Структура простейшей программы на ЯП ВУ С++/С имеет вид:

```
void main()
{
}
```

Или, если основной модуль представлен в виде функции:

```
int main()
{
return 0;
}
```

В общем виде:

```
<тип возвращаемого значения> <имя> ([ параметры ])
{
тело функции, включающее операторы – является блоком
}
```

Программный код на языке С++/С состоит из директив препроцессора, функций, операторов; одна из функций должна иметь имя *main*; если функция не должна возвращать значение, указывается тип *void*(пусто).

К неисполняемой части программы, кроме директив препроцессора, описаний заголовков функций, относятся операторы, задающие тип переменных. Переменная — это именованная область памяти, в которой хранятся данные определенного типа. У переменной есть имя и значение. Имя служит для обращения к области памяти, в которой хранится значение. Во время выполнения программы значение переменной можно изменять. Перед использованием, любая переменная должна быть описана [8].

Переменные задаются в общем виде в форме:

```
<тип переменной> <имя переменной>;
```

Все типы переменных и константных значений языка можно разделить на основные(базовые, простые) и составные. Основные: целые, вещественные, символьные, логические. Составные типы строятся из основных. К составным относятся массивы, перечисления, записи(структуры), функции, объединения, классы(для C++). Переменные в С могут быть разных типов: простые, структурированные, динамические структуры (образуемые в динамической памяти с помощью указателей). Например, int a; bool b; float **mas.

Более подробно классификация типов изложена в лекциях по курсу (ios.unitech@ut-mo.ru) и в последующих практиках данного пособия.

4. Последовательность выполнения практической работы

1. Активируйте интегрированную среду программирования IDE Visual Studio, войдите в редактор и выполните программный код, который предоставила система.
2. Убедившись, что заданный системой программный код верно работает, и система «правильно видит» все используемые файлы – сохраните проект, вызовом контекстного меню на вкладке меню Файл: «Сохранить всё».
3. Обратите внимание на директиву препроцессора #include <iostream>, которая появилась у Вас в окне редактора после первых двух строк комментария к программному коду. Каждая директива препроцессора несёт свою исполнительную нагрузку и подсоединяет те библиотеки и файлы, в которых находятся нужные функции для правильного исполнения программного кода. В данном случае, для вывода на экран текстового сообщения «Hello, world!».
4. Все операции по вводу-выводу (форматированному и неформатированному) на консоль и работе с консолью содержатся, соответственно, в файлах iostream, stdlib.h, математические формулы находятся в библиотеке math.h, по работе с внешними файлами - fstream, stdlib.h. Но от версии к версии, состав библиотек и файлов, передаваемых на обработку препроцессору меняется, поэтому всегда необходимо обращаться к справочной литературе от фирмы-разработчика IDE. Познакомьтесь по материалам лекций и справочной литературы с операторами #include, #define, using

namespace std, с составом библиотек и заголовочных файлов, необходимых для начальной работы в среде.

5. Перейдите к набору алгоритма решения задачи в кодах языка программирования C/C++. В качестве закрепления навыков по начальному использованию языка C++, выполните Примеры 1.1-1.3 простейших алгоритмов. Отладьте коды и протестируйте решение для различных значений переменных.
6. Ответьте на контрольные вопросы.
7. Составьте отчёт по практической работе.

Пример 1.1 Простейшие математические функции

```
#include <iostream>
using namespace std;
int main()
{	float x;
	setlocale(LC_ALL, "RUS");
	cout << "Привет, студент" << "\nHello, student!!! ";
	cout << "\n Введи аргумент!!!";
	cin >> x;
	cout << "\n Значение функции x^2: \t" << pow(x, 2);
	cout << "\n Значение функции sin(x): \t" << sin(x)
	return 0;
}
```

Пример 1.2 Операции с целыми числами

```
#include <iostream>
using namespace std;
int main()
{
	setlocale(LC_ALL, "RUS");
	int x, y;
	system("cls");
cout << "Введи с клавиатуры x = ";
	cin >> x;
cout << "Введи с клавиатуры y = ";
	cin >> y;
cout << "\nЦелая часть функции x/y = " << x / y;
cout << "\nОстаток от целочисленного деления x/y = " << x % y;
```

```
cout << "\nАбсолютное значение целого переменного"<< x <<"
= " << abs(x);
        return 0;
}
```

Пример 1.3 Уравнение прямой

```
#include <iostream>
int main()
{
setlocale(LC_ALL, "RUS");
cout << "Hello student!\n";
float x1, y1, x2, y2;
float k, b;
cout << "Введите координаты точки A(x1; y1): \n";
cin >> x1 >> y1;
cout << " Введите координаты точки B(x2; y2): \n";
scanf_s("%f%f", &x2, &y2); // показана альтернативная
//организация форматированного ввода в стиле ЯП Си
//  cin >> x2 >> y2;
k = (y1 - y2) / (x1 - x2);
b = y2 - k * x2;
printf("y = %.2fx + %.2f\n", k, b); // альтернативный
//форматированный вывод в стиле ЯП Си
// Замечание: не стоит смешивать форматированный и
//бесформатные ввод-вывод, несмотря на то, что C++ полностью
//совместим с Си. Изучите функции форматированного вывода в
C++.
return 0;
}
```

5. Требования к оформлению отчёта

Отчёт по практической работе должен включать следующие разделы.

1. Титульный лист (Приложение А).
2. Цель работы и постановку каждого выполняемого задания.
3. Описание входных, выходных данных, модели решения задачи.

4. Скриншоты окон в среде программирования, подтверждающие работоспособность задач и примеров, выполненных в кодах изучаемых ЯП ВУ.

5. Выводы по работе.

6. Ответы на контрольные вопросы.

7. Список использованной литературы и Интернет-источников.

Контрольные вопросы

1. Какие этапы содержит процесс решения задачи на ЭВМ. Кратко охарактеризуйте их.

2. На каком этапе решения задач в ВС происходит выбор ЯП и системы программирования.

3. В чём отличие ЯП ВУ C++ и Си?

4. Какие сильные стороны унаследовал от Си современный ЯП ВУ C++?

5. Какие недостатки и по сей день преследуют C/C++?

6. Какие критерии оценки ЯП ВУ существенны на сегодняшний день?

7. Какие программные модули включает система программирования?

8. Отличительные черты интегрированных сред программирования (IDE).

9. Создание каких проектов и на каких ЯП поддерживает IDE MS Visual Studio 2019?

10. На какой вкладке меню MS Visual Studio находятся функции: создать новый проект, открыть существующий проект, быстро пересобрать решение?

11. Для чего предназначено окно «Обозреватель решений?

12. В каком окне системы программирования MS Visual Studio можно получить результат решения задачи, записанной в кодах выбранного ЯП?

13. Перечислите основные компоненты ЯП C++. Какие символы можно использовать в C/C++ для построения лексем, выражений, операторов?

14. Какие конструкции в ЯП ВУ относятся к неисполняемым операторам?

15. Составьте табличку простых типов данных для C++, в которой в виде полей укажите: название типа данных; вид информации(данных), для которых применим тип; максимальный и минимальный размер представляемых данных в байтах(диапазон) в ВС.

16. Какие структурированные типы поддерживаются в C++.

17. Выполните собственную задачу решения арифметического выражения в среде VS на ЯП ВУ. В арифметическом выражении предусмотреть различные математические функции и операции, использовать форматированный и бесформатный ввод-вывод.

Практическая работа 2. АЛГОРИТМИЧЕСКИЕ СТРУКТУРЫ: ЛИНЕЙНАЯ, ВЕТВЛЕНИЕ, ЦИКЛИЧЕСКАЯ

ЦЕЛЬ РАБОТЫ: Исследование основных алгоритмических структур, приобретение навыков по отладке и тестированию программ, содержащих данные структуры на ЯП С/С++.

1. Основные понятия

Алгоритм решения любой структурированной программы должен опираться на базовые алгоритмические структуры: линейную, ветвление, цикл (Рисунок 5).

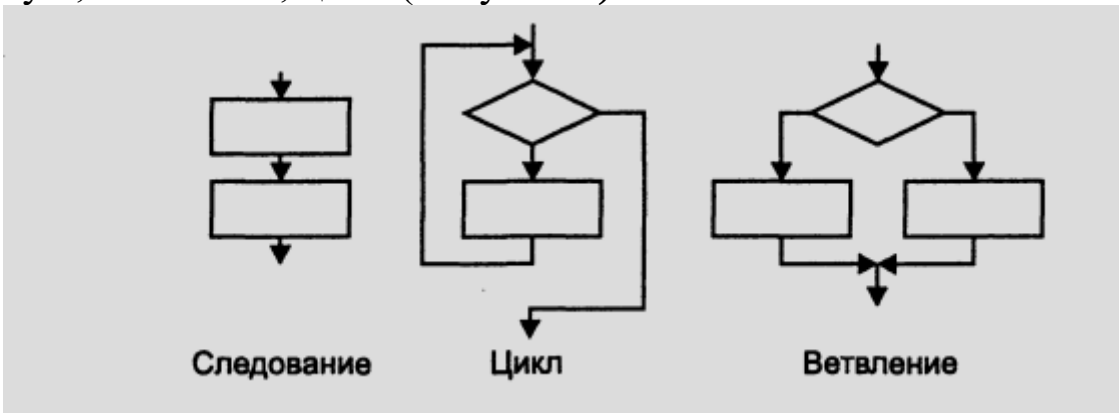

Рисунок 5 -Базовые алгоритмические структуры

Тело любой программы (см. практическая работа 1) относится к исполняемым операторам, располагается в сегменте кода в ОЗУ и идентифицируется скобками { – начало программы, } – конец программы. Иными словами, данный блок {} – является *составным оператором*, может использоваться не только для записи начала и конца программного модуля, но и для выделения нескольких команд в одно целое и определения выполнения данных команд в порядке очереди. Составной оператор может находиться в любом месте программы, где разрешен простой оператор.

Организация программ разветвляющейся структуры.

Оператор ветвления относится к составным, он задает выполнение одного из альтернативных блоков(составных операторов) в зависимости от результата выражения , являющимся *выбором* (или вычисляемым *условием*). Для выражений блока «Условие» в языке С/С++ могут быть использованы следующие операции сравнения(операции отношения):

> Больше
>= Больше или равно
< Меньше
<= Меньше или равно
!= Не равно
== Равно

Результатом данных операций является значение *true (истина)* или *false(ложь),* кроме того, любое значение, не равное нулю, интерпретируется как *true*. Например, выражение *if(n)* будет истинным всякий раз, когда переменная *n* отлична от 0, т.е. выражения условия будет эквивалентно записи *if(n!=0).*

А также логические операции:
операция && – логическое И;
операция || – логическое ИЛИ;
операция ! – логическое НЕ.

Существуют структуры, задающие ветвление (Рисунок 6). Их можно описать в кодах C/C++ следующим шаблоном:

if (…) {…} [*else {…}*], где [] – обозначают возможное отсутствие данной части составного оператора ветвления.

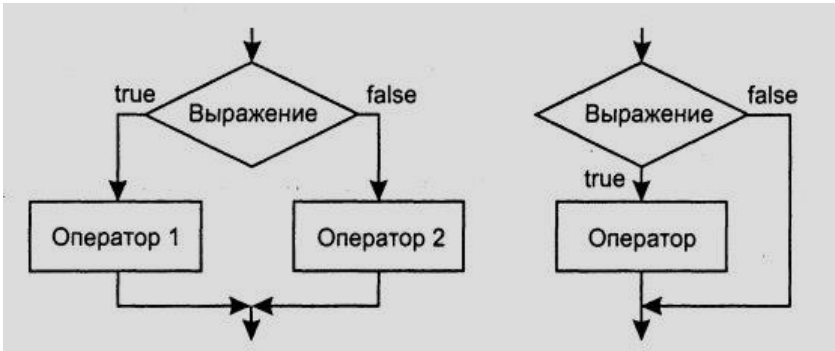

Рисунок 6 – Структуры ветвления

Составной оператор ветвления работает следующим образом: вычисляется выражение (условие) и, в зависимости от результата, выбирается на выполнение один из альтернативных блоков операторов, далее программа идёт своим чередом.

Оператор выбора *switch* (или множественное *if*) также является составным и предназначен для выбора одного из блоков операторов при выполнении условия. Как правило, каждый блок нумеруется каким-либо константным выражением(меткой), по которому и будет происходить идентификация блока операторов, который должен

быть выполнен по условию в переключателе *switch* (Рисунок 7). В общем виде, оператор можно представить как:

switch (<выражение>)
{
case <константное выражение 1>: [блок операторов 1] *break*;
case <константное выражение 2>: [блок операторов 2] *break*;
…
case <константное выражение n>: [блок операторов n] *break*;
[*default*: блок операторов *break*;]
}

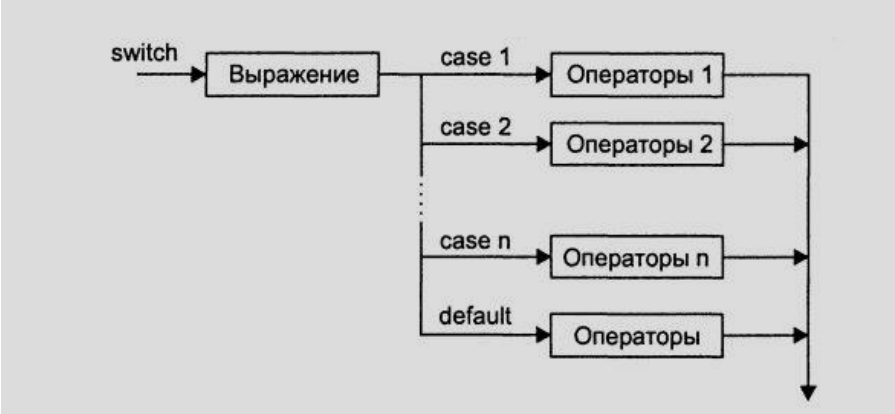

Рисунок 7 – Схема оператора switch

Switch, case, break, default – ключевые слова языка C/C++; <выражение> – программный код, в котором используются переменные целого или символьного типа;
<константное выражение 1>,…,<константное выражение n> – метки, тип которых совпадает с типом <выражение>;

Порядок выполнения оператора выбора: вычисляется <выражение>; если одна из меток совпадает со значением выражения, то программа передает управление на соответствующий блок операторов [блок операторов 1], …, [блок операторов n](между *case* и *break*). Последний блок операторов, следующий за *default*, выполняются в том случае, если значение выражения не совпало ни с одной из меток; инструкция *default* может и отсутствовать.

Следующая важная структура алгоритмизации – «Цикл». В программировании, циклическая структура задаётся блоком операций (тело цикла), которые должны быть выполнены заданное число раз (итераций) или число раз, пока не будет достигнуто некоторое условие. Исходя из этого в С/Си++ предусмотрена

реализация трёх циклических структур: цикл с предусловием, цикл с постусловием, цикл с заданным числом итераций (рисунок 8).

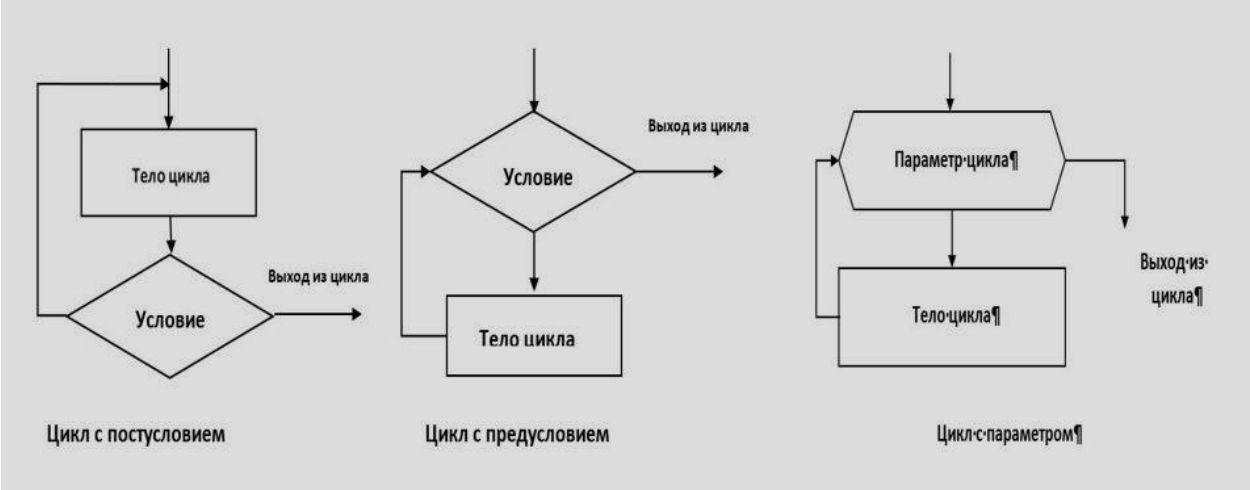

Рисунок 8 – Виды циклических структур

Цикл с постусловием в кодах рассматриваемых ЯП имеет формат:

Do {

< тело цикла: команды кода (инструкции)>

} *while*(<условие выполнения цикла>);

Данный цикл выполняется до тех пор пока условие выполнения цикла – истинно, Но из схемы видно, что если даже условие ложно, один раз цикл всё-таки выполнится, так как тело цикла стоит впереди условия!

Вторая конструкция на рисунке 8 – цикл с предусловием. Его синтаксис в C/C++ следующий:

while (<условие выполнения цикла>){

< тело цикла: команды кода (инструкции)>

}

Данный цикл сначала проверяет условие и если оно выполняется (истинно) – выполняется и тело цикла.

Третий вид цикла – с параметром (или счётчиком итераций цикла) записывается в кодах языка по следующему шаблону:

for (определение параметра(инициализация счётчика); условие для выполнения цикла; изменение параметра(счётчика))

{

< тело цикла >

}

Инициализация счётчика выполняется один раз, при начальном входе в цикл. Проверка условия происходит каждый раз, когда цикл переходит на новую итерацию, также на каждой итерации идёт изменение параметра, как правило, увеличение или уменьшение его.

Не зависимо от вида цикла, чтобы не выполнять лишние итерации, когда уже достигнут требуемый результат, из цикла надо выйти. Для этого есть оператор *break*. Иногда необходимо перейти к следующей итерации цикла, в этом случае используют оператор *continue*.

Цикл *for* обычно используют для обработки структур данных, имеющих фиксированную длину, представленных типом массив или объектами - экземплярами класса вектор(C++)(см. Практику 3-5).

2. Методические указания

1. Прежде, чем приступить к выполнению примеров данной главы – рекомендуется изучить основные операции, допустимые в ЯП C++/C, в соответствии с их приоритетами. Полный список операций можно посмотреть, например, в рекомендуемой литературе по курсу [5],[8].

2. Очень важно понимать синтаксис применяемых конструкций, но современные среды программирования имеют мощные средства синтаксического анализа, поэтому всегда можно воспользоваться подсказкой системы в написании лексем и операторов, нажав на текущей инструкции сочетание клавиш<Ctrl>+<Пробельная клавиша> для IDE Visual Studio. Отладка – это процесс исправления ошибок в написании кода, как правило синтаксических.

3. Процесс тестирования более сложен и требует аналитических способностей у разработчика кода. При тестировании, заранее готовьте тестовый пакет, чтобы проверить как можно больше ветвей алгоритма. Корректируйте тесты по мере нахождения семантических ошибок в программном коде решения задачи. Сначала проверьте правильность работы алгоритма на ввод-вывод, затем приступайте к детальному исследованию алгоритма на правильность работы при разных значениях входных данных.

4. Дополните задачу «Пример 1.2» (из предыдущей практической работы) алгоритмической структурой «ветвление», отладьте и протестируйте полученный код (Пример 2.1) для разных значений логической переменной *a*.

Пример 2.1 Структура ветвление

```
int main()
{
	setlocale(LC_ALL, "RUS");
	int x, y;
	bool a;
	system("cls");
	cout << "Введи с клавиатуры  x = ";
	cin >> x;
	cout << "Введи с клавиатуры  y = ";
	cin >> y;
	cout << "Введи с клавиатуры  a = ";
	cin >> a;
	if (a){
		// Если логическая переменная a задана - то выполняется
		// x / y
	cout << "\nЦелая часть функции x/y = " << x / y;
		}
	else
		{	// Если логическая переменная a не определена - то
		//выполняется x % y
	cout << "\n Остаток от деления x/y = " << x % y;
		}
	return 0;
}
```

5. Выполните Пример 2.2, в стиле ЯП Си. Дополните решение математической моделью, подтверждающей правильность полученного результата.

Пример 2.2 Определение вида треугольника

```
int main() {
   setlocale(LC_ALL,"rus");
   int a, b, c;
   printf("a=");
   scanf_s("%d", &a);
```

```
        printf("b=");
        scanf_s("%d", &b);
        printf("c=");
        scanf_s("%d", &c);
        if (a + b <= c || a + c <= b || b + c <= a)
           printf("Треугольник не существует\n");
        else
           if (a != b && a != c && b != c)
              printf("Разносторонний треугольник\n");
           else if (a = = b && b = = c )
                 printf("Равносторонний треугольник\n");
              else
                 printf("Равнобедренный треугольник\n");
        return 0;
     }
```

6. Исследуйте структуру выбор - множественное *if*; выполните Пример 2.3.

Пример 2.3 Простейший калькулятор

```
int main()
{
     setlocale(LC_ALL,"rus");
     // простейший калькулятор:
     int x, y, res;
     char oper;
     bool priz = true; // переменная, идентифицирующая, что
//операция состоялась
cout << "\n Введите x= ";
     cin >> x;
cout << "\n Введите y= ";
     cin >> y;
cout << "\n Введите знак операции: ";
     cin >> oper;
     switch (oper) {
          case '+': res = a + b; break;
          case '-': res = a - b; break;
          case '*': res = a * b; break;
          case '/': res = a / b; break;
```

```
            default: {cout << "\n" << oper << " - это неизвестная
операция"; priz = false; break;
        }
        if (priz) cout << "\n Результат операции: " << res;
                    return 0;
              }
```

7. Далее выполните примеры на различные виды циклической структуры: примеры 2.4-2.6.

Пример 2.4 Квадраты чисел с использованием цикла с предусловием.

```
int main()
  {
    setlocale(LC_ALL,"rus");
    int i = 1, n = 6;
       while (i < n)
       {
     cout << i << " * " << i << " = " << i * i << endl;
          i++;
       }
       return 0;
    }
```

Пример 2.5 Квадраты чисел с использованием цикла с постусловием.

```
int main()
   {
    setlocale(LC_ALL,"rus");
    int i = 1, n = 8;
      do
     {
     cout << i << " * " << i << " = " << i * i << endl;
          i++;
      } while (i <= n);
        return 0;
    }
```

Пример 2.6 Квадраты чисел с использованием цикла for.

```
int main()
   {
    setlocale(LC_ALL,"rus");
```

```
int i = 1, n = 10;
   for (int i = 1; i < n; i++)
   {
      std::cout << i << " * " << i << " = " << i * i << endl;
   }
      return 0;
   }
```

Если в теле цикла одна инструкция, то блок не используется, то есть {} не ставятся. Проверьте это положение самостоятельно для трёх видов цикла.

8. Ответьте на контрольные вопросы.
9. Составьте отчёт по практической работе.

3. Варианты заданий для самостоятельной работы

Ветвление

Используя изученные алгоритмические структуры, выполните решение следующих задач:

1) Определите, является день недели выходным или рабочим по введённому с клавиатуры номеру дня недели. Выведите его название.

2) Даны длины трёх отрезков, проверьте, будут ли они сторонами треугольника? Вычислите площадь треугольника.

3) Дано двухзначное число, определить, является ли сумма цифр данного числа однозначным или двузначным числом.

4) Дано трёхзначное число. Определить является ли сумма его цифр двузначным числом.

5) Дано двухзначное число, определить, является ли произведение его цифр однозначным или двузначным числом.

6) Проверить, попадает ли заданное с клавиатуры число *a* в заданный с клавиатуры интервал. Предусмотреть два типа вводимых данных – целый и вещественный.

7) Ввести три числа, выдать на консоль эти числа в порядке рейтинга – прямого и обратного. Предусмотреть два типа вводимых данных – целый и вещественный.

8) Треугольник задан координатами вершин, определить, принадлежит ли введённая с клавиатуры точка A(x,y) треугольнику?

9) Треугольник задан координатами вершин, найдите уравнение одной из высот треугольника.

10) В равнобедренной трапеции (заданной координатами вершин) найти точку пересечения боковых сторон.

11) Равнобедренная трапеция задана координатами вершин. Определить, принадлежит ли точка A(x,y) трапеции?

12) Определить, будет ли студент получать стипендию по результатам сессии из четырёх предметов. Получение оценки «тройки» и ниже – лишает студента стипендии.

13) Определить в каком координатном углу в системе координат (x,y) находится введённая с клавиатуры точка A(X,Y)?

14) Дано трехзначное целое число, определить, содержит оно 2 одинаковых цифры? Все цифры одинаковы?

15) Дано трехзначное целое число. Составить программу, которая определяет, является ли сумма цифр числом меньше 15. Число вводить с клавиатуры.

16) Определить для четырёхзначного числа, введённого с клавиатуры, какие цифры этого числа четные, а какие нечетные.

17) Определить, не приведет ли к переполнению сложение двух целых чисел.

18) Определить, не приведет ли к переполнению умножение двух целых чисел.

19) - 25) Решить систему линейных уравнений. Вариант условия системы взять из предмета 1 семестра «Теоретические основы информатики»(«Информатика»).

26) Даны координаты четырех вершин. Определить, является ли данный четырехугольник трапецией.

27) Даны координаты четырех вершин. Определить, является ли данный четырехугольник параллелограммом.

Циклы

Используя изученные циклические структуры, выполните решение следующих задач:

1) Вывести на экран в одну строку целое число, затем его квадрат и куб. Операцию повторить для всех целых чисел от 1 до 20.

2) Вычислите сумму только нечетных чисел из заданного диапазона.

3) Вычислите сумму только четных чисел из заданного диапазона.

4) Выведите на консоль таблицу умножения

5) Используя итерационный *алгоритм нахождения корня n-степени* вычислите $y(x)=\sqrt[n]{x}$, задав точность eps ≤ 0.01.

6) Рассчитайте факториал n! для заданного натурального числа n.

7) Вычислите сумму факториалов 1!+ 2!+ …+ n! до заданного n.

8) Дано трёхзначное целое число, используя операцию деления на 2 данного числа и получаемых при этом чисел, уменьшите исходное число до числа, не превышающего число A. Сколько получилось итераций, чтобы достичь результат.

9) Дано четырёхзначное целое число, используя операцию деления на 3 данного числа и получаемых при этом целых чисел, уменьшите исходное число до не превышающего число A. Сколько понадобилось итераций, чтобы достичь результат.

10) Найдите произведение чисел Фибоначчи, начиная со второго числа, не превышающее наперёд заданное пятизначное число A. Сколько чисел Фибоначчи понадобилось перемножить?

11) Найдите сумму чисел Фибоначчи, не превышающую наперёд заданное четырёхзначное число A. Сколько чисел Фибоначчи понадобилось сложить?

12) Представьте целое число в виде простых множителей.

13) Вычислите функцию $f(x) = 2x - sin(x)$ на отрезке $x \subset [a,b]$ с шагом dx. Результат выдать в табличном виде.

14) Рассчитайте сумму и произведение вводимых с клавиатуры различных вещественных чисел: результат разделите отдельно для отрицательных и положительных чисел. Признаком конца операции выберите ввод числа, равного нулю.

15) 20) Решите методом касательных уравнение функции $f(x)=0$ для заданной точности eps ≤ 0.01. Сколько итераций понадобилось для этого? Данные для функции $f(x)$ приведены в таблице 1. Увеличьте точность результата и получите новые значения функции: изменились ли они?

Таблица 1 – Исходные данные для вариантов 15) – 20)

1)	$F(x) = 5x - x^2 + 14$
2)	$F(x) = x^2 + 2x + 2$
3)	$F(x) = 3x^2 - 4x + 2$
4)	$F(x) = -2x^2 + 3x + 6$
5)	$F(x) = -2(x - 1)^2 + 8$
6)	$F(x) = -x^2 - 4x - 1$

4. Требования к оформлению отчёта

Отчёт по практической работе должен включать следующие разделы.

1. Титульный лист (Приложение А).
2. Цель работы и постановку каждого выполняемого задания.
3. Описание входных, выходных данных, модели решения задачи.
4. Скриншоты окон в среде программирования, подтверждающие работоспособность задач и примеров, выполненных в кодах изучаемых ЯП ВУ.
5. Выводы по работе.
6. Ответы на контрольные вопросы.
7. Список использованной литературы и Интернет-источников.

Контрольные вопросы

1. В каких случаях целесообразно применение структуры «выбор», а в каких – «ветвление»?
2. Какой тип выражения может быть использован в операторе *switch* структуры *«выбор»?*
3. Чем отличается цикл *For …* от других видов циклов в структурном программировании.
4. Могут ли циклы *for (…){…}* быть вложенными?
5. Согласно *Теореме структурирования*, любой программный код в структурном программировании (например, на ЯП ВУ Си) можно написать с помощью двух алгоритмических структур. Каких?

6. Необходимо решить задачу численным методом, где количество итераций вычислений некой функции неизвестно заранее. Известна лишь точность. Какой тип цикла в C++/C целесообразно использовать? Приведите пример реализации.

7. Если в записи цикла *For*( ){} задана переменная – счётчик итераций цикла, можно ли её изменить вне тела цикла? Поясните.

8. Почему нежелательно использование оператора безусловного перехода *goto* в структурном и ООП программировании?

9. Чем отличается действие операторов перехода *break* и *continue*?

10. Поясните действие оператора *return* в C++.

Практическая работа 3. Структурированные типы. ОДНОМЕРНЫЕ И МНОГОМЕРНЫЕ МАССИВЫ

ЦЕЛЬ РАБОТЫ: Продолжить исследование типов данных и методов работы с ними в C/C++, приобрести практические навыки по отладке и тестированию программ, содержащих переменные типа «массив».

1. Основные понятия

В типизированных ЯП ВУ, к которым относятся рассматриваемые языки C/C++, для переменных, прежде чем их использовать должен быть объявлен тип. Типы данных, как правило, делятся на базовые (простые) и структурированные (или определяемые пользователем), а также ссылочный тип, который является основой построения более сложных структур - динамических. В свою очередь, среди структурированных, есть типы для представления однородных данных или неоднородных данных [1],[9].

Для задания однородных данных служит тип массив – это структурированный тип, его элементы одного типа и задаются в определённом порядке следования, их положение однозначно определяется индексом элемента; причём индекс элемента может быть одномерным, двумерным, в общем случае, *к*-мерным. Количество элементов и их тип должно быть определено заранее в массиве, до момента компиляции, если массив статический; и количество элементов определяется в ходе выполнения программного кода, если массив динамический, в этом случае до компиляции объявляется только указатель на тип будущих элементов массива.

Элементы типа «массив» можно представить в виде схемы, как показано на рисунке 9. A[N] - название массива, N -количество элементов массива, Ai – элемент массива, i = 1÷N – индекс массива, кроме того если индекс массива представлен в виде одного числа, то это - одномерный массив и его элементы последовательно нумеруются A_1, A_2, …, Ai , … A_N. (рисунок 9 а)). Что касается двухмерного массива A[N,M], то его элементы представляются двузначным индексом Aij, где массив имеет уже матричную или

табличную структуру, размера NxM, где N- количество строк, M – количество столбцов в матрице (таблице)(рисунок 9,б).

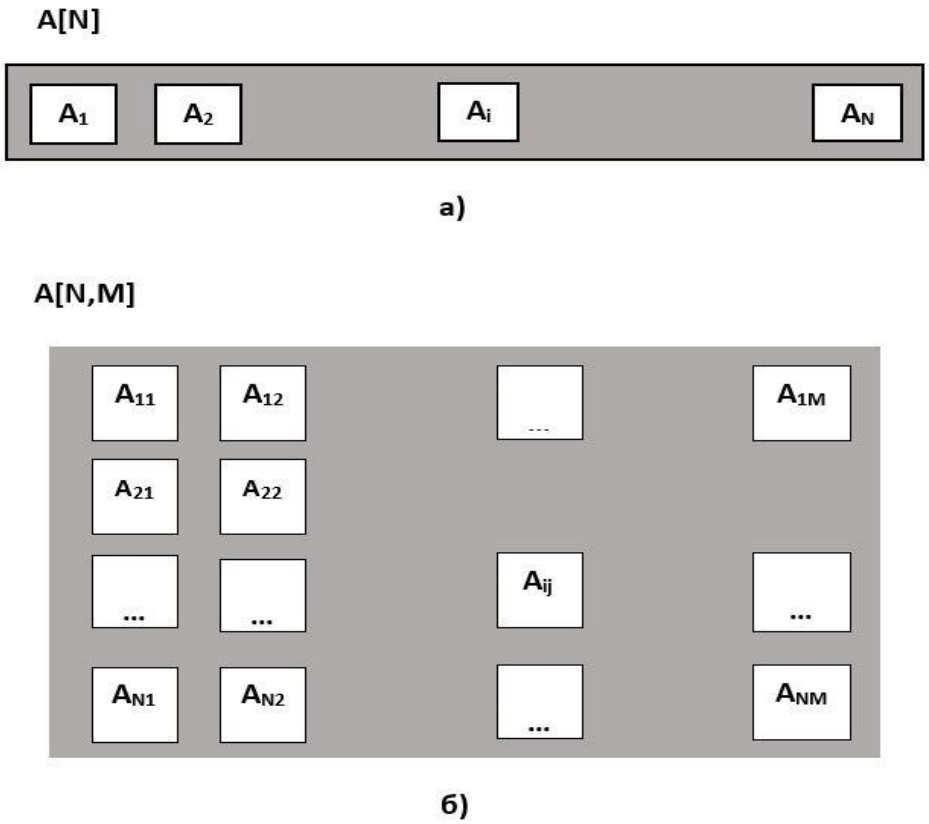

Рисунок 9 – Структура данных типа массив

В общем случае, индекс массива может быть многомерным(например, размерности d), то есть для представления одного элемента массива Aijl…k служит точка в d-мерном пространстве, размер которого равен количеству индексов i,j,l,…, k. Массив будет именоваться, как A[N, M, H, …, S], где i =1÷N, j =1÷M, l=1÷H, … , k = 1÷S. Шаблон структурированного типа «массив» на C/C++ выглядит так:

<тип> <имя массива> [<размер массива>];

Запись массива в кодах C/C++ для разных типов данных:

```
int const n=100;
int const L=10;
int a[n], *b, F[L][L];
float cc[10];
char str[4];
```

Инициализировать массив можно по-разному, это может выглядеть в виде следующих выражений:

int A[4] = {1,2,3,4};
int const N= 3, M = 2;
int numb[N][M] = { {1, 2}, {3, 4}, {5, 6} };
int F[M][N] = { 1, 0, 0, 1, 1, 0,};
*char str[M*N] = {'E','R','R','O','R','5'};*

2. Методические указания

1. При выполнении работ с массивами проход по элементам данных осуществляется с помощью цикла for, если его размер заранее известен. Причём, для двумерных массивов, согласно хранению элементов в памяти, сначала идёт выборка строки 1, затем строки 2 и т.д. Для трёхмерного массива – сначала указывается индекс «плоскости», а затем проход по элементам строк, как в двумерном массиве. Выполните проход и выдачу элементов на консоль, объявленных в пункте 1. Добавьте к ним выдачу на консоль трёхмерного массива.

```
int main() {
        int A[4] = { 1,2,3,4 };
        int const N = 3, M = 2;
        int numb[N][M] = { {1, 2}, {3, 4}, {5, 6} };
        int F[M][N] = { 1, 0, 0, 1, 1, 0, };
        char str[M * N] = { 'E','R','R','O','R','5' };
        int i;
    cout << "\n Исходный одномерный массив :\n";
        for (i = 0; i < 4; i++)
            cout << A[i] << "\t";
    cout << "\n Исходный двумерный массив NxM:\n";
        for (i = 0; i < N; i++) {
            cout << "\n";
                for (int j = 0; j < M; j++)
            cout << numb[i][j] << "\t";
        }
    cout << "\n Исходный двумерный массив MxN:\n";
        for (i = 0; i < M; i++) {
            cout << "\n";
                for (int j = 0; j < N; j++)
```

```
                cout << F[i][j] << "\t";
        }
        cout << "\nИсходный массив символов\n";
        for (i = 0; i < M * N; i++)
            cout << str[i];
        return 0;
    }
```

2. Как правило, для тестирования не нужен массив данных полного размера (или максимального), объявленного при компиляции, поэтому можно воспользоваться для статических массивов следующим приёмом:

```
int const n1 = 100;
int n, a[n1];
…
cout << " Введи размер массива:\n";
    cin >> n;
    cout << "\n Исходный массив:\n";
    for (int i = 0; i < n; i++)
    {
        // произвольное формирование  массива
        a[i] = rand() % 10- rand() % 10;
            cout << a[i] << " ";
    }
```

3. Для осуществления элементарных навыков по работе с массивами, выполните отладку и тестирование примеров (Пример 3.1 -3.4).

Пример 3.1 Расчёт суммы элементов массива

```
    int main()
    {
        Setlocale (LC_ALL, "RUS");
        const int n = 5;
        int i, sum;
        int mas[n] = { 32, 4, -1, -4, 0 };
        for (i = 0, sum = 0; i < n; i++) sum += mas[i];
        cout << "Сумма элементов массива: " <<  sum;
        return 0;
    }
```

Дополните данный пример произвольным формированием массива и выводом массива любой длины, не превышающей заданные размеры.

Пример 3.2 Упорядочивание элементов массива в порядке возрастания, без использования дополнительного массива.

```
int main() {
	const int n2 = 20;
	int b[n2];
	int i, n, a, imin;
	setlocale(LC_ALL, "RUS");
	cout << " Введи размер массива:\n";
	cin >> n;
	cout << "\n Исходный массив:\n";
	for (i = 0; i < n; i++)
	{
		b[i] = rand() % 100- rand() % 100;
		cout << b[i] << "\t";
	}
	for (i = 0; i < n - 1; i++) {
		 imin = i;
		for (int j = 1 + i; j < n; j++)
			if (b[j] < b[imin]) imin = j;
			 a = b[i];
			b[i] = b[imin];
		b[imin] = a;
	}
	cout << "\nВывод упорядоченного массива:\n";
	for (i = 0; i < n; i++) cout << b[i] << "\t";
	return 0;
}
```

Пример 3.3 Заполнение элементов верхнего треугольника квадратной «единичной» матрицы нулями.

```
void main() {
	const int n2 = 20;
	int a[n2][n2];
	int i, n, k;
	cout << "Введи размер матрицы n= ";
		cin >> n;
```

```
cout << "Элементы верхнего треугольника матрицы: \n";
if (n % 2 == 0)
k = n / 2; // число строк для вывода в матрице, если n чётно
else
k = n / 2 + 1; //число строк для вывода в матрице, если n нечётно
        for (i = 0; i <k; i++){
                cout << "\n";
                for (int j = 0; j < n; j++)
                {
if ((i <= j) && (j  <= (n - 1 - i))) a[i][j] = 0; else
                a[i][j] = 1;
                        cout << a[i][j] << "\t";
                }}}
```

Пример 3.4 Заполнение элементов верхнего и нижнего треугольника заданной квадратной матрицы выбранным символом

```
void main() {
        const int n2 = 20;
        char a[n2][n2],c;
        int i, j, n;
        cout << "Введи размер матрицы n=\t";
        cin >> n;
        cout << "\n Заполнение верхнего и нижнего треугольника матрицы выбранным символом: \n";
        c = '*';
        for (i = 0; i < n; i++)
        {
                cout << "\n";
                for (j = 0; j < n; j++)
        {
        if ((i<=j && j<=(n-1-i)) || (i>=j && i>=(n-1-j))) a[i][j]=c; else
                                a[i][j]='-';
                        cout << a[i][j] << "\t";
                }}}
```

4. Ответьте на контрольные вопросы.
5. Составьте отчёт по практической работе.

3. Варианты заданий для самостоятельной работы

Одномерные массивы

1) Дан одномерный массив, состоящий из вещественных элементов, вычислить сумму отрицательных элементов массива.

2) Дан одномерный массив, состоящий из вещественных элементов, вычислить произведение элементов массива, расположенных между максимальным и минимальным элементами.

3) Дан одномерный массив, состоящий из вещественных элементов, отсортируйте элементы массива по возрастанию, используя «пузырьковый» метод.

4) Дан одномерный массив, состоящий из вещественных элементов, вычислить сумму положительных элементов массива;

5) Дан одномерный массив, состоящий из целых элементов, вычислить произведение элементов массива, расположенных между максимальным по модулю и минимальным по модулю элементами.

6) Дан одномерный массив, состоящий из вещественных элементов, используя метод перестановок упорядочить элементы массива по убыванию.

7) Определить минимальный и максимальный элемент одномерного целочисленного массива. Найти количество элементов, находящихся между первым отрицательным элементом массива и минимальным элементом.

8) Дан одномерный массив, состоящий из вещественных элементов, вычислить минимальный и максимальный элемент массива и определить количество элементов между первым нулевым элементом массива и максимальным.

9) Дан одномерный целочисленный массив, найти количество элементов массива, кратных пяти.

10) Дан одномерный целочисленный массив, определить какое количество отрицательных и положительных элементов он содержит; рассчитать среднее арифметическое всех элементов массива.

11) Дан одномерный массив, состоящий из целых элементов, вычислить минимальное произведение двух рядом стоящих чисел.

12) Дан одномерный целочисленный массив, определить произведение элементов массива с четными номерами.

13) Дан одномерный целочисленный массив, определить сумму элементов массива, расположенных между первым и последним нулевыми элементами.

14) Дан одномерный целочисленный массив, упорядочить массив следующим образом: положительные элементы – нулевые – отрицательные.

15) Дан одномерный массив, состоящий из вещественных элементов в количестве 50, найти сумму положительных элементов и произведение отрицательных элементов, попадающих в диапазон [-5;5].

16) Дан одномерный массив, состоящий из 100 вещественных элементов, найти сумму положительных элементов и произведение отрицательных элементов, порядковый номер которых кратен трём.

17) Дан одномерный массив, состоящий из 50 вещественных элементов, найти среди элементов массива числа, не превышающие по модулю число 5 и имеющие чётные индексы; упорядочить найденные элементы по возрастанию.

18) Последовательно вводите с консоли целые числа, до нажатия определённого сочетания клавиш – прекращения ввода. Составьте из введённых чисел массив, у которого сначала располагаются отрицательные элементы, затем нулевые и в конце – положительные. числа. Подсчитайте количество нулевых элементов с нечётными индексами.

19) Даны два целочисленных массива в диапазоне [-50;50], проверьте, есть ли среди элементов массивов уникальные неповторяющиеся ни в одном массиве числа. Составьте из этих чисел третий массив и вычислите сумму его элементов.

20) Дан одномерный целочисленный массив, проверьте, есть ли среди элементов массива арифметическая прогрессия, с коэффициентом 3? Если результат положительный, сформируйте из найденной прогрессии новый упорядоченный массив.

21) Дан одномерный целочисленный массив, проверьте, есть ли среди элементов массива арифметическая прогрессия, с коэффициентом 5? Если результат положительный, сформируйте из найденной прогрессии новый упорядоченный массив.

22) Определить минимальный и максимальный элемент вещественного массива. Найти количество элементов, находящихся между максимальным и минимальным элементами.

Многомерные массивы

1) Дана целочисленная матрица размера MxN, найти количество столбцов, содержащих нулевые элементы.

2) Дана целочисленная матрица размера MxN, найти количество строк, содержащих убывающие последовательности элементов.

3) Дана целочисленная матрица размера MxN, найти количество столбцов, не содержащих ни одного отрицательного элемента.

4) Дана целочисленная матрица размера MxN, найти количество столбцов(или – один, или – нет), содержащих максимальное число отрицательных элементов.

5) Дана целочисленная матрица размера NxN, найти количество диагоналей, параллельных главной, не содержащих ни одного отрицательного элемента.

6) Дана целочисленная матрица размера NxN, найти произведение элементов в тех строках, которые не содержат отрицательных элементов.

7) Дана целочисленная матрица размера MxN, найти номер строки, в которой находится самая длинная серия одинаковых элементов.

8) Дана целочисленная матрица размера NxN, найти количество диагоналей, параллельных побочной, содержащих только отрицательные элементы.

9) Дана целочисленная матрица размера NxN, найти количество диагоналей, параллельных главной, содержащих хотя бы один нулевой элемент.

10) Дана целочисленная матрица размера NxN, определить сумму элементов каждой диагонали,, параллельных главной.

11) Дана целочисленная матрица размера NxN, определить сумму элементов каждой диагонали, параллельных побочной и найти среди этих сумм максимальное значение.

12) Дана вещественная матрица размера MxN, найти сумму элементов матрицы для каждой строки и для каждого столбца. Определить среди полученных значений максимальное и минимальное с указанием принадлежности строке и столбцу.

13) Дана вещественная матрица размера NxN, найти произведение элементов матрицы для каждой строки и для каждого

столбца. Определить среди полученных значений максимальное и минимальное с указанием принадлежности строке и столбцу.

14) Дана вещественная матрица размера NxN, определить, является ли она верхней треугольной матрицей?

15) Дана целочисленная квадратная матрица размера NxN, найти в ней число повторений каждого элемента.

16) Дана целочисленная квадратная матрица размера NxN, определить сумму элементов матрицы, лежащих одновременно ниже побочной и ниже главной диагоналей.

17) Дана целочисленная квадратная матрица размера NxN, определить сумму элементов матрицы, лежащих одновременно ниже побочной и выше главной диагоналей.

18) Дана целочисленная квадратная матрица размера NxN, определить сумму элементов матрицы, лежащих одновременно выше побочной и выше главной диагоналей.

19) Дана целочисленная квадратная матрица размера NxN, определить сумму элементов матрицы, лежащих одновременно ниже побочной и выше главной диагоналей.

20) Дана квадратная матрица размера NxN. Зеркально отразить ее элементы относительно а) горизонтальной оси симметрии; б) вертикальной оси симметрии.

21) Дана квадратная матрица размера NxN. Зеркально отразить ее элементы относительно а) главной диагонали; б) побочной диагонали.

22) Дана квадратная матрица размера NxN. Повернуть данную матрицу по часовой стрелке на угол а) 90 градусов; б) 180 градусов; с) 270 градусов.

4. Требования к оформлению отчёта

Отчёт по практической работе должен включать следующие разделы.

1. Титульный лист (Приложение А).
2. Цель работы и постановку каждого выполняемого задания.
3. Описание входных, выходных данных, модели решения задачи.

4. Скриншоты окон в среде программирования, подтверждающие работоспособность задач и примеров, выполненных в кодах изучаемых ЯП ВУ.

5. Выводы по работе.

6. Ответы на контрольные вопросы.

7. Список использованной литературы и Интернет-источников.

Контрольные вопросы

1. Какие данные могут быть представлены в программном коде структурированным типом «массив»?

2. Возможно ли присутствие в массиве различных типов данных?

3. Чем отличаются структурированные типы от простых в C/C++?

4. До выполнения кода всегда ли должен быть задан размер массива?

5. Что определяет размерность массива?

6. Какому значению равен индекс первого элемента одномерного массива в C/C++?

7. Объясните алгоритм сортировки элементов массива методом вставки (включением элементов).

8. Объясните алгоритм сортировки элементов массива методом обмена. Как ещё называют этот метод?

9. Какой из методов сортировки элементов массива: выбором или Шелла считается более эффективным. Почему?

10. Для чего применяют вложенные циклы к элементам массива?

11. Какие массивы называют многомерными? Приведите пример их объявления и инициализации на C/C++.

Практическая работа №4 ДИНАМИЧЕСКАЯ ПАМЯТЬ И МАССИВЫ

ЦЕЛЬ РАБОТЫ: Исследование работы с динамической памятью в C/C++ и выделение данной памяти под массивы в ходе выполнения программного кода.

1. Основные понятия

Согласно структуре пользовательского приложения и отображения его в ОЗУ, каждой программе предоставляются определённые адреса ячеек памяти (адресное пространство) и область памяти, адресуемая с помощью этого адресного пространства. Данная область памяти делится на несколько областей со своим функциональным назначением(рисунок 10).

Рисунок 10 – Структура распределения памяти под прикладную программу

Участки памяти под код программы и глобальные переменные выделяются на все время выполнения программы и являются постоянной величиной для конкретной среды[6]. Эти области памяти называются, соответственно, сегмент кода (CS-Code segment) и сегмент данных (DS - Data segment). В случае обращения к подпрограммам – возникает необходимость хранить локальные переменные, которые принадлежат вызываемым объектам, для этого

используется сегмент стека (SS - Stack segment) – стековое пространство, размер которого варьируется. Стековая память ограничена и пользователь заранее должен определить, хватит ли ему данной памяти, например, для вызываемых рекурсивных функций - стек может переполниться и программа завершит своё действие. В C/C++ такой способ выделения памяти называется статическим и для пользовательских задач эффективно применяется при небольших объёмах данных, размер которых известен заранее.

Но, количество данных может изменятся в ходе выполнения программного кода, требуемое количество памяти под них не может быть определено заранее. Существует ещё одна область свободной памяти (между постоянной областью памяти и сегментом стека) – называемая динамической памятью (Heap) или «кучей», которая выделяется в C/C++ под данные по мере необходимости *во время выполнения программы*. Этот способ выделения памяти называется динамическим, а область указанной памяти – *динамической,* для загруженной на выполнение в ОЗУ пользовательской задачи (программного кода). Доступ к динамической памяти осуществляется через *указатели* – специальные переменные, объявляемые в программе до выполнения кода.

Указатели предназначены для хранения адресов динамической памяти. После того, как память выделена, она доступна везде, где доступен указатель, адресующий этот участок динамической памяти. В C/C++ после окончания работы с динамическими объектами, выделенную им память в куче, необходимо освобождать, во избежание образования «мусора» и сокращения свободно адресуемого пространства динамической памяти программы. Для этого служат специальные библиотечные функции. Если динамическая память не освобождена до окончания программы, то она освобождается автоматически при завершении программы. Указатели не являются самостоятельными выражениями, они обязательно связываются с другими типами данных, определёнными в конкретном ЯП.

В C++ различают три вида указателей:

1. *Указатели на объект*. Указатель этого вида содержит адрес динамической памяти, в которой хранятся данные определённого типа (простого или структурированного; кроме ссылочного, определённого в языке C/C++ как &, и битового поля).

Записывается указатель на объект по следующему шаблону: <тип> *<имя>;

2. *Указатели на функцию*. Данный указатель хранит *адрес* в CS, по которому располагается исполняемый код функции. Указатели на функции используются для косвенного вызова функции, не через ее имя, а через обращение к переменной, хранящей ее адрес. Кроме того, с помощью указателя можно передать имя функции в другую функцию в качестве параметра. Указатель функции имеет тип «указатель функции, возвращающей значение заданного типа и имеющей аргументы заданного типа»[7].

Записывается указатель на функцию по следующему шаблону: <тип> (*<имя функции>)(список типов аргументов);

Например, создание указателя *bool (*fun1) (float, float);* - задает указатель с именем *fun1* на функцию, возвращающую значение типа *bool* и имеющую два аргумента типа *float*.

3. *Указатели на void*. Эти указатели применяются в тех случаях, когда конкретный тип объекта (адрес которого требуется хранить), не определен. Такой случай возможен, например, когда в одной и той же переменной в разные моменты времени требуется хранить адреса объектов различных типов. Указателю на void можно присвоить значение указателя любого типа, а также сравнивать его с любыми указателями. Единственным условием является преобразование его к какому-либо типу явным образом, до выполнения действий с областью памяти, на которую он ссылается [4].

Примеры указателей в кодах C/C++ на различные объекты:

int a; // целая переменная

*int *pa;* // указатель на целую переменную

const int ca = 1; // целая константа

*const int *pca;* // указатель на целую константу

*int *const cp = &a;* // указатель-константа на целую переменную

*const int *const cc = &ca;* // указатель-константа на целую константу.

В C/C++ для выделения (освобождения) памяти под динамические переменные в куче используется два способа: с помощью директив(операций) *new* и *delete*; с помощью функций *malloc* /*calloc* и *free* (в стиле Си).

Директива *new* позволяет выделить и сделать доступным свободный участок, размеры которого соответствуют типу данных, определяемому именем типа: *new* <имя типа> [<инициализатор>];

В выделенный участок заносится значение, определяемое инициализатором, который не является обязательным элементом. В случае успешного выполнения, *new* возвращает адрес начала выделенного участка памяти; отсутствие нужного участка в свободно адресуемой памяти, приводит к возврату директивой *new* нулевого значения адреса (*NULL*).

В целом, использование указателя и рассматриваемой операции выглядит согласно шаблону:

<указатель> = *new* <имя типа> [<инициализатор>];

И может быть записано в кодах C++ следующим образом:

```
double *p;          //объявление переменной
p=new double;       //выделение памяти для переменной
*p = 0.252525;      //присваивание значения
```

В качестве типа можно использовать, например, стандартные типы *int, long, float, double, char*, так и структурированные массивы, записи, множества, и др. В качестве имени типа в операции *new* может быть использован массив:

```
int *r;
r = new int[5];,
```

где будет выделено 20 байт памяти для массива в 5 элементов типа *int*, если для представления в системе целого стандартного типа используется 4 байта. Причём, размер массива обязательно должен быть определён до вызова операции выделения памяти.

В случае *int *n = new int;* директива *new* выполняет выделение достаточного для размещения величины типа *int* участка динамической памяти и записывает адрес начала этого участка в переменную *n;* кроме того, память под сам указатель выделяется во время компиляции, а память под переменные, адрес начала которых хранится в указателе, будет выделена во время выполнения кода.

Чтобы освободить выделенную *new* динамическую память, используется операция *delete* <указатель>. Например, для вышеуказанной переменной *int *n = new int* это будет так:

```
delete n;
```

Для освобождения памяти, выделенной для массива *r = new int[5];* необходимо указать следующее выражение *delete* [] *r;*

Указатели надо обязательно *инициализировать*. Существуют следующие способы:

1. Присваивание указателю адреса однотипной переменной.

*int *p, a, b[10];*

p = &a;

*int *pm = b;*

2. Присваивание указателю пустого значения *NULL*.

*int *pm = NULL;*

*int *p = 0;*

3. Присваивание указателю адреса выделенной динамической памяти с помощью функций new и malloc.

*int *a = new int;*

*int *m = new int(20);*

*int *n = new int[20];*

*int *k = (int*)malloc(sizeof(int));*

Операцию выделения памяти с помощью *new* использовать предпочтительнее, чем с помощью *malloc*, так как при использовании *new* не надо контролировать размер выделяемой памяти и использовать *sizeof* (размер объекта или типа), это происходит автоматически.

4. В указатель можно вводить непосредственно конкретные данные адреса с клавиатуры, используя директиву форматного ввода (характерную для ЯП Си) *scanf()* со спецификацией *%p*.

Указатели это особые переменные, с ними можно выполнять следующие операции:

— разадресация или косвенное обращение к объекту (*);
— присваивание;
— сложение с константой;
— вычитание;
— инкремент (++), декремент (- -);
— сравнение;
— приведение типов.

При работе с указателями часто используется операция получения адреса (& - оператор ссылка). Ссылка представляет собой *синоним имени*, который был указан при инициализации ссылки. Если мы изменяем ссылку, то значит изменяется и переменная, на

которую она ссылается. Общий формат записи ссылки: <тип>&<имя>;

Например:

int a;

int &sa = a;

const char &b = '\t' ;

В языке C/C++ имя массива является указателем на первый элемент. Например, если определен массив *int a[n]*, то в переменной *a* содержится адрес первого элемента массива и значение переменной *a* в этом случае можно присваивать указателю того же типа. Если же задан двумерный массив, например, *a[n][m]*, то для него определен массив указателей *a[n]*. В *i*-м элементе массива *a[i]* содержится адрес первого элемента *i*-й строки. В переменной *a* содержится адрес первого элемента массива, тогда справедливо утверждение, что ссылка *&a[0][0]* эквивалентна *a*.

2. Методические указания

1. Важно отметить, что указатель (а ему сопутствует символ «звёздочка» - *) относится непосредственно к имени переменной или константы(в общем виде – объекта), поэтому для того, чтобы объявить несколько указателей, требуется ставить «*»перед именем каждого объекта: *int *a, x, *y;* Выполните задание переменных и указателей различного типа.
2. При определении указателя надо стремиться выполнить его инициализацию, то есть присвоение начального значения. Непреднамеренное использование неинициализированных указателей — распространенный источник ошибок в программах. Инициализатор записывается после имени указателя либо в круглых скобках, либо после знака равенства. Инициализируйте переменные из п.1.
3. Выполните пример 4.1 программного код с указателями и объясните полученный результат.

Пример 4.1 Исследование указателей.

```
void main()
{
	setlocale(LC_ALL, "rus");
	int* pa, * pb;
```

```
pa = new int;
*pa = 2;
pb = pa;
cout << *pa << "  " << *pb << "\n";
cout << pa << "  " << pb << "\n";
pb = new int;
*pb = 8;
cout << *pa << "  " << *pb << "\n";
delete pa;
pa = pb;
cout << *pa << "  " << *pb << "\n";
delete pa;
system("pause");
}
```

4. *Динамические массивы* создаются в памяти с помощью указателя и операции *new*, при этом, указатель всегда задаёт адрес первого элемента, с которого начнётся «построение» массива, а по операции *new* выделяется необходимое количество непрерывной области памяти для размещения всех элементов массива во время выполнения программного кода. Например, для одномерного массива из пяти элементов вещественного типа: *float*a=new float[5];* или для многомерного массива размера N=3(строки) и M=5(столбцов): *float *a = new float[N*M];*

5. Альтернативный способ создания динамического массива, опираясь на — использование функции *malloc* библиотеки Си: *int n = 10; float *q = (float *) malloc(n * sizeof(float));* Добавьте в программный код выражения для задания динамических массивов разной размерности и разными способами.

6. Преимущество динамических массивов состоит в том, что объем памяти, выделяемой под массив варьируется во время выполнения программы, а не фиксируется на этапе компиляции заданием максимального размера массива. Доступ к элементам динамического массива осуществляется аналогично статическим массивам – по индексу: *p[i]* или **(p+i)*, обращение к *i*–му элементу одномерного массива *p*. Доступ к *p(i,j)*–му элементу двумерного массива можно осуществить обращением: *p[i][j*], или другими способами: **(p [i]+j)* или **(*(p +i)+j)*. Выполните различные обращения к элементам динамических массивов в отлаживаемом

коде. В конце работы с массивами, освободите память из-под данных структур.

7. Правильнее всего, резервировать память под многомерные массивы во время выполнения программы следующим способом: *int n,m; int **a; … a=new int*[n]; for(int i=0;i<n;i++) a[i]=new int[m];* Используйте данный способ, зациклив программный код по вводу с клавиатуры различных значений размерности массива. Произведите тестирование и отладку, обязательно добавив в цикл освобождение памяти из под массивов (пример 4.2-4.4).

Пример 4.2 Реализация динамических массивов

```
int main()
{
 int n, m ;
            float* b = new float[n];
            float* a = new float[n * m];
      for (int i = 0; i < n; i++) {//одномерный массив
            b[i] = rand() % 100 - rand() % 100;
                  cout << b[i] << " ";
            }
            cout << endl;
            for (int i = 0; i < n; i++) {//Двумерный массив
                  for (int j = 0; j < m; j++) {
                        (a + i * m)[j] = rand() % 10 - rand() % 10;
                        cout << (a+i*m)[j] << "\t";
                  }
                  cout << endl;
            }
      delete[] a; delete[] b;
      //************************************
      system("pause");
      return 0;
}
```

Пример 4.3 Инициализация многомерного массива

```
#include <iostream>
using namespace std;
```

```
int main()
{setlocale(LC_ALL, "RUS");
	int **a;
	int n, m;
	cout << "\n Введите размерность массива:"
		<< "\nили(Ctrl + z Enter) – выход из программы\n";
		cin >> n >> m;
		while (!feof(stdin))// Начало цикла. Цикл выполняется до тех пор, пока не будет введен символ Ctrl + z.
	{		a = new int *[n];
			for (int i = 0; i < n; i++)
			a[i] = new int[m];
			//Заполнение массива произвольными числами из диапазона [-10;10]
			for (int i = 0; i < n; i++) {
				for (int j = 0; j < m; j++) {
					a[i][j] = rand() % 10 - rand() % 10;
					cout << a[i][j] << "\t";
				} cout << endl;
			}
		for (int i = 0; i < n; i++)
		//освобождение памяти для указателей a[i]
			delete a[i];
		//освобождение памяти для значений
		delete[] a;
		//*************************************
		cout << "\nВведите размерность массива:" << "\n или(Ctrl + z Enter) – выход из программы\n";
		cin >> n >> m;}
//*************************************
system("pause");
	return 0;
}
```

Пример 4.4 Отрисовка шахматной доски заданного размера с определённой шириной клетки

```
#include <iostream>
```

```
using namespace std;
int main()
{
	setlocale(LC_ALL, "RUS");
	int  n, m, s, pr1,pr2,priz, k = 0, l = 0;
	int **mas;
// pr1,pr2,priz - индикаторы "заполнителя клетки" 0 или 1
//mxm - размер клетки (в позициях)
// sxs - размер игрового поля в клетках
// nxn -  размер игрового поля в позициях – вспомогательная переменная
//k, l -  вспомогательные переменные
cout << "Введите размер игрового поля(в клетках) и размер клетки:\n"
	<< "\n(Ctrl + z Enter – выход из программы\n";
	cin >> s >> m;
	while (!feof(stdin))// Цикл выполняется до тех пор,
				// пока не будет введен символ Ctrl + z.
	{n = s * m;
	mas = new int* [n];
	for (int i = 0; i < n; i++)
		mas[i] = new int[n];
cout << "\nИгровое поле в заданных значениях:\n";
		for (int i = 0;i < n; i++) {
			if ((i) % m == 0) l = int(i / m);
	//Как только должна быть отрисована новая клетка - //смена "заполнителя"(0 или 1)
	if (((m * l) <= i) && ((l % 2) == 0)) pr1 = 1;
		else pr1 = 0;
		for (int j = 0;j < n;j++) {
			if ((j) % m == 0) k = int(j / m);
	//Смена заполнителя для новой клетки
	if (((m * k) <= j) && ((k % 2) == 0)) pr2 = 1;
				else pr2 = 0;
	/*Анализируем слои по вертикали(pr1) и по горизонтали(pr2)*/
		if (pr1 == pr2) priz = 1;
		else priz = 0;
```

```
                mas[i][j] = priz;
                    }
                }
// Отрисовка поля**********************************
                for (int i = 0;i < n; i++) {
                    for (int j = 0;j < n;j++) {
                        cout << " " << mas[i][j];
                    }
                    cout << endl;
                }
                for (int i = 0;i < n; i++)
        //освобождение памяти для каждого указателя - mas[i]
                    delete[] mas[i];
                //освобождение памяти для mas
                delete[] mas;
                //**************************************
cout << "Введите размер игрового поля(в клетках) и размер клетки:\n"
    << "\n(Ctrl + z Enter – выход из программы\n";
                cin >> s >> m;
    }
    return 0;
}
```

Данный алгоритм можно оптимизировать – попробуйте это сделать самостоятельно.

8. Ответьте на контрольные вопросы.
9. Составьте отчёт по практической работе.

3. Варианты заданий для самостоятельной работы

1. Выполнить варианты для самостоятельной работы (п.3) практической работы 3, используя динамическое выделение памяти под структурированный тип данных – массив. Предусмотреть интерфейс программы с возможностью тестирования кода для различных значений размерности массива, не выходя из программы.
2. Выполнить несколько заданий (2-3) с заменой двумерных массивов на трехмерные. Во всех вариантах использовать инструкции освобождения динамической памяти.

3. Использовать в 2-3х примерах самостоятельной работы выделение динамической памяти средствами ЯП ВУ Си.

4. Требования к оформлению отчёта

Отчёт по практической работе должен включать следующие разделы.

1. Титульный лист (Приложение А).
2. Цель работы и постановку каждого выполняемого задания.
3. Описание входных, выходных данных, модели решения задачи.
4. Скриншоты окон в среде программирования, подтверждающие работоспособность задач и примеров, выполненных в кодах изучаемых ЯП ВУ.
5. Выводы по работе.
6. Ответы на контрольные вопросы.
7. Список использованной литературы и Интернет-источников.

Контрольные вопросы

1. Какую память называют динамической по отношению к программному коду?
2. Что такое указатель и его назначение.
3. Достаточно ли объявления указателя в программе, чтобы получить доступ к динамической памяти?
4. Перечислите виды указателей и форматы задания указателя для каждого вида.
5. Обязательна ли инициализация указателей?
6. Какие существуют способы инициализации указателя, перечислите на примерах.
7. Можно ли инициализировать указатель кодом int *p = NULL.
8. Какие операции над указателями возможны.
9. Как организовать «динамический» массив в программном коде?
10. Как через имя и указатель можно обратиться к любому элементу массива?
11. Перечислите способы доступа к элементам динамического массива через указатель.

12. Как задать доступ к элементам трёхмерного массива. Покажите на примере.
13. Как освободить выделенную динамическую память.
14. Покажите на примере выделение и освобождение динамической памяти средствами ЯП ВУ С.

Практическая работа №5 ОСНОВЫ РАБОТЫ С ТЕКСТОВОЙ ИНФОРМАЦИЕЙ И ФАЙЛАМИ

ЦЕЛЬ РАБОТЫ: Изучить основные принципы работы с символьными данными и текстовыми файлами, закрепить практические навыки по отладке и тестированию задач, использующих внешние источники хранения данных – файлы.

1.Основные понятия

В настоящее время большинство данных, которые необходимы людям при управлении экономикой и социальной сферой с помощью ЭВМ – слабо структурированы, как правило, это инструкции, поправки, алгоритмы в словесной форме, качественные показатели, примечания, и т.д. Чтобы организовать работу с такими данными в ЯП ВУ используются типы данных, позволяющие хранить символы, строки, тексты.

Символьный тип (*char*) - для задания любого символа, предназначенного и понимаемого данной ВС. Как правило, это 1 байт. Тип *char*, может быть со знаком или без знака. В величинах со знаком можно хранить значения в диапазоне от -128 до 127. При использовании спецификатора *unsigned* значения могут находиться в пределах от 0 до 255. Этого достаточно для хранения любого символа из 256-символьного набора кодовой таблицы ASCII. То есть, у любого символа или сочетания клавиш, возможных с клавиатуры, - есть код, выражаемый соответствующим числом из кодовой таблицы. Пример, демонстрирующий данный факт, может выглядеть следующим образом:

```
#include <iostream>
using namespace std;
int main()
{
	setlocale(LC_ALL, "RUS");
	int ch;
	cout << " Введи код символа (Ctrl + z – выход из программы ): ";
	cin >> ch;
	while (!feof(stdin))
```

```
        {
        cout << "Коду " << ch << "  символ "
        << (char)ch << "\n\n"; // символ выводится в консоль.
        cout << "Введи число (код символа) или для выхода из программы Ctrl + z \n";
        cin >> ch;
        }
        return 0;
    }
```

Рассматриваемый тип *char* служит основой для задания текстовой строки, как массива символов. Строки в C/C++ представляют собой массив символов, заканчивающийся нуль-символом (‘\0’). Нуль-символ, в свою очередь, определяет фактическую длина строки. Даже пустая строка заканчивается этим символом. Например, *char st[20] = "World!";* Если представить заданную строку как массив, то это следующие символы: ‘*W*’, ‘*o*’, ‘*r*’, ‘*l*’, ’*d*’, ‘*!*’, ‘\0’. Всего выделено 20 байт, 6 байт занято символами, седьмой – символом окончания строки, 14 байт свободные.

Если строка при определении инициализируется, её размерность можно опускать, например: *char st[ ] = "World!";* Длину такой строки можно определить с помощью функции *strlen*. Кроме того, используя изложенный в практической работе 4 материал по указателям, в применении к строковым переменным, можно задать ту же строку как: *char *st = "World!";* Пример формирования с помощью указателя строки символов можно представить следующим программным кодом:

```
    int main()
    {
        setlocale(LC_ALL , "RUS");
        char* sr = new char[20];
    char* ex = new char[20],*p = ex;// указатель на начало строки
        cin >> sr;
        while (*sr != 0) *p++ = *sr++;
        *p = 0; // завершающий конец строки нуль
        cout << ex <<"с длиной строки "<< strlen(ex);
        return 0;
    }
```

Таким образом, со строками можно работать через массивы и указатели, но в С определён перечень библиотечных функций из модуля <string.h>, а в стандартной библиотеке С++ определён пользовательский класс string, которые позволяют более продуктивно работать со строками и символьной информацией. С помощью встроенных функций и методов класса можно проводить индексацию, присваивание, сравнение, добавление, объединение строк, поиск подстрок и др. [4]

strlen(st) возвращает длину строки st;

strchr(st,ch) возвращает значение указателя на первое вхождение символа в строке *st*, который совпадает с символом в переменной *ch*. Если символ не найден, функция *strchr* возвращает нулевой указатель.

strcpy(st1, st2) копирует строку st2 в строку st1;

strcat(st1, st2) присоединяет строку st2 к строке st1;

strstr(st1, st2) определяет, содержится ли строка st2 в строке st1. Если содержится, то *strstr* возвращает значение указателя на первое вхождение st2 в st1. Иначе strstr возвращает нулевой указатель;

strcmp(st1, st2) сравнивает в алфавитном порядке строки st1 и st2: при st1 < st2 значение сравнения отрицательно; при st1 = st2 значение сравнения равно нулю; при st1 > st2 значение сравнения положительно.

Файл – это именованная совокупность байтов или логическое устройство (например, *prn, con*) – потенциальный источник или приемник информации. Как правило, когда мы работаем с файлами, то подразумеваем внешние запоминающие устройства для хранения информации.

Для того, чтобы обратиться к файлу, в программном коде любого ЯП ВУ есть файловый тип, через него создаётся логическое имя файла в виде файловой переменной. В стиле С это:

*FILE *<имя>;*

При вводе-выводе данные рассматриваются как поток байтов. Поток можно открыть для чтения и(или) записи в двоичном или текстовом режиме. Чтобы привязать действительное имя физического файла на носителе к логическому имени, используется функция *fopen()*. Чтобы закрыть работу с физическим файлом применяется функция *fclose()*.

Например,

*FILE *f1;*

f1 = fopen(“<имя файла на носителе>”,”<режим доступа к файлу>”);

В данной записи:

<имя файла на носителе> – имя конкретного файла на внешнем носителе;

<режим доступа к файлу> определяет, какие действия возможны с открываемым файлом.

Возможными действиями (режимами доступа) могут быть:

r – файл открыт для чтения;

w – файл открыт для записи;

a – файл открыт для записи, причём, в конец файла;

r+ – файл открыт для чтения и записи;

w+ – файл открыт для записи и чтения.

Функция *fopen()* возвращает указатель файла. Если при открытии файла произошла ошибка, то функция возвращает *NULL*.

Учитывая это, логично, начальную работу с файлом представить следующим образом:

```
if ((f1 = fopen ("test.txt","w+”)) != NULL)
{ printf ("Файл открыт, \n"); … fclose(f1); }
else
printf (“Невозможно открыть файл! \n”);
…
```

Определены следующие функции для работы с файловыми переменными:

форматный ввод для записи информации из файла в память компьютера *fscanf(f1, <управляющая строка>, <список переменных>);*

форматный ввод для записи информации из памяти компьютера в файл *fprintf(f1,<управляющая строка>,<список переменных>).*

Текстовый файл является файлом последовательного доступа, его можно представить как набор строк произвольной длины. Последовательный файл отличается от файлов с другой организацией тем, что чтение (или запись) из файла (в файл) ведутся байт за байтом от начала к концу. Для использования функций ввода-вывода в стиле Си требуется подключить к программе заголовочный файл <*stdio.h*> или <*cstdio*> . Необходимо отметить,

что не все файлы обладают одинаковыми возможностями, файл на жестком диске предоставляет прямой доступ к своим записям, а некоторые принтеры последовательного доступа, но все потоки в файловой системе языка Си одинаковы!

Язык C++ полностью поддерживает файловую систему языка С, поэтому выше указанным механизмом работы с файловыми переменными можно воспользоваться и при написании кода на C++. Однако, в языке C++ определена собственная объектно-ориентированная, система ввода-вывода, включающая функции и операторы. Эта система полностью дублирует все функциональные возможности средств ввода-вывода языка Си, поэтому программисты могут использовать эти два подхода как равные. Второй подход ориентирован на потоковый ввод(вывод), существующий в C++.

Существует пять предопределенных потоков, которые открываются в начале работы программы:

stdin - стандартный ввод;

stdout - стандартный вывод;

stderr - стандартный вывод сообщений об ошибках;

stdaux - стандартный дополнительный поток;

stdprn - стандартная печать.

Первые три потока по умолчанию относятся к консоли. Эти указатели можно использовать в любой функции ввода-вывода там, где требуется указатель потока, такими функциями могут быть:

fread и *fwrite* - чтение и запись потока байтов;

getc, fgetc - чтение символа из потока, из стандартного потока *stdin — getchar*;

putc, fputc - запись символа в поток, в стандартный поток *stdout — putchar*;

fgets - чтение строки из потока, из стандартного потока *stdin — gets*;

fputs - запись строки в поток, в стандартный поток *stdout — puts*.

Преимущество использования потоков в том, что они легче в простых случаях ввода/вывода, не требующих форматирования и потоковые операции можно переопределить для собственных классов (этот тип данных будет подробно рассмотрен в практических работах в рамках курса “Объектно-ориентированное программирование”).

2. Методические указания

1. Известно, что все символы, используемые типом *char*, заданы в таблице ASCII-кодов, например, если вывод идёт на консоль. Часто требуется знать значение ASCII-кода символа, заданного переменной типа *char*. При выводе на консоль переменной *d* (заданной как *char d;*) – будет выдан соответствующий символ, а при преобразовании значения величины типа *char* к значению типа *int*, можно получить ASCII-код символа:

cout << (int) d << endl;

Аналогично, при использовании потока *cin* для считывания переменной типа *char,* можно использовать код: *cin >> d;*

Например, при выполнении указанного кода по считыванию с консоли одного символа, переменная получает значение, равное его ASCII-коду.

2. Выполните следующий код (Пример 5.1) для вывода на экран английского алфавита. Измените его и выдайте на консоль все символы русского алфавита.

Пример 5.1 Вывод символов английского алфавита.

```
void main()
{ char d;
setlocale(LC_ALL,"RUS");
   cout << "Символы английского алфавита:\n";
   for (d = 'a'; d <= 'z'; d++) // Цикл от a до z.
      cout << d << " "; // Вывод параметра цикла d
   cout << "\n";
   }
```

3. Переменным типа *char* можно явно присваивать числовые значения – коды символов (Пример 5.2) или организовать посимвольное считывание всего входного потока (Пример 5.3).

Пример 5.2 Вывод символа и его ASCII-кода

```
#include <iostream>
using namespace std;
```

```
int main()
{
    unsigned char d = 'B';
    cout << d << " " << (int)d << endl;
    d = 126;      // char присвоено числовое значение
    cout << d << " " << (int)d << endl;
    return 0;
  }
```

Пример 5.3 Последовательное посимвольное считывание входного потока

```
#include <iostream>
using namespace std;
int main()
{
  char ch;
  string a;
  while (cin >> ch) // Цикл пока считывание успешно
  {
    cout << ch << " " << (int)ch << endl;
    // возможны любые непротиворечивые действия с символом
    }
  return 0;
  }
```

Для того, чтобы корректно завершить ввод входного потока при вводе символов с клавиатуры, необходимо нажать сочетание клавиш *Ctrl-d* в ОС Linux и *Ctrl-z* в операционной системе линейки Windows.

Чтобы при вводе не игнорировались символы–разделители: пробелы, символы новой строки и табуляции, необходимо для потока ввода *cin* установить манипулятор *noskipws* из библиотеки *sstream* помощи инструкции: *cin >> noskipws;*

4. В языке С можно формировать с помощью типа *char* также массив символов и строки. Массив - множество элементов одного типа. Строка, как было указано выше, обязательно заканчивается ‘\0’, называемым «концом строки». Принципиальная

же разница между строкой и массивом состоит в том, что присутствие в строке символа ‘\0’ неявно задает размер массива [4].

Специального формата для объявления строки в языке С не существует, инициализацию строк можно проводить по-разному, выполните пример 5.4 и проверьте разные способы инициализации и ввода строк.

Пример 5.4 Ввод и вывод строк

```
void main()
{
   setlocale(LC_ALL, "RUS");
   char ch[80]="", a[80];
   const char* s1="Hello world";
   int count = 0;
   string s;
    cout << " " << s1<< "\n";
    cout << "Введите строку символов \n";
      getline(cin, s);
   cout << " " << s << "\n";
   cout << "Введите строку символов (Ctrl + z Enter – конец ввода)\n";
   while (!feof(stdin))
   {    gets_s(ch);
      count ++;
      cout << count <<". " << ch << "\n"; }
 do {
      cin >> a;
      cout << "! " << a << "\n";
      cout << "Продолжать ввод? (Enter – да)\n";
   } while (_getch() == 13);
}
```

Для ввода строк с клавиатуры можно использовать функции: *scanf()* – функция форматного ввода строки; *gets()*– функция бесформатного ввода строки; *cin* – директива бесформатного ввода строки.

5. Работать со строками в C/C++ , как с массивами символов, которые оканчиваются на нулевой байт '\0' не безопасно, и действия с такими строками могут носить непредвиденный характер.

Поэтому, рекомендуется для хранения строк в C++ использовать тип std::string из модуля <string> , а в C для работы со строками символов имеется стандартные функции из библиотеки, содержащейся в файле <string.h>.

В примере 5.4 показан также строковый тип string. Добавьте в листинг кода для переменной *s* типа *string* различные операции для одной строки и нескольких строк (создайте ещё несколько переменных типа string) и проверьте их действие.

Например:

Cout<< “В слове”<< s << “содержится ” << s.size() << “слов \n”;

Сложите строки, осуществив конкатенацию двух или трёх строк.

Для строки типа *string*, измените размер, используя метод *resize.*

string s1=”Hello, World!”;

s1.resize(17,’w’);

s1.resize(6);

Необходимо обратить внимание на функцию getline (cin, s) в примере, так как она позволяет считывать строки со всеми пробелами и разделителями.

6. Таким образом, работать с текстом можно по-разному, но всегда нужно проверять условия конца считывания символов, во избежание дальнейших ошибок. Особенно, это касается работы с файловым типом. В этом случае, оправдано применение функций *feoff* и *error:*

— *int feof (FILE*)* возвращает не равное нулю значение, если достигнут конец файла, а иначе – «ноль»;

— *int ferror (FILE*)* возвращает не равное нулю значение, если обнаружена ошибка ввода/вывода, а иначе – «ноль».

В примере 5.5 показаны различные способы чтения данных из текстового файла, с проверкой указанных значений.

Пример 5.5 Чтение текстового файла

```
#include <iostream>
#include <stdlib.h>
#include <string.h>
//#define CRT_SECURE_NO_WARNINGS
//#pragma warning(disable: 4996)
```

```
using namespace std;
int main()
{
	setlocale(LC_ALL, "RUS");
	FILE* f1;
	char c1;
	char* st[20];
	int k , i, priz;
	system("cls");
	cout << "\nВведите способ чтения файла"
		<< "\n1 - посимвольно"
		<< "\n2 - построчно"
		<< "\n(Ctrl + z Enter – выход из программы\n";
	cin >> priz;
	while (!feof(stdin))// Начало цикла. Цикл выполняется до тех пор, пока не будет введен символ Ctrl + z.
		{
			if ((f1 = fopen("1.txt", "r")) != NULL)
{ // Если файл открыт, выполняется блок операторов в скобках.
			//чтение файла 2 способами 1 - посимвольно,
				// 2 - построчно.
				// 1 - способ**********************
				switch (priz)
				{
			case 1: cout << "Программа выводит текст, "<< "который вводится из файла посимвольно. \n";
					c1 = ' ';
				 while (!feof(f1)){
					cout << c1;   c1 = getc(f1);// В переменную c1 из файла вводим символ.
					 // Выводим символ на экран.
				}  // Цикл, пока не конец файла.
				break;
			case 2:	//2 способ************
			cout << "Программа выводит текст, "
		<< "который вводится из файла построчно \n";
				k = 0;
				 while (!feof(f1))
```

```
                    {
                st[k] = new char[81];
                fgets(st[k], 79, f1);
        k++; // Формируем индекс для текущей строки.
                    }
        printf("\n Из файла ввели %d строки:\n", k);
                for (i = 0; i < k; i++)
                    printf("%s", st[i]);
                break;
            default:
cout << "\nКоманда меню с номером" << priz<< "отсутствует\n";
                break;
            }
            fclose(f1);
        }
        else // Если файл не открыт, выдается сообщение об ошибке.
            printf("Ошибка при открытии файла!!!\n");
        //**********************************
        cout << "\nВведите способ чтения файла"
            << "\n1 - посимвольно"
            << "\n2 - построчно"
        << "\n(Ctrl + z Enter – выход из программы\n";
        cin >> priz;
    }
    system("pause");
}
```

7. Ответьте на контрольные вопросы.
8. Составьте отчёт по практической работе.

3. Варианты заданий для самостоятельной работы

Для всех вариантов заданий – текстовые данные должны быть получены из файла!

1) Подсчитать количество слов во введенном многострочном тексте.

2) Подсчитать количество предложений во введенном тексте.

3) Найти и вывести на экран первые слова всех предложений текста.

4) Найти и вывести на экран последние слова всех предложений текста.

5) Найти в тексте все слова(слово) максимальной длины.

6) Найти в тексте все слова(слово) минимальной длины.

7) Пронумеровать все предложения текста и выдать номер предложения, в котором больше всего коротких слов(размер слов задать с консоли).

8) Пронумеровать все предложения текста и выдать номер предложения, в котором больше всего раз повторилось заданное слово.

9) Определить номер предложения, в котором больше всего раз встретилось слово минимальной длины из текста.

10) Определить номер предложения, в котором больше всего раз встретилось слово максимальной длины из текста.

11) Определить, в каком по счету предложении впервые встретилось заданное слово.

12) Найти и записать в столбик все двойные комбинации букв, встретившиеся в словах текста.

13) Найти и напечатать слова, которые начинаются и заканчиваются одной и той же буквой.

14) Найти в тексте все встретившиеся цифры.

15) Найти в тексте все гласные буквы и удалить их. Полученный текст выдать на экран.

16) Подсчитать сколько раз встретилась в тексте каждая гласная буква.

17) Подсчитать. сколько раз встретилась в тексте каждая цифра.

18) Найти все строки в тексте, в которых 5 раз встретилась буква «а».

19) Найти в каждой строке текста количество цифр – это значение вывести в начале каждой строки.

20) Имеется текст с двоичной записью чисел. Провести анализ цифр и выдать текст без незначащих нулей.

21) Найти все управляющие символы в тексте и выдать их на консоль.

22) Записать текст на экран в обратном порядке: сначала последнее предложение, предпоследнее и т.д.

23) Определить количество точек и запятых в тексте.

24) Осуществить проверку правильности написания пробелов: слева от знака пунктуации пробел не ставится, справа ставится один пробел. Исправленный текст напечатать.

25) Найти все дробные числа в тексте. Подсчитать их количество.

26) Найти все дробные числа в тексте. Округлить их до одного знака после запятой.

27) Имеется текст – вывести на экран в строку через пробел все заголовочные буквы.

28) В каждой строке выделить последние слова и выдать их в столбик на консоль.

29) Подсчитать во всех предложениях количество слов и выдать текст в упорядоченном виде: в соответствии с рейтингом по возрастанию количества слов.

30) Найти слова в тексте, у которых первая и последняя буквы – одинаковы. Количество таких слов сосчитать.

4. Требования к оформлению отчёта

Отчёт по практической работе должен включать следующие разделы.

1. Титульный лист (Приложение А).
2. Цель работы и постановку каждого выполняемого задания.
3. Описание входных, выходных данных, модель решения задачи.
4. Скриншоты окон в среде программирования, подтверждающие работоспособность задач и примеров, выполненных в кодах изучаемых ЯП ВУ.
5. Выводы по работе.
6. Ответы на контрольные вопросы.
7. Список использованной литературы и Интернет-источников.

Контрольные вопросы

1. Каким образом можно задать текст в алгоритме кода.

2. Перечислите преимущества использования внешних источников хранения данных.
3. Какие операции можно выполнять над символьными данными.
4. Как можно задать строки текста и обратиться к ним.
5. Перечислите основные функции, которые позволяют работать с символьными типами.
6. С каким доступом бывают файлы и в чём их отличие.
7. Какие начальные и завершающие операторы должны быть выполнены при работе с файлами? Объясните их назначение.
8. Почему массивы символов нежелательно использовать для задания строк в C/C++?
9. Какие функции со строками позволяют находить необходимые сочетания символов в тексте?
10. Укажите признаки конца строки, файла?

Практическая работа №6 Структурированные типы. СТРУКТУРЫ

ЦЕЛЬ РАБОТЫ: Изучить особенности построения типа данных «структура», освоить формат доступа к элементам структуры и к элементам массива структур. Научится создавать текстовые файлы с массивом структур, освоить технику ввода и вывода элементов структур.

1. Основные понятия

В объективном мире, наряду с однородными данными, встречается большое множество объектов, которые характеризуются разными наборами параметров: количественными, текстовыми, логическими…. Поэтому, решая задачи на ЭВМ с такими данными, необходимо иметь тип данных для описания сущностей с различными параметрами. Таким типом является в С/С++ структура, а в языках программирования, подобно Паскаль, - тип называется «запись». Это статический структурированный тип для задания неоднородных данных.

Кроме того, в языке С++ структура является прототипом класса, у которого отсутствует динамическая составляющая – методы, и обладает всеми свойствами класса.

Для задания структур в ЯП ВУ с С-подобным синтаксисом, используется зарезервированное слово *struct* и сам тип задаётся следующим образом:

Struct <имя>
{
тип <имя элемента 1>;
тип <имя элемента 1>;
…
тип <имя элемента 1>;
}< список описателей, необязательно>;

Элементы структуры называются *полями структуры* и могут иметь любой тип, за исключением, типа данной структуры. Однако, можно в качестве типа элементов использовать указатель на данную

структуру, что имеет смысл при создании элементов динамических структур, таких как списки, деревья.

Если отсутствует имя структуры, то обязательно должен быть указан список описателей переменных, указателей или массивов - тогда заданная структура является типом элементов этого списка:

Например,

```
struct {
char name[10];
char fio[20];
char otch[30];
int nom;
float schet;
}work[50], *pr;
```

Если список отсутствует, то описание структуры задаёт новый тип, который далее будет использоваться для создания новых переменных, например:

```
struct all
{
char name[10];
char fio[20];
char otch[30];
int nom;
float schet;
} ;
all work[50], *pr;
```

Как только структура объявлена, после этого сразу можно использовать имя структуры, а содержание её определить позже в программном коде. Например:

```
struct Tree;
struct All
{
Tree *pr;
All *left, *right;};
struct Tree { char fio[100]; int a; };
```

Инициализировать структуру можно после её объявления, перечислив назначенные начальные значения полям структуры в фигурных скобках в порядке следования полей. Например,

```
struct {
```

char name[10];
char fio[20];
char otch[30];
int nom;
float schet;
}All={”Alecsander”,”Ivanov”,”Ivanovich”,55, 300.77};

В целом, со структурами, можно выполнять следующие операции:

— присваивать одну структуру другой, если их тип совпадает;

— передавать в функцию в качестве параметра;

— возвращать, как значение функции;

— пользователь сам может переопределить операции со структурой [5];

stract All {/...*/};*
*All xxx, work[50], *pr;*

Доступ к элементам структуры осуществляется через имя структуры следующим образом: *xxx.name=”Ivanov”; xxx.nom=55; work[11].schet=300.77;* или при обращении через указатель: *pr -> fio = ”Ivanov”;*

2.Методические указания

1. Как было отмечено, полями структуры могут быть стандартные типы или типы, определённые пользователем ранее. Вполне возможно, определение пользователем структуры, в которой одним из полей будет элемент типа «структура». Тогда обращение к полям в виде структуры должно содержать двухступенчатый выбор. Кроме того, в структурах – определённой и вызывающей могут использоваться одинаковые имена полей. Выполните пример 6.1, чтобы самостоятельно убедиться в этих положениях.

Пример 6.1 Различные способы задания элементов структур

```
int main()
{
   setlocale(LC_ALL, "RUS");
   const int n1=4;
   int n=0;
```

```
struct All { int a; char x[5]; };
struct Boll { All a; double x; }x[n1];
cout << "Введите элементы структуры Boll: \n";
while (n < n1)
{
cin >> x[n].x ;
cin >> x[n].a.x ;
cin >> x[n].a.a;
   n++;
}
for (int i = 0; i < n1; i++)
{//вывод  элементов структуры Boll
cout << i << ". " << x[i].a.a << x[i].a.x  << x[i].x << "\n";
}
return 0;}
```

2. Работа со структурами, как правило, ведётся с использованием внешнего источника данных - файла, когда объём исходных данных можно задавать произвольным количеством записей. Выполните пример 6.2.

Пример 6.2. Массив структур.

Имеется файл с информацией о студентах учебного заведения, представленный следующей структурой записей: фамилия, имя, отчество, пол, факультет, курс, группа, оценки по пяти основным предметам и место проживания. Требуется получить информацию из файла, упорядочить обучающихся по фамилии и выдать список на консоль [4].

```
int main()
    {
        setlocale(LC_ALL, "RUS");
        const int n1 = 100;
        int n;
        struct styd{
            char fam[20];
            char name[15];
            char ot[20];
            char pol;
```

```
        char fakul[3];
        int kurs;
        int gruppa;
        int a[5];
        char city[15];};
    FILE* f1;
    styd sp[n1], sp1;
    int i = 0, im;
    system("cls");
    if ((f1 = fopen("список.txt", "r")) == NULL)
    {printf("Ошибка при открытии файла!!!\n");
// Если файл не открыт, выдается сообщение об ошибке.
        goto m1;}
    else {//Исходные данные из файла
        while (!feof(f1)){
            fscanf(f1, "%s ", &sp[i].fam);
            fscanf(f1, "%s ", &sp[i].name);
            fscanf(f1, "%s ", &sp[i].ot);
            fscanf(f1, "%c ", &sp[i].pol);
            fscanf(f1, "%s ", &sp[i].fakul);
            fscanf(f1, "%d ", &sp[i].kurs);
            fscanf(f1, "%d ", &sp[i].gruppa);
            for (int j = 0; j < 5; j++)
                fscanf(f1, "%d ", &sp[i].a[j]);
            fscanf(f1, "%s\n", &sp[i].city);
            printf("\n");
            printf("%s ", sp[i].fam);
            printf("%s ", sp[i].name);
            printf("%s\t", sp[i].ot);
            printf("%c ", sp[i].pol);
            printf("%s ", sp[i].fakul);
            printf("%d ", sp[i].kurs);
            printf("%d\t", sp[i].gruppa);
            for (int j = 0; j < 5; j++)
            printf("%d ", sp[i].a[j]);
            printf("\t%s", sp[i].city);
            i++;}
        fclose(f1);
```

```
            n = i;
    // Сортировка массива структур
            cout << "\nСтуденты после сортировки\n";
            for (i = 0; i < n - 1; i++)
            {
                im = i;
                for (int j = i + 1; j < n; j++)
                if (strcmp(sp[im].fam, sp[j].fam) > 0) im = j;
                sp1 = sp[i];
                sp[i] = sp[im];
                sp[im] = sp1;
            }
            // Вывод отсортированного массива
            for (i = 0; i < n; i++)
            {
                printf("%s ", sp[i].fam);
                printf("%s ", sp[i].name);
                printf("%s\t", sp[i].ot);
                printf("%c ", sp[i].pol);
                printf("%s ", sp[i].fakul);
                printf("%d ", sp[i].kurs);
                printf("%d\t", sp[i].gruppa);
                for (int j = 0; j < 5; j++)
                printf("%d ", sp[i].a[j]);
                printf("\t%s\n", sp[i].city);
            }
    }
    m1: system("pause");
        return 0;
    }
```

3. Ответьте на контрольные вопросы.
4. Составьте отчёт по практической работе.

3. Варианты заданий для самостоятельной работы

Во всех вариантах работа по получению и хранению данных должна быть выполнена с внешними источниками данных – файлами. Необходимо создать запись «студент» со следующими

полями: фамилия, имя, отчество, пол, факультет, курс, группа, отметки по пяти предметам, город проживания. Сформировать текстовый файл с данными указанной структуры, объёмом не менее 15-20 записей о студентах. Данные о студентах из файла читаются в массив записей. После выполнения задания, полученные результаты о студентах записываются из массива записей в выходной файл в виде таблицы или списка, где одна строка содержит все характеристики элемента «студент».

1) Напечатать в консоль список студентов, в котором студенты-задолжники расположены по алфавиту. Студентов-задолжников выделить.

2) Сформировать список студентов, в котором студентки-отличницы расположены по алфавиту. Студентов-отличниц выделить.

3) Сформировать список студентов, в котором студенты пятых курсов расположены в порядке убывания средних балов. Список вывести на консоль.

4) Сформировать список студентов, в котором студенты-задолжники по двум предметам расположены по алфавиту. Студентов-задолжников при выводе выделить.

5) Исключить из списка студентов каждого факультета, которые по всем предметам имеют двойки. Новый список выдать на консоль.

6) Сформировать список студентов, в котором неуспевающие студенты расположены в конце списка. Список выдать на консоль.

7) Сформировать список студентов по факультету БИО, в котором на первом месте перечисляются отличники. Студенты-отличники перечисляются в порядке убывания номеров курса. Внутри курса - в порядке убывания номеров групп. Список отличников распечатать.

8) На каждом курсе найти группы, в которых больше всего отличников. Сформировать список, в котором на первом месте перечислить студентов из этих групп. Группы перечислять в порядке убывания номеров курса. Список выдать на консоль.

9) Из общего списка студентов сформировать список по факультетам (порядок по факультетам произвольный). Внутри

факультета студентов группировать по возрастанию номеров курса. Список выдать на консоль.

10) Сформировать список студентов, в котором студентки, проживающие в городе Королёв, и имеющие все пятерки, расположены по алфавиту, а затем идут остальные обучающиеся

11) Добавить в список несколько студентов в диалоговом режиме. Новый список упорядочить по факультетам, на каждом факультете – по алфавиту. Новый список выдать.

12) Сформировать список студентов, в котором студенты расположены по алфавиту мест проживания. Список выдать на консоль.

13) По заданному факультету подсчитать средней бал каждого курса. Сформировать список, в котором курсы данного факультета располагаются по убыванию среднего бала. Список распечатать в консоли.

14) Сформировать список студентов, в котором студенты-отличники расположены в начале списка. Список выдать в консоль.

15) Сформировать список студентов, в котором неуспевающие студенты упорядочены по алфавиту, а затем идут остальные студенты по факультетам.

16) По заданному номеру курса подсчитать средней бал каждой группы. Сформировать список, в котором группы данного курса располагаются по убыванию среднего бала. Список выдать.

17) Выдать курс и фамилии девушек по алфавиту. Определить на каком курсе обучается больше всего девушек факультета ИО? Сформировать список студентов факультета ИО по курсам. Курсы располагать по возрастанию студенток в нем.

18) Найти самую распространенную фамилию юношей из списка студентов факультета ЭБ. Сформировать список студентов ЭБ, в котором на первом месте перечисляются группы с распространенной фамилией. Если таких групп несколько, то группы располагать по убыванию номеров групп. Студентов с самой распространенной фамилией при выводе пометить.

19) Сформировать список студентов, в котором студентки расположены по убыванию среднего балла. Список студентов выдать.

20) Определить самую распространенную фамилию юношей из списка студентов. Сформировать список студентов, в котором

фамилии юношей расположить в порядке уменьшения их частоты появления в списке. Фамилии юношей одинаково распространенных располагать по алфавиту. Студентов с самой распространенной фамилией выделить.

21) Сформировать список студентов по группам факультета ИБ. Список по группам формировать по возрастанию количества неуспевающих студентов. Список выдать в консоль. Неуспевающих студентов при выводе пометить.

22) Сформировать список студентов по курсам. Список по курсам формировать по убыванию процентов отличников среди юношей.

23) Выдать номер курса факультета УТС, на котором больший процент отличников юношей. Составить список студентов, в котором группы факультета УТС расположены по убыванию процента отличников в нем.

24) Определить самую распространенную фамилию юношей из списка студентов первого курса. Сформировать список студентов, в котором студенты найденного курса располагаются первыми. Если таких курсов несколько, то списки выдавать по убыванию номера курса

25) Определить самую распространенную фамилию девушки из списка студентов первого курса. Сформировать список, в котором на первом месте перечисляются факультеты с распространенной фамилией. Если факультетов несколько, то порядок по факультетам произвольный. Список напечатать в консоль. Студенток с самой распространенной фамилией при выводе пометить.

26) Выдать самую распространенную фамилию девушки из списка студентов факультета ЭБ. Сформировать список студентов ЭБ, в котором на первом месте перечисляются группы с распространенной фамилией. Если таких групп несколько, то группы располагать по убыванию номеров групп. Студенток с самой распространенной фамилией при выводе пометить.

27) Выдать курс и фамилии студенток по алфавиту, на котором обучается больше всего девушек. Сформировать список студентов, в котором курсы расположить по уменьшению количества девушек в нем.

4. Требования к оформлению отчёта

Отчёт по практической работе должен включать следующие разделы.

1. Титульный лист (Приложение А).
2. Цель работы и постановку каждого выполняемого задания.
3. Описание входных, выходных данных, модели решения задачи.
4. Скриншоты окон в среде программирования, подтверждающие работоспособность задач и примеров, выполненных в кодах изучаемых ЯП ВУ.
5. Выводы по работе.
6. Ответы на контрольные вопросы.
7. Список использованной литературы и Интернет-источников.

Контрольные вопросы

1. В каких случаях для описания данных в программе целесообразно использование типа «структура»?
2. Какое ключевое слово начинает определение структуры?
3. Как называются элементы структуры?
4. Как записать тип «структура» на C/C++ ?
5. Возможно ли в структуре задание элементов одного типа?
6. Какими образом можно обратиться к полям структуры?
7. Каким образом осуществляется доступ о к элементу массива структур?
8. Как обратиться к полям структуры через указатель на нее?
9. Опишите способы ввода/вывода элементов массива структур.
10. Какая разница между определениями структуры и массива?

Практическая работа №7 ФУНКЦИИ

ЦЕЛЬ РАБОТЫ: Закрепить практические навыки по использованию функций, как основной формы представления подпрограмм на С/С++. Освоить формирование функций с различным типом параметров, способы обращений к функциям и алгоритм выполнения программного кода, содержащего функции.

1. Основные понятия

Подпрограммой называется логически законченная часть программного кода, обладающая собственным именем и имеющая (не обязательно) список параметров, которые должны быть определены при вызове подпрограммы. Подпрограмма может быть объявлена как процедура или функция. Следовательно, *функция* – это подпрограмма, в отличие от процедуры она типизирована и возвращает значение такого типа, каким она объявлена.

Для языка С в общем виде структура функции выглядит как:

<тип_функции> <имя_функции> ([список_параметров]) {тело_функции},

для языка программирования С++ структура аналогична, только скорректирована под объективно-ориентированную направленность языка и его возможности:

[класс] <тип_функции> <имя_функции> ([список_параметров]) [throw (вид исключения)] {тело_функции}.

В других языках программирования высокого уровня для объявления функций и процедур имеются специальные зарезервированные слова, определяющие вид подпрограммы, такие как function, procedure [1].

Для функции, не возвращающей значение в рассматриваемых ЯП ВУ определён тип *void*. Значение, которое возвращается функцией, называется результатом. Функция может возвращать только одно значение: тип возвращаемого функцией значения может быть любым, кроме массива и функции (но может быть указателем на массив или функцию). Если функция возвращает значение, то в теле функции обязательно присутствует директива *return*. В список параметров функции могут быть включены параметры любых типов [8], например:

*void f1(int *mas, FILE *f, int n){…}*
*void sortMass (int *a, int n) {…}*
double f2(int a, float b){return 1}

Как правило, при построении кода на C/C++ , одну из функций называют именем *main* - она основная, с неё начинается выполнение алгоритма программы, а затем вызываются остальные функции. Таким образом, получается лаконично написанный главный модуль решения задачи на ЭВМ.

Вызываемые функции должны быть заранее объявлены в программе и представление кода должно быть видимо из того модуля, который их вызывает. Параметры в момент вызова функции должны быть однозначно определены, так как компилятор во время вызова функции проверяет доступность тех значений параметров и переменных, которые должны быть задействованы во время выполнения функции. Имена параметров при объявлении функции можно опускать.

В определении, в объявлении и при вызове одной и той же функции типы и порядок следования параметров должны совпадать [5]. Вызывать функцию можно в любом месте программы и сколько угодно раз, где требуется рассчитать или получить возвращаемое функцией значение, кроме того, можно использовать функцию в составе вычисляемых выражений в коде.

2. Методические указания

1. Подпрограмма выполняет преобразование входных параметров в выходные для получения определённого результата либо по изменению данных параметров, либо по получению с помощью них каких-то других требуемых решений. Поэтому, объявляемая функция может содержать набор различных параметров, называемых *формальными*, причём, их имена могут отличаться от имён, которыми они могут инициализироваться при вызове функции. В момент вызова функции параметры становятся *фактическими*.

2. Каждая функция является логически обособленным блоком кода – поэтому параметры, объявленные внутри функции, являются *локальными,* и их имена могут совпадать с именами переменных, объявленных в вызывающей программе или других

подпрограммах. Локальные переменные существуют ровно столько времени и сохраняются в сегменте стека столько (рис. 10, практическая работа 4) – сколько работает подпрограмма (функция или процедура), затем они уничтожаются. В качестве исключения констатируется ситуация, когда переменные являются глобальными или объявлены с модификатором *static*. В случае совпадения имен глобальных и локальных переменных в коде функции приоритет имеют локальные переменные,

3. Различают несколько способов передачи параметров в функцию на C/C++.

Первый, когда формальные параметры передаются в вызываемую функцию по значению. При этом, изменение их в ходе выполнения функции, никак не влияет на их значение в вызывающей функцию программе. Шаблон функции с данным типом параметров:
<тип_функции> <имя_функции>(<тип_перем.> <имя_перемен)

Выполните пример 7.1 для функции с данным типом параметров. Объясните, почему изменились (или не изменились) используемые параметры для функции f1 до вызова подпрограммы и после.

Пример 7.1 Передача параметров в функцию по значению

```
#include <iostream>
using namespace std;
int f1(int x, int y);
void main(){
	setlocale(LC_ALL, "rus");
	int x = 6; int y = 5;
	cout << " До вызова функции y = " << y << "\n";
	f1(x, y);
	cout << "После вызова функции y = " << y << "\n";
	cout << " Сумма = " << x+y << "\n";}
int f1(int x, int y)
{	int z;
	y = x; //y принимает значение x
	cout << "Параметр y меняется и равен = " << y << "\n";
	z = y + x;
	cout << "Расчёт z = " << z << "\n";
	return z;
```

}

Второй способ передачи параметров состоит в том, что в функцию передается не имя переменной-параметра, а значение адреса этой переменной-параметра. Данный способ передачи параметров, в общем случае, можно выразить заголовком:
<тип_функции><имя_функции> (<тип_перем.> *<имя_перемен>)

Пример 7.2 демонстрирует способ передачи параметров по адресу для переменной x и ещё раз объясняет механизм передачи параметров по значению (переменная m).

Пример 7.2 Передача параметров в функцию по адресу и значению

```
#include <iostream>
using namespace std;
void f1(int* x, int m);
void main(){
setlocale(LC_ALL,"RUS");
int y = 2; int m = 10;
cout << " Расчёт суммы: z=y+m\n";
cout << "Адрес переменной y = " << &y << "\n";
cout << "Значение до вызова функции y = " << y << "\n";
cout << "Значение до вызова функции m = " << m << "\n";
cout << "Значение  z = " << y + m << "\n\n";
f1(&y,m); // В функцию пересылаем адрес переменной y
cout << "Вышли из функции. \n";
cout << "Значение переменной y после вызова функции y = " <<
y << "\n";
cout << "Значение переменной m после вызова функции m = "
<< m << "\n";
cout << "Значение  z = " << y+m << "\n";}
void f1(int* x, int n) {
// x – переменная для адреса
// n – переменная для значения
cout << "Выполняется функция. \n";
cout << "В указатель x передан адрес переменной y = " << x << "\n";
// Выводится число из адреса, который содержится в x.
cout << "В этом адресе число = " << *x << "\n";
cout<<"Изменяем значения переданных параметров в функции\n";
```

```
*x = 11; //Число 11 засылается в адрес, который содержится в x
n = 0;        // Значение второго параметра обнуляется
cout << "Первая переменная в функции y = " << *x << "\n";
cout << "Вторая переменная в функции m = " << n << "\n";
cout << "Значение  z = " << *x + n << "\n\n";}
```

Третий способ передачи параметров - по ссылке. Общий формат заголовка функции в этом случае:

<тип_функции><имя_функции> (<тип_перем.> & <имя_перемен>)

Выполните пример 7.3 и объясните самостоятельно полученный результат по передаче параметров в функцию по ссылке. Обратите внимание на названия переменных в заголовке функции и теле программного кода функции.

Пример 7.3 Передача параметра в функцию по ссылке

```
#include <iostream>
using namespace std;
int f1(int& z);
void main()
{setlocale(LC_ALL, "rus");
        int y = 0;
        cout << "Главный модуль\n";
        cout << "Значение переменной  y = " << y << "\n\n";
        f1(y);
        cout << "Вышли из функции.\n";
        cout << "Значение переменной  y = " << y << "\n";
    }
int f1(int& x){
    cout << "Выполняется функция - получена ссылка на ячейку, где хранится переменная.\n";
    cout << "В функцию передано значение " << x << "\n";
    x = 100;
    cout << "Изменяем значение переменной " << x << "\n\n";
    return 0;
}
```

4. Необходимо обратить внимание и подробно изучить механизм передачи в подпрограмму элементов массива, как наиболее востребованного типа для представления однородных данных. Суть механизма заключается в передаче указателя на адрес

первого элемента массива, а затем получения доступа ко всем другим элементам через переданный адрес. Структура вызова функции с параметрами, передающими элементы массива может быть представлена как:
<тип_функции><имя_функции>(<тип_массива><имя_массива>);

причём, размер массива можно указать как явно, так и в виде указателя на первый элемент, например: void fun(float a[]), int fun(int *a), double fun(int b[100]). Главное, чтобы передаваемый массив был определён в главной функции(модуле) из которой вызываются функции с параметрами типа «массив».

Часто, в основном модуле объявляется тип-массив, а все действия с ним можно определить набором соответствующих функций, в том числе и ввод элементов из различных источников (консоль, файл), и вывод. Выполните пример 7.4, демонстрирующий работу функций по вводу, сортировке и выводу на консоль и в файл элементов массива.

Пример 7.4 Передача элементов массива в функцию

```
#include <iostream>
using namespace std;
void vvod(int *a, int m);
void sort(int *a, int m);
bool vyvod(int *a, int m, FILE* f);
int main(){
        setlocale(LC_ALL, "RUS");
                int *x, n;
                FILE* f1;
                char ss[7];
                //n - размерность исходного массива
cout << "Введите размерность массива"<< "\n(Ctrl + z Enter – выход из программы\n"; cin >> n;
                x = new int[n];
cout << "\n Введите имя файла\n "; cin >> ss;
                if ((f1 = fopen(ss, "a+")) == NULL)
                {
printf("Ошибка при открытии файла!!!\n"); exit;
//Если не удалось открыть файл - выдается сообщение об ошибке и выход из модуля.
```

```
            }
while (!feof(stdin))
// Возможно задание массивов разной длины
        //Цикл выполняется до тех пор,
            // пока не будет введен символ Ctrl + z.
        {
                vvod (x, n);
                sort (x, n);
                vyvod(x, n,f1);
        //**************************************
            delete [] x;
        cout << "\n Введите размерность массива(Ctrl + z Enter – выход из программы):\n ";
            cin >> n;
            x = new int[n];// Выделение памяти под новый массив
            }
        fclose(f1);
            return 0;}
void vvod(int *a, int m)
{cout << "\n Исходный массив\n";
        //выдача на консоль исходных данных
        for (int istr = 0; istr < m; istr++)
        {
            a[istr] = rand() % 20 - rand() % 20;
            printf("%d\t", a[istr]);
        }
}
        void sort(int *a, int m){
            int   imi, buff;
            cout << "\nВошли в сортировку\n";
        for (int i = 0; i < m; i++){
        imi = i;
            for (int j = i + 1; j < m; j++)
                if (a[imi] > a[j]) imi = j;
            buff = a[i];
        a[i] = a[imi];
            a[imi] = buff;
            }}
```

```
bool vyvod(int* a, int m, FILE* f)
{cout << "\n Выходной массив\n";
        for (int i = 0; i < m; i++) {
                printf("%d\t", a[i]);
                        fprintf(f, "%d\t", a[i]);
}
        fprintf(f, "\n");
cout << "Результаты записаны в файл " << "\n";
        return 1;}
```

Для двумерных массивов действуют те же правила при передаче параметров, интерфейсная строка функции, работающей с таким массивом может быть следующая:

void fun2(float a[][m]), int fun(int **a), double fun(int b[10][10]).

void f_1(int x[][m]), где m – ранее объявленная константа для размерности массива.

5. Передача имен функций в качестве параметров также возможна. Вызов различных функций может потребоваться при обработке каких-либо ситуаций при множественных реакциях какого-либо объекта на возникающее событие. Для этого можно создать указатель на функцию, которую необходимо передать как параметр в вызывающую её функцию или в основной модуль.

Например, имеется функция void f1(int a){} и функция void f2(int b){}.

Создаётся указатель на выше представленные функции, как: void (*pf)(int); - указатель на функцию pf = &f1; или pf = &f2; (возможно записать pf=f1 или pf=f2 - указателю присваивается адрес одной из функций или в содержимое ячеек, закреплённых за указателем pf, загружается содержимое переменных f1 или f2 – как адрес начала функции). Затем, можно обратиться к функциям f1 или f2 через функцию, в качестве параметра которой будут передаваться адреса одной из функций: pf(5); или pf(100); или (*pf)(5);

В целях понятности программного кода, часто в C++ используют переименование типов с помощью зарезервированного слова typedef, массивы указателей на функции: typedef void (*PF)(int) или PF ff[] ={&f1, &f2, &f3} Тип указателя при этом и типы вызываемых функций должны совпадать [5].

6. Существует ещё один тип функций – который называется *рекурсивные функции*. Функция, вызывающая саму себя – это и есть

рекурсивная функция(ещё её называют *прямой* рекурсией, в отличие от *косвенной*, когда две или несколько функций вызывают друг друга). Как известно, все параметры функции хранятся в сегменте стека во время работы функции, поэтому при рекурсии, все параметры предыдущих вызовов функции также должны храниться в стеке. Стек ограничен, поэтому при использовании рекурсивных функций возможно переполнение сегмента стека и останов работы программного модуля.

Алгоритм Евклида является наиболее древним рекурсивным методом отыскания наибольшего общего делителя (НОД) для двух натуральных чисел. Выполните пример 7.5, демонстрирующий этот алгоритм.

Пример 7.5 Рекурсивная функция нахождения НОД для двух натуральных чисел

```
#include <iostream>
using namespace std;
int gcd(int m, int n);
void main()
{
	setlocale(LC_ALL, "RUS");
	int y = 6; int x = 15;
	cout << "Вызывается функция НОД \n\n";
cout<<"Значение НОД для "<<y<<” и ”<<x<<” равно ”
<<gcd(x,y) << "\n";
}
	int gcd(int m, int n)
	{
		if (n == 0) return m;
		return gcd(n, m % n);
	}
```

Классическими примерами использования рекурсивных функций также являются: расчёт факториала n! и нахождение чисел ряда Фибоначчи. Найдите в рекомендованной литературе данные алгоритмы. Реализуйте их в среде программирования на C/C++. Опишите результат.

Рекурсивные функции чаще всего применяют для компактной записи решения задач, основанных на рекурсивных алгоритмах или

для реализации динамических структур, таких как деревья. Однако, необходимо помнить, что ввиду переполнения сегмента стека, при возрастающем числе вызовов функции, рекурсия опасна и требует значительного времени центрального процессора ЭВМ. Любую задачу, основанную на рекурсии – можно решить и без использования рекурсивных функций – прямых или косвенных.

7. Ответьте на контрольные вопросы.
8. Составьте отчёт по практической работе.

3. Варианты заданий для самостоятельной работы

Для всех вариантов: создать меню, использовав указатели на функции и предусмотреть вызов функций, у которых представлены параметры различного вида. Для этого используйте варианты уже выполненных практических заданий 3, 4, 5. Необходимо использовать передачу параметров в этих кодах алгоритмов решения задач по значению, адресу и ссылке.

4. Требования к оформлению отчёта

Отчёт по практической работе должен включать следующие разделы.

1. Титульный лист (Приложение А).
2. Цель работы и постановку каждого выполняемого задания.
3. Описание входных, выходных данных, модели решения задачи.
4. Скриншоты фрагментов консольных окон и внешних файлов(при необходимости), подтверждающие работоспособность представленных алгоритмов решения задач в среде программирования на изучаемых ЯП ВУ.
5. Выводы по работе.
6. Ответы на контрольные вопросы.
7. Список использованной литературы и Интернет-источников.

Контрольные вопросы

1 Укажите, при каких условиях целесообразно использовать функции?

2 Приведите различные примеры вызова функций в кодах ЯП ВУ C/C++.

3 Представьте общую схему объявления функции.

4 Приведите примеры возможных способов передачи параметров в функцию.

5 Укажите, как можно организовать передачу элементов одномерных массивов в функцию.

6 Приведите различные способы передачи двумерного и многомерного массива, в общем случае, в функцию.

7 В чём разница между глобальными и локальными переменными, используемыми в программном коде.

8 Какие параметры называют формальными, а какие – фактическими по отношению к подпрограмме.

9 Может ли функция возвращать пустое значение? Если да, то каким образом – поясните.

10 Какие значения переменных не может возвращать функция?

11 В какой момент осуществляется вход и выход из подпрограммы?

12 Если в программном коде есть выражения, то может функция быть частью выражения?

13 Как ведут себя локальные переменные, объявленные в функции: сохраняются ли их значения, когда вызов функции завершён?

14 Какую функцию называют рекурсивной. Приведите пример использования рекурсивной функции.

15 В чём состоят достоинства и недостатки рекурсивных функций?

16 Чем отличается прямая рекурсия от косвенной в отношении рекурсивных функций.

Практическая работа №8 Динамические структуры. СПИСКИ

ЦЕЛЬ РАБОТЫ: Исследование динамических структур данных на примере односвязных списков

1. Основные понятия

В классификации типов данных любого алгоритмического языка программирования выделяется ветвь типов для представления динамических структур данных. Традиционно, к данным такого типа относят стеки, очереди, списки, деревья.

Под такие типы данных память выделяется во время выполнения программы, но элемент данных и тип связи элементов должен быть определён во время компиляции. Таким образом, элемент данных динамических структур состоит из *информационной* и *указательной* части, благодаря чему элементы динамических структур могут располагаться в ОЗУ в произвольном порядке.

Причём, информационная часть (или поле) элементов таких структур может быть представлена как стандартными типами, так и структурированными, кроме файлового. Указательная часть (поле) определяет – как будут связаны элементы в структуре: линейно, нелинейно, кольцевым порядком или односвязным, двусвязным. По мере появления новых элементов в структуре во время выполнения программы, они подсоединяются к существующим элементам с помощью указательной части, а память под новые элементы выделяется отдельными блоками (которые представляют единую структуру благодаря указателям). Поэтому, и название таких типов отражает их смысл: «динамические структуры данных». Порядок операций над данными типами строго определён и зависит от типа связей между элементами.

Данные, представленные такими типами, эффективно используются в задачах сортировки, так как не требуют перестановки, а лишь требуют изменения указательного поля.

Описание простейшего элемента динамической структуры данных(иногда элементы называются узлами или компонентами структуры) можно представить следующим способом:

```
struct Init
{
//информационная часть динамической структуры
```

```
int n;
float a;
//указательная часть динамической структуры
Init *p;
};
```

В приведенном элементе поле данных представлено стандартными типами – целочисленным и вещественным, а указательная часть структуры – содержит адрес следующего такого же точно элемента структуры, значит, указатель имеет тип Init.

Одной из наиболее популярных динамических структур является список: линейная динамическая структура; список может быть *однонаправленным*, когда указатель содержит ссылку на следующий элемент; *двунаправленным*, когда указатель содержит две ссылки: на предыдущий и последующий элемент; *циклическим(кольцевым)*, когда последний элемент содержит ссылку на первый элемент (или наоборот, когда первый элемент содержит ссылку на последний).

Например, элемент двунаправленного линейного списка может быть в кодах рассматриваемого языка программирования представлен, как:

```
struct Init
{
int n;
float a;
//указательная часть списка:
Init  *next;
Init  *prev;
};
```

К списку применимо такое понятие, как «ключ». В качестве ключа могут выступать различные части поля данных: обычно, текстовые или целочисленные типы. Например, для списка из элементов «студент», в котором поле данных содержит ФИО, пол, город проживания, факультет, курс, группу, оценки по предметам, для разных целей работы со списком, ключом может быть как фамилия обучаемого, так и составной ключ «курс-группа».

Для динамической структуры «список» определены следующие операции:

— начальное формирование списка;

— добавление элемента в конец списка;

— чтение элемента с заданным ключом;

— вставка элемента в заданное место списка (до или после элемента с заданным ключом);

— удаление элемента с заданным ключом;

— упорядочивание списка по ключу.

Динамическая структура «очередь» является частным случаем структуры «список»: новые элементы поступают в конец списка, а выбор элементов осуществляется из начала списка.

Для работы с динамическими структурами в C++ есть специальные средства, последовательные контейнеры, к которым относятся: векторы (vector), двусторонние очереди (deque) и списки (list), а также так называемые адаптеры, то есть варианты, контейнеров — стеки (stack), очереди(queue) и очереди с приоритетами (priority queue).[5]

2. Методические указания

1. Согласно приведенным операциям со списками, перед выполнением индивидуального занятия по вариантам, выполните примеры по формированию списка в стиле ЯП С.

В примере 8.1 информационное поле представлено целочисленным типом, определяющим формирование списка из любых целых положительных и отрицательных чисел.

Пример 8.1 Начальное формирование списка

```
#include <iostream>
using namespace std;
void main()
{
	setlocale(LC_ALL, "RUS");
	struct node
	{
		int inf; // информационное поле.
		node *next; // указательная часть элемента списка
	};
	node *r, *rp = NULL, *fr = NULL, *er = NULL;
 /*	fr – указатель на начальный элемент списка
```

```
        er – указатель на последний элемент списка
        r – указатель для формирования нового узла списка
        rp – вспомогательный указатель */
        int a; // вспомогательная переменная
        FILE *f;
        if ((f = fopen("Text1.txt", "r")) == NULL)
        {printf("Ошибка при открытии файла!!!\n");
        goto m1;}
        else {// Начало формирования списка.
                do // Начало цикла ввода чисел из файла.
                {
                fscanf(f, "%d", &a); // Ввод числа из файла.
                r = new node;
r->inf = a; //Инициализация поля данных inf нового элемента списка.
r->next = NULL; //Инициализация указательного поля next нового элемента списка
if (fr == NULL) // Проверка списка
fr = r; // Первый элемент сформирован
        else  //Если список существует, то подсоединение
                //следующего элемента
                er->next = r; // новый элемент списка.
                        er = r; // новый элемент пока последний
                } while (!feof(f)); //
                fclose(f);}// Список сформирован.
        cout << "Новый список:\n";// Вывод списка в консоль.
        r = fr;
        while (r != NULL){
                cout << r->inf << "\n";
        r = r->next; // переход к следующему элементу списка
        }
m1:
}
```

2. Далее расширьте код и добавьте в него подпрограмму удаления элемента списка по заданному ключу (Пример 8.2).

Пример 8.2 Фрагмент кода для удаления элемента из списка

```
{
```

```
cout << "\n Удалить элемент со значением = ";
cin >> a;
r = fr;
if (r->inf == a)
{
        cout << "\n удаляется первый элемент списка\n";
        fr = fr->next;
        delete r;
}
else
{
while ((r->inf != a) && (r != NULL)) // поиск элемента для удаления
        {       rp = r; // адрес пройденного элемента.
                r = r->next;
        } // переход на новый элемент.
        if (r->inf = = a) // проверка искомого элемента
        {
cout << " Удаляется элемент со значением= " << r->inf << "\n\n";
                rp->next = r->next;
                delete r;
}
```

3. Дополните код второй подпрограммой: вставка нового элемента в список (Пример 8.3).

Пример 8.3 Фрагмент кода для вставки нового элемента в список

```
{//Вставка нового элемента в список
int priz;
cout << "\nВведите индикатор вставки элемента:"
                << "\n1 - первый элемент"
                << "\n2 - любой, кроме первого"
                << "\nCtrl + z Enter – выход из п/программы\n";
cin >> priz;
while (!feof(stdin))//Цикл выполняется до ввода Ctrl + z
        {
                switch (priz)
                {
case 1:
```

```
 cout << "Программа вставки первого элемента списка \n";
        cout << "\n\nВведите значение для нового элемента n = ";
                cin >> a;
                rp = new node;
                        rp->inf = a;
                        rp->next = fr; // Новый элемент делаем первым.
fr = rp; // В указатель пересылаем адрес нового элемента.
                cout << "\n Список с новым головным элементом:\n";
                        r = fr;
                        while (r != NULL)
                        {
        cout << r->inf << " ";
                r = r->next;
                        }
        break;
case 2:
        int b;
cout << "\nВставить в список элемент со значением b= ";
        cin >> b;
        cout << "\nЗа элементом со значением n = ";
                cin >> a;
                r = fr;
        while ((r->inf != a) && (r->next != NULL))
        {
        r = r->next;
        }
        if (r->inf == a)
        {
        // Формирование нового элемента
        rp = new node;
                rp->inf = b;
                rp->next = r->next;
        r->next = rp; // присоединение нового элемента списка
cout << "Список после вставки элемента \n";
                        r = fr;
                        while (r != NULL)
                {
                cout << r->inf << " ";
```

```
                r = r->next;
            }
        }
        else
cout << "\nЭлемент со значением " << a << " не найден ";
                break;
        default:
cout << "\nПризнак вставки элемента неверен" << priz << "\n ";
            break;
                }
//**********************************
        cout << "\nВведите индикатор вставки элемента:"
            << "\n1 - первый элемент"
            << "\n2 - любой, кроме первого"
            << "\n(Ctrl + z Enter – выход из п/программы\n";
            cin >> priz;
}
}
```

Объясните, чем отличается вставка первого элемента списка от других элементов?

4. Ответьте на контрольные вопросы.
5. Составьте отчёт по практической работе.

3. Варианты заданий для самостоятельной работы

Задание для всех вариантов выполнить, сначала используя односвязный список, затем двусвязный.

1) Построить список из входной последовательности целых чисел. Удалить из него все отрицательные числа.
2) Построить список из входной последовательности целых чисел. Удалить из него все положительные числа.
3) Построить список из входной последовательности целых чисел. Сформировать новый список в порядке: отрицательные, нулевые, положительные числа.
4) Построить список из входной последовательности целых чисел. Сформировать новый список в порядке: положительные, нулевые, отрицательные числа.

5) Построить список из входной последовательности целых чисел. Сформировать новый список в порядке: нулевые отрицательные, положительные числа.

6) Построить список из входной последовательности целых чисел. Сформировать новый список из построенного, в котором не будет чётных чисел.

7) Построить список из входной последовательности целых чисел. Сформировать новый список из построенного, путём удаления из него всех нечётных чисел.

8) Построить линейный список из входной последовательности целых чисел, затем сформировать новый список из построенного, в котором установлен порядок: нечётные числа, нулевые, затем чётные.

9) Построить линейный список из входной последовательности целых чисел, затем сформировать новый список из построенного, в котором установлен порядок: чётные числа, нулевые, затем нечётные.

10) Построить линейный список из входной последовательности целых чисел, сформировать новый список из построенного, в котором удалить все числа, кроме простых.

11) Построить линейный список из входной последовательности целых чисел, сформировать новый список из построенного, в котором удалить все простые числа.

12) Построить линейный список из входной последовательности целых чисел, сформировать новый список из построенного, в котором поменять местами первое и последнее число.

13) Построить линейный список из входной последовательности целых чисел, сформировать новый список из построенного, в котором соседние числа попарно меняются местами.

14) Построить линейный список из входной последовательности целых чисел, сформировать новый список из построенного, в котором поменять местами первое и максимальное число, последнее и минимальное.

15) Построить линейный список из входной последовательности целых чисел; сформировать новый список из построенного, в котором удалить все числа, кратные 3.

16) Построить линейный список из входной последовательности целых чисел, сформировать новый список из построенного, в котором удалить все числа, кратные 5.

17) Построить линейный список из входной последовательности целых чисел, затем сформировать новый список из построенного, в котором нет нулевых элементов.

18) Построить линейный список из входной последовательности целых чисел, затем сформировать новый список из построенного, в котором удалить все одинаковые элементы.

Для вариантов 19–25 использовать в качестве источника исходных данных текстовый файл, в котором записано 10-15 фамилий в алфавитном порядке.

19) Построить линейный список из текстовой информации из файла; затем получить новый список, удалив из построенного списка фамилии, начинающиеся на букву «К».

20) Построить линейный список из текстовой информации из файла; затем получить новый список, в котором останутся только фамилии, начинающиеся на букву «О».

21) Построить линейный список из текстовой информации из файла; затем получить новый список, удалив из построенного списка самую длинную фамилию.

22) Построить линейный список из текстовой информации из файла; затем получить новый список, удалить из него самую короткую фамилию.

23) Построить линейный список из текстовой информации из файла; затем получить новый список, вставить в него введенную с консоли фамилию, сохранив алфавитный порядок.

24) Построить линейный список из текстовой информации из файла; затем получить новый список: найти в нём требуемую фамилию (допустить несколько одинаковых фамилий) и удалить её из списка.

25) Проверить на правильность сформированный список из текстовой информации из файла: фамилии должны состоять только из букв – прописных(начальная буква) и строчных. Неверно записанные элементы удалить из списка, получив при этом новый список без ошибок.

4. Требования к оформлению отчёта

Отчёт по практической работе должен включать следующие разделы.

1. Титульный лист (Приложение А).
2. Цель работы и постановку каждого выполняемого задания.
3. Описание входных, выходных, вспомогательных данных, модель решения задачи.
4. Скриншоты фрагментов консольных окон и внешних файлов(при необходимости), подтверждающие работоспособность представленных алгоритмов решения задач в среде программирования на изучаемых ЯП ВУ.
5. Выводы по работе.
6. Ответы на контрольные вопросы.
7. Список использованной литературы и Интернет-источников.

Контрольные вопросы

1. Как будет выглядеть в общем виде элемент линейного однонаправленного списка из чисел.
2. Как определить элемент линейного двунаправленного списка из слов.
3. Дан односвязный линейный список. Привести фрагмент кода выдачи данного списка на консоль.
4. Опишите механизм удаления из списка элемента, заданного ключом.
5. Опишите механизм вставки в список заданного элемента.
6. Поясните запись кода Init * fst = NULL; Init *en = NULL;
7. Определены переменные Init * fst = NULL, *en, *p. Какое действие будет выполнено по данному коду: p -> a.
8. Необходимо удалить из линейного односвязного списка первый элемент. Какие действия необходимо выполнить?
9. Написать фрагмент кода для вставки в линейный односвязный список первого элемента.
10. Задайте, в общем виде, элемент динамической структуры типа «Стек».
11. Какие операции предусмотрены с элементами динамической структуры типа «Стек».

12. Задайте, в общем виде, элемент динамической структуры типа «Очередь».

13. Какие операции предусмотрены с элементами динамической структуры типа «Очередь».

14. Какие типы данных относятся к динамическим структурам?

15. Какие виды полей содержит элемент любой динамической структуры.

ЛИТЕРАТУРА

1. Голицына О.Л., Попов И. И. Программирование на языках высокого уровня: учебное пособие. – М: ФОРУМ, 2011. -496 с.: ил.
2. Исаева Г.Н., Куриков Д.В. Новые возможности перспективных языков программирования// Эволюционные процессы информационных технологий: сборник трудов по материалам 5-й всеросс. научно-технич. конф. 5 апреля 2020/ коллек. автор под общей науч. редак. В.М. Артюшенко, В.И. Воловач. -М.- «Научный консультант», 2020 – С.63-70.
3. Исаева Г.Н., Пахомов Д.А Возможности современных языков программирования высокого уровня// Современные информационные технологии. Сборник трудов под науч. ред. док. техн. наук, проф. В.М. Артюшенко. – М.: «Научный консультант», 2015. – С.167-175
4. Новиков Е. А. Языки программирования. Язык С. Версия 1.0 [Электронный ресурс] : конспект лекций / Е. А. Новиков, Ю. А. Шитов. – Электрон. дан. (3 Мб). – Красноярск : ИПК СФУ, 2008 – 400с.
5. Павловская, Т.А. С/С++. Программирование на языке высокого уровня / Т.А. Павловская // СПб.: Питер, 2009. – 461 с.
6. Свердлов С.З. Языки программирования и методы трансляции. Учебное пособие. -СПб.: Питер, 2007, - 638с.
7. Симонович С.В. Информатика: Базовый курс: учеб./ С.В. Симонович -Стандарт третьего поколения Изд. 3. -СПб., Питер, 2022. -640 с.
8. Шилдт Г. С++. Базовый курс / Шилдт Герберт, -Диалектика-Вильямс, - 2018, - 624с.
9. Электронный ресурс, код доступа: https://www.intuit.ru/https: (Дата обращения 9.07.2023)
10. Электронный ресурс, код доступа: https://learn.microsoft.com/ru-ru/cpp/overview/visual-cpp-in-visual-studio?view=msvc-170: (Дата обращения 9.07.2023)
11. Электронный ресурс, код доступа: https://visualstudio.microsoft.com/ru/ (Дата обращения 9.07.2023)

ПРИЛОЖЕНИЕ А

Федеральное государственное бюджетное образовательное учреждение высшего образования
«ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ ИМЕНИ ДВАЖДЫ ГЕРОЯ СОВЕТСКОГО СОЮЗА, ЛЕТЧИКА-КОСМОНАВТА А.А. ЛЕОНОВА»

ФАКУЛЬТЕТ ИНФОКОММУНИКАЦИОННЫХ СИСТЕМ И ТЕХНОЛОГИЙ

КАФЕДРА ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И УПРАВЛЯЮЩИХ СИСТЕМ

Отчёт по практической работе №_

по дисциплине «Языки программирования»

Выполнил: *студент гр.* _____

Преподаватель: _____

Королев, 202__г.

Галина Николаевна Исаева
Надежда Вадимовна Логачёва
Юрий Вениаминович Стреналюк

«ЯЗЫКИ ПРОГРАММИРОВАНИЯ»
Практикум по курсу «Языки программирования»

Учебное пособие