МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ
«БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

КАФЕДРА ИНФОРМАТИКИ И ПРИКЛАДНОЙ МАТЕМАТИКИ

МЕТОДИЧЕСКИЕ МАТЕРИАЛЫ К ЛЕКЦИЯМ ПО ДИСЦИПЛИНЕ

Информатика и компьютерная графика

для студентов специальности
«Автоматизация технологических процессов и производств»

Программирование на С с использованием открытых инструментальных средств

Издание дополненное и переработанное

БРЕСТ 2017

УДК 657.22(075)
ББК 65.052я73
М 92

В методических материалах приводятся необходимые теоретические сведения по языку программирования С и используемой инструментальной среде, которые применяются при разработке проекта в рамках лабораторных работ. Также дано описание проекта на уровне этапа предварительного проектирования системы. В рамках реализации проекта отрабатываются типовые элементы программирования на примере создания меню, сопровождения картотек и формирования печатных форм.

Методические материалы предназначены для использования на лекциях и в ходе выполнения лабораторных работ по дисциплине «Информатика и компьютерная графика» студентами специальности «Автоматизация технологических процессов и производств».

Составители: С.В. Мухов, доцент, к.т.н.,
Г.Л. Муравьев, доцент, к.т.н.
С.И. Парфомук, к.т.н.

ОГЛАВЛЕНИЕ

Глава 1. Основные понятия и терминология компьютерных технологий

— Как корабль назовешь, так он и поплывет.
Капитан Врунгель о терминологии.

Прежде чем приступить к реализации в рамках лабораторных работ проекта «Учет движения специальных жидкостей», который базируется на минимальной типовой модели обработки данных, попробуем навести порядок в компьютерной терминологии с учетом специфики прикладного пользователя. Использовать в учебных целях вышеуказанную модель мы будем потому, что, во-первых, в рамках данной модели прекрасно отрабатываются все типовые элементы обработки данных, и, во-вторых, вполне возможно, что в будущем наши студенты на производстве столкнутся именно с такими работами. Детализировать проект до мельчайших деталей мы не будем, но такие функции, как отработка сопровождения файлов, ввод и преобразование данных, формирование отчетов и их вывод на печать, создание меню, будут полностью отработаны в рамках нашего проекта.

КАРТОТЕКА есть совокупность объектов учета одного типа. Синонимы — справочник, файл, таблица. Как правило, термин «справочник» используется в случае использования картотеки для задания реквизита в первичном документе его выборкой из этой картотеки.

РЕКВИЗИТ есть характеристика объекта. Синоним — поле. Ключевой реквизит (**КЛЮЧ**) — это реквизит (или реквизиты), однозначно определяющий объект (карточку в картотеке). Если используется несколько реквизитов, то говорят о **СЛОЖНОМ КЛЮЧЕ**.

КАРТОЧКА ОБЪЕКТА есть совокупность реквизитов, описывающих объект. Как правило, карточка ассоциируется с отображением реквизитов на экране монитора в виде экранной формы. Синоним — запись. **ЭКРАННАЯ ФОРМА** есть отображение на экране монитора реквизитов карточки и специальных полей для вызова функций обработки данных.

При формировании отчетности в компьютерных технологиях используются следующие понятия: выходная форма, отчет, шаблон выходной формы, запрос на формирование выходной формы, уровень отчета.

ВЫХОДНАЯ ФОРМА — сформированный программным способом отчетный документ, предназначенный для вывода в виде твердой копии. Возможен ее предварительный просмотр на экране монитора. Синонимы — **ОТЧЕТ**, табуляграмма, машинограмма, форма, **ПЕЧАТНАЯ ФОРМА**, документ.

ШАБЛОН ВЫХОДНОЙ ФОРМЫ — специальное описание выходной формы с указаниями, как размещать поля и какие уровни отчета используются при формировании выходной формы. Синонимы — отчет, описание отчета, **ОПИСАНИЕ ВЫХОДНОЙ ФОРМЫ**.

ЗАПРОС НА ФОРМИРОВАНИЕ ВЫХОДНОЙ ФОРМЫ — специальное описание формирования выходной формы, содержащее указание способа формирования промежуточного набора данных, шаблон выходной формы и место, куда выводить форму. Синоним — **ЗАПРОС**.

УРОВЕНЬ ОТЧЕТА — уровни иерархического расположения данных, которые «отлавливаются» программой формирования выходной формы, а именно, при изменении указанных реквизитов счетчик выводится на печать и сбрасывается в ноль для выполнения суммирования на следующем уровне отчета. Как правило, уровень отчета определен в описании формы, а иерархия определяется указанием сортировки по этим реквизитам в запросе на формирование формы или описании отчета.

В отчетности, как правило, выделяют списковые и итоговые отчетные формы. В качестве примера списковой формы из проекта, который будет создан в рамках лабораторных работ, можно привести «Оборот специальных жидкостей по объекту». «Сводная ведомость прихода по объекту» и «Сводная ведомость расхода по объекту» представляют собой итоговые формы.

Для понимания того, что и откуда берется в системе, используется графическое представление учетной модели, которое будем называть **ФУНКЦИОНАЛЬНОЙ СХЕМОЙ ОБРАБОТКИ ДАННЫХ**. Пример схемы обработки данных нашего проекта приводится на рисунке 1.1. Цифры в угловых скобках в реальной схеме обработки данных отсутствуют, здесь же они указывают типовые работы по обработке данных, о которых говорится в главе 2.

Рисунок 1.1 — Схема обработки данных системы «Учет движения специальных жидкостей»

Для **СХЕМАТИЧЕСКОГО ИЗОБРАЖЕНИЯ КАРТОТЕКИ** на схеме обработки данных применяются прямоугольники, в которых выделены поля для указания НАЗВАНИЯ КАРТОТЕКИ, КЛЮЧА и СВЯЗУЮЩИХ РЕКВИЗИТОВ. **СВЯЗУЮЩИЕ РЕКВИЗИТЫ** указывают, что для объекта, описанного в конкретной карточке, по связующему реквизиту объекта будет выбрана информация из карточки другой картотеки. Обратим внимание на то, что в схеме обработки данных указаны только ключевые реквизиты и поля, используемые для связи картотек.

Необходимость использования схемы обработки данных вытекает из того факта, что для реквизитов, выбранных из других картотек, сотрудник должен отрабатывать корректировку не на текущей картотеке, а на картотеках, где эти реквизиты содержатся (стрелки «реквизит»— «ключ картотеки»). Схема обработки данных также помогает представить порядок формирования картотек (стрелки «название картотеки»— «название картотеки») и отчетов (стрелки «название картотеки»— «название группы отчетов»). Для **СХЕМАТИЧЕСКОГО ИЗОБРАЖЕНИЯ ПЕЧАТНЫХ ФОРМ** применяются прямоугольники, в которых выделены поля для указания НАЗВАНИЯ ГРУППЫ ОТЧЕТОВ и ПЕРЕЧНЯ ОТЧЕТНЫХ ФОРМ, например, «Оборот по объекту», «Сводная приход по объекту» и «Сводная расход по объекту».

На вышеуказанной схеме обработки данных можно видеть, что:

— на основании картотеки первичных документов формируется Регистрационный журнал (разноска в рег. журнал);

— на основании Регистрационного журнала методом двойной записи формируется Книга счетов;

— на основании Книги счетов формируются отчетные формы;

— при заполнении полей «номер» и «дата» документа в первичном документе используется картотека «настройка системы»;

— заполнение полей «откуда» и «куда» в первичном документе выполняется с помощью выборки из справочника «Объекты учета».

ОПИСАНИЕ КАРТОТЕКИ (карточки, структуры записи, объекта) есть определение реквизитов, определяющих объект, с помощью табличного списка этих реквизитов с указанием смысла и специфики реквизита, его обозначения в программах, типа и значности (размера) реквизита. Как правило, для описания объекта картотеки достаточно просмотра конкретной экранной формы (карточки объекта).

Тип реквизита может быть символьным (C — character), цифровым (N — numeric), содержащим дату (D — date) и т.д. Значность для символьного поля задает длину поля, для цифрового поля — количество цифр (дополнительно может быть задано количество цифр после запятой для дробного значения). Не путать со значимостью, означающей важность свойства. Пример описания картотеки приводится в таблице 1.1.

Таблица 1.1 – Картотека ОБЪЕКТЫ УЧЕТА

Реквизит	Обозначение	Тип и значность
Поле связи,используется программистом, = 0	fioa_o_0	C1
Код объекта – ключ	fioa_o_k	C3
Название объекта	fioa_o_n	C10
Тип объекта о-организация м-мастер с-станок	fioa_o_typ	C1
Норма списания на станок	fioa_o_nrm	N5

ГРУППА РАБОТ есть совокупность технологически связанных между собой элементарных выполняемых пользователем процедур (работ). В компьютерной системе, как правило, группа работ связывается с элементами экранного меню высшего уровня. Пример **ОПИСАНИЯ РАБОТ** в виде таблицы для нашего проекта смотри ниже (таблица 1.2).

Автоматизированное рабочее место (АРМ) называется ФУНКЦИОНАЛЬНО ПОЛНЫМ, если с его помощью на данном участке учета обеспечивается выполнение всех работ, производимых ранее вручную.

Таблица 1.2 — Описание работ системы «Учет движения специальных жидкостей»

Группа работ	Работы
Формирование первичных документов ДОКУМЕНТЫ	— Ввод текущей даты — Ввод и разноска в журналы первичных документов
Ведение рег.журнала и книги счетов РЖ	— Просмотр Регистрационного журнала (РЖ) — Формирование книги счетов из регистрационного журнала — Просмотр книги счетов (КС)
Формирование отчетности ОТЧЕТЫ	— Определение отчетных форм — Оборотная ведомость по объекту — Сводная ведомость по приходу — Сводная ведомость по расходу
Обслуживание картотек КАРТОТЕКИ	— Объекты учета — Настройка АРМ'а
Ведение архивов АРХИВ	— Копирование подсистемы — Восстановление подсистемы
Выход из подсистемы ВЫХОД	— Выход из подсистемы

Для вызова процедур АРМ'а используется экранное меню.

При проектировании «хорошего» меню необходимо соблюдение следующих принципов:

— выполняемые работы систематизируются по группам работ, а именно: выделяются группы работ как единые методологически объединенные последовательности элементарных работ, например, группа работ «Формирование отчетности (отчетность)», и эти группы работ выносят на верхний уровень меню;

— структура сложившейся ручной технологии учетного процесса должна максимально соответствовать компьютерной технологии обработки данных с точностью до «спрятанных» разработчиком шагов компьютерного расчета.

При указании работы для ее корректного определения необходимо указывать элементы экранного меню верхнего уровня, например, указание «выполнить ежемесячное формирование оборотной ведомости» может выглядеть таким образом — «Отчеты/Оборотная ведомость по объекту».

Одной из наиболее значимых и часто используемых типовых работ (процедур) является обслуживание картотек.

При работе с картотекой возможны следующие режимы работы:

— работа с конкретной карточкой (редактирование карточки);

— просмотр картотеки (или выборка карточки для обработки).

При работе с экранной формой могут быть использованы поля следующих типов:

— текст;

— отображаемое (неизменяемое) поле;

— редактируемое (изменяемое) поле;

— кнопка функционального вызова.

При этом могут использоваться поля специального вида, например, поле для обеспечения выборки из справочника или списка значений, переключатели и т.д., но такие поля сводятся к вышеуказанным базовым типам.

Желательно, а в рамках лабораторных работ и обязательно, выделять ключевые поля (ключ — поле, однозначно определяющее карточку) и простые данные различной обводкой.

Типовые функциональные кнопки:

— кнопки позиционирования (выбрать, назад, вперед);

— добавить (возможны варианты — дублировать, добавить, вставить);

— удалить (возможны варианты — удалить, пометить к удалению, обход);

— сортировка (возможны варианты по виду сортировки);

— найти с использованием фильтра;

— выход.

Обслуживание (сопровождение) картотеки включает в себя следующие работы, вызывающие изменение картотеки:

— создание новой карточки (добавить в конец картотеки или вставить в текущей позиции новую карточку);

— логическое удаление карточки (пометка карточки «удалить» для последующего физического удаления карточки);

— сортировка картотеки с физическим удалением карточек, ранее помеченных «удалить».

Просмотр картотеки есть отображение картотеки на экране монитора в виде сканируемой матрицы (browse, grid). Назначение режима просмотра картотеки состоит в обеспечении выборки карточки для дальнейшей ее обработки.

Как правило, эти многие работы обеспечиваются в виде функционального вызова в режиме «работа с карточкой». Для этого непосредственно на карточке (точнее, ее экранной форме), размещаются поля, отработка по которым приводит к выполнению функционального вызова (так называемые кнопки или поля функционального вызова).

Для отработки функционального вызова для некоторого поля ввода, как правило, используется двойное нажатие кнопки мышки, одно нажатие — установка поля, двойное нажатие — вызов функции.

При проектировании «правильной» экранной формы необходимо соблюдение следующих принципов:

— экранная форма типизирована как по геометрии экрана, так и по используемым функциональным вызовам;

— структура экрана отвечает сложившейся форме первичного документа или картотеки.

СОПРОВОЖДЕНИЕ ПРОГРАММНЫХ СРЕДСТВ есть модификация программного обеспечения исходя из изменения потребностей пользователей (расширение или модификация функций) или при обнаружении ошибок в его функционировании. **ОСНОВНЫМ ТРЕБОВАНИЕМ К ПРОГРАММНОМУ ОБЕСПЕЧЕНИЮ КОМПЬЮТЕРНЫХ СИСТЕМ С ИЗМЕНЯЕМОЙ ПРИКЛАДНОЙ ОБЛАСТЬЮ ЯВЛЯЕТСЯ НАЛИЧИЕ ГАРАНТИЙ СОПРОВОЖДЕНИЯ.**

ЭКСПЛУАТАЦИЯ ПРОГРАММНЫХ СРЕДСТВ есть ввод данных и запуск программ (в том числе и процедур восстановления системы) согласно соответствующим инструкциям.

Глава 2. Описание прикладной системы

— **Наилучший способ изучения** информационных **наук есть решение практических задач** по созданию систем обработки данных, **а не запоминание правил** и операторов.

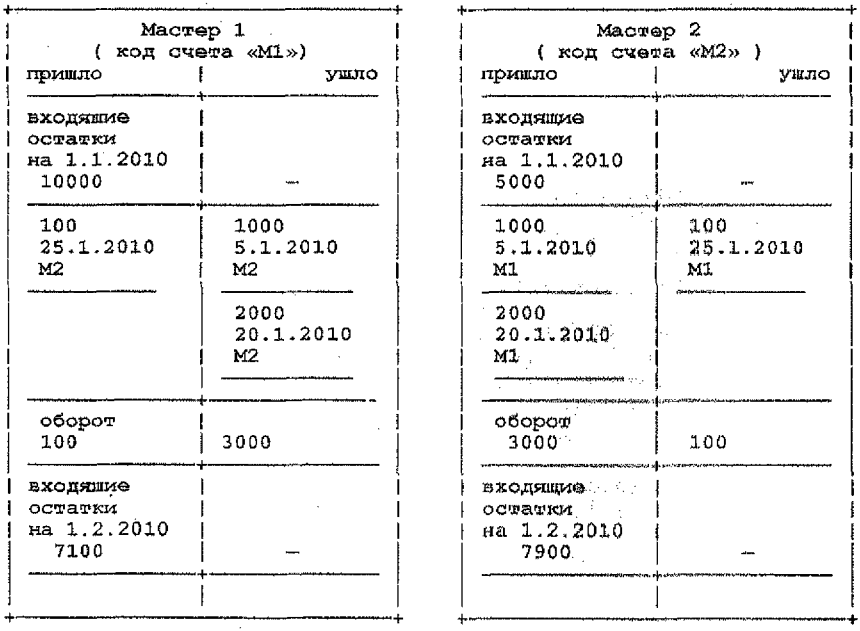
И. Ньютон об изучении информатики.

В качестве примера системы для обработки данных будет использоваться модель упрощенной учетной системы. Как уже говорилось выше, в рамках данной модели отрабатываются все основные процедуры обработки данных, а именно: создание и сопровождение картотек, использование справочников, разноска данных, программное формирование картотек, формирование печатных форм. В данной главе предлагается упрощенная трактовка учетных процедур на примере операций передачи специальной жидкости между объектами учета (организации, мастера, станки), а именно: по каждому субъекту учета ведется **СЧЕТ**, который представляет собой просто записи по приходу (дебет) и расходу (кредит) с легко отрабатываемой процедурой вычисления остатков по счету.

Одним из примеров конкретной реализации счета являются поименованные листки на каждый объект учета. Название листка будем считать кодом счета. Каждый листик разделен вертикальной линией на две части: левая половинка — приход по объекту учета или дебет счета (лат.debet — он должен), правая половинка — расход по объекту учета или кредит счета (лат.credit — он верит — т.е. дает деньги на веру). Нетрудно заметить, что такой листок есть не что иное, как широко известный бухгалтерский «самолетик». Все листки (счета) в совокупности называются **КНИГОЙ СЧЕТОВ** (смотри рисунок 2.1). На основании Книги счетов легко получается самая простая отчетная форма под названием «Остатки по объекту на указанную дату».

В нашей системе (смотри схему обработки данных на рисунке 1.1) операция записывается сначала в картотеку первичных документов, причем при ее оформлении используется **<1>** выход на картотеку «Объекты учета» с использованием фильтра по типу операции для выборки реквизитов «откуда» и «куда» с последующим занесением информации из выбранной записи «объект учета» в запись «первичный документ». Также при создании первичного документа **<2>** используется картотека «настройка системы» для заполнения полей «номер документа» и «дата документа». Завершающим аккордом обработки первичного документа является **<3>** формирование и занесение записи «движение» в Регистрационный журнал на основании текущей записи «первичный документ» (разноска первичного документа). Далее выполняется **<4>** программное формирование всех записей Книги счетов из всех записей Регистрационного журнала с выборкой по фильтру «интервал + счет», который хранится в картотеке «настройка системы».

Впоследствии из Книги счетов с помощью классических операций выборка, сортировка и формирование печатной формы по шаблону формируются <5> списковые и <6> итоговые печатные формы. Для сопровождения картотек создаются <7> классические экранные формы, <8> просмотровые экранные формы и <9> экранные формы для ввода настроек системы. Все вышеуказанные работы (вызов экранных форм <7>,<8>,<9>; вызов программного модуля <4>; формирование печатных форм <5>,<6>) вызываются с помощью меню, для описания которого служит «Описание работ» (смотри таблицу 1.2). Реализация проекта (комплект программ в рамках одного программного вызова) и его запуск на выполнение – это просто завершающий технический штрих в нашей работе.



Мастер 1 (код счета «М1»)	
пришло	ушло
входящие остатки на 1.1.2010 10000	–
100 25.1.2010 М2	1000 5.1.2010 М2
	2000 20.1.2010 М2
оборот 100	3000
входящие остатки на 1.2.2010 7100	–

Мастер 2 (код счета «М2»)	
пришло	ушло
входящие остатки на 1.1.2010 5000	–
1000 5.1.2010 М1	100 25.1.2010 М1
2000 20.1.2010 М1	
оборот 3000	100
входящие остатки на 1.2.2010 7900	–

Рисунок 2.1 — Пример самой маленькой в мире Книги счетов

Из опыта разработки и сопровождения компьютерных систем можно сказать, что **ОПИСАНИЕ СИСТЕМЫ, которое достаточно и необходимо для ее однозначного определения и может использоваться в качестве «Руководства пользователя», должно содержать:**

— функциональную схему обработки данных;

— описание картотек;

— описание выполняемых работ, которые сгруппированы технологически.

В рамках лабораторных работ необходимо разработать упрощенную систему учета. Схема обработки данных для нашего проекта приведена на рисунке 1.1. Описание картотек самостоятельно оформляются студентами в формате, приведенном в таблице 1.1, на основании изучения аналога нашей системы, который размещен на сервере университета. Описание работ приведено в таблице 1.2.

Внешний вид экранных форм и меню проектируется по аналогии с формами системы на сервере. Печатные формы, формируемые учебной системой, приведены на рисунке 2.2. Скриншоты экранных форм и меню, а также образцы формируемых печатных форм и первичных документов после разработки системы могут прилагаться в качестве приложения к Инструкциям по эксплуатации (и, соответственно, в наши отчеты по лабораторным работам и курсовые проекты).

Регистрационный журнал

Дата	Документ	Операция	Приход	Расход	Количество
05.01.12	НВП/123 от 05.01.12	Передача спирта	М2	М1	1000

Книга счетов

Дата	Документ	Операция	Объект	Корр. объект	Приход	Расход
05.01.12	НВП/123 от 05.01.12	Передача спирта	М2	М1	1000	0
05.01.12	НВП/123 от 05.01.12	Передача спирта	М1	М2	0	1000

Оборотная ведомость
по объекту М2 за период с 1.01.2013 до 1.02.2013

Дата	Документ	Операция	Корр. объект	Входящий остаток !!не обр	Обороты Приход	Обороты Расход	Исходящий остаток !!не обр
05.01	НВП/123	перед.спирта	М1		1000	0	
Итого по объекту М2					1000	0	

Сводная ведомость по расходу
по объекту М1 за период с 1.01.2013 до 1.02.2013

Расход на	Количество
М2	1000
Итого по объекту М1	1000

Сводная ведомость по приходу
по объекту М2 за период с 1.01.2013 до 1.02.2013

Приход от	Количество
М1	1000
Итого по объекту М2	1000

Рисунок 2.2 – Печатные формы, формируемые учебной системой

Выходные формы должны быть ориентированы на типизацию, а также учитывать специфику использования вычислительной техники, а именно: а) используются отчетные формы по ОДНОМУ счету с их формированием по классической методике — выборка, сортировка, генератор отчетов; б) переход к вертикальному размещению граф с целью обеспечения ФИКСИРОВАННОГО количества граф в строке. Касательно печатных форм Регистрационный журнал и Книга счетов можно отметить, что они: а) являются классическими списковыми формами; б) необходимы, прежде всего, ДЛЯ ПРОВЕРКИ (верификации) правильности работы программы формирования Книги счетов из Регистрационного журнала. **Отметим, что в графах Приход и Расход в Регистрационном журнале указываются КОДЫ ОБЪЕКТОВ, а в Книге счетов указывается КОЛИЧЕСТВО.** Оборотная ведомость по объекту учета является классической списковой формой с фильтром согласно некоторым настройкам системы, сортировкой и суммированием по полям — уровням отчета. Сводная ведомость по расходу является классической двухуровневой итоговой формой по уровням отчета «объект», «корреспондирующий объект». Сводная ведомость по приходу представляет для нас интерес как форма, которая должна быть получена из формы сводная ведомость по расходу как модификация с минимальными затратами, так как она рассчитывается аналогично ведомости по расходу, но вместо поля «расход» суммируется поле «приход».

Глава 3. Краткое описание языка программирования С

— Красиво – значит просто и со вкусом.
Коко Шанель о моде и о программировании.

Существуют следующие инструментальные среды (инструментальная среда – это не день после вторника с инструментами в руках, а комплекс программ, который обеспечивает разработку приложений):

— ассемблера (Turbo Assembler,...);
— классические языки программирования (C, C++, Pascal,...);
— системы управления базами данных (Oracle, FoxPro,...);
— классические языки программирования с расширенной библиотечной поддержкой (CBuilder, Delphi,...).

Ассемблер предназначен для работы на аппаратном уровне, например, для написания драйверов. Программы зависят от конкретной аппаратной платформы, и хоть сами операторы ассемблера очень простые, но работают на нем только профессионалы. А почему? А потому, что приходится разбираться с очень сложными структурами данных.

Классические языки программирования обеспечили независимость от аппаратной платформы и «человеческий» интерфейс операторов.

СУБД за счет вынесенного определения данных в отдельные структуры позволили автоматически или с минимальными затратами генерировать простые приложения типа «экранные формы» для сопровождения картотек и формирование печатных форм из этих картотек.

Языки программирования с расширенной поддержкой конечно хороши, но требуют хорошего знания используемых библиотек, т.е. достаточно высокой квалификации разработчика.

Вообще говоря, существуют еще электронные таблицы, но они эффективны только для разового ввода данных и рисования графика для курсового проекта.

В рамках курса ставится задача обучения элементам программирования на базе языка программирования С. В данной главе приводится классическое описание С для компилятора GCC. Если в описании нечто работает в GCC, но не является верным для любого компилятора, то это нечто помечено таким образом – (GCC). Отметим, что описание языка усечено до достаточного минимума для работы студента, например, указаны не все библиотечные функции, а только используемые в рамках проекта. Если некоторая возможность компилятора С GCC не является стандартом и не поддерживается всеми компиляторами (например, в С GCC можно использовать символ $ в обозначении переменных), то эта возможность в этом материале не описывается. В качестве инструментальной среды будем использовать комплект Geany/ GCC/ MinGW (Geany – сопровождение проекта, GCC – компилятор, MinGW – поддержка Windows).

3.1. Формальные требования к исходному тексту программы

3.1.1. Разделители и комментарии

Пробелы, символы табуляции перевода на новую строку и перевода страницы используются как разделители. Вместо одного из таких символов может использоваться любое их количество. Для повышения читабельности текста выгодно использовать символы табуляции. Отметим, что наиболее выгодна табуляция на 3 или 4 позиции.

Для описания исходного кода можно и нужно использовать комментарии. Комментарии начинаются парой символов /*, заканчиваются парой символов */ . Однострочные комментарии могут быть оформлены в стиле С++ с использованием пары слэшей // (GCC). Комментарии разрешены везде, где допустимы пробелы.

```
// Однострочный комментарий
a = b + 5; // Однострочный комментарий
/* Однострочный комментарий */
/*
Многострочный комментарий
Тяжела и неказиста жизнь тупого программиста
*/
```

Отметим, что **самый хороший комментарий не заменит хорошее именование переменных, простую логику алгоритма, структурирование алгоритма с использованием табуляции и использование пустых строк для выделения логического блока операций**. В примерах, приведенных ниже, мы не всегда будем делать именно так и правильно, а будем стараться построчно сжимать исходный текст с целью экономии места и, следовательно, денег студента.

Вложенные комментарии запрещены (GCC).

!!! Для комментария программного кода можно использовать препроцессор

```
#if 0
программный код
#endif
```

3.1.2 Идентификаторы

Идентификаторы используются как имена переменных, функций, меток и типов данных. Допустимые символы: цифры, латинские прописные и строчные буквы, символ подчеркивания (_) . Первый символ не может быть цифрой.

Отметим, что:

— идентификатор может быть произвольной длины, но не все символы учитываются компилятором (< 64)(GCC) и редактором связей (<32);

— **!!! прописные и строчные буквы – это разные символы.** При использовании только строчных букв и подчерка меньше проблем с надежностью программ.

3.1.3. Зарезервированные слова

Типы данных: **char, double, enum, float, int, long, short, struct, union, unsigned, void.**

Размер объекта **sizeof**. Вычисление sizeof выполняется во время компиляции.

Определение сокращенной формы описания или переопределения существующего типа данных **typedef**.

Классы памяти: **auto, extern, register, static.**

Операторы: **break, case, continue, default, do, else, for, goto, if, return, switch, while.**

3.2. Базовые типы данных

К базовым (основным) типам данных относятся целые числа (int, short, long, long long (GCC), unsigned), символы (char) и числа с плавающей точкой (float, double). На основе базовых типов данных строятся производные типы данных (структуры, объединения, перечисления).

3.2.1. Целые константы

Десятичные: цифры 0 — 9; для не нулевых констант первой цифрой не должен быть 0 .

2 111 956 1007

Восьмеричные: цифры 0 — 7; начинаются с 0. Заметим, что **восьмеричная система счисления является порождением безбуквенных клавиатур, то есть устарело**

012 = 10 (десятичное) 0123 = 1 * 64 + 2 * 8 + 3 = 83 (десятичное).

Шестнадцатеричные: цифры 0 — 9, буквы a — f или A — F для значений 10 — 15; начинаются с 0x или 0X.

0x 2 = 18 (десятичное) 0x1B9= 1 * 256 + 11 * 16 + 9 = 441 (десятичное)

3.2.2. Длинные целые константы

Длинная целая константа явно определяется латинской буквой l или L (long), буквами ll или LL (long long)(GCC), стоящими после константы. **Прописную l желательно не использовать в силу схожести начертания с 1.**

Длинная десятичная: 12l = 12 (десятичное) 956LL = 956 (десятичное)

Длинная восьмеричная: 012l = 10 (десятичное) 076LL = 62 (десятичное)

Длинная шестнадцатеричная: 0x12l = 18 (десятичное) OXA3LL = 163 (десятичное)

3.2.3. Константы с плавающей точкой

Константа с плавающей точкой всегда представляется числом с плавающей точкой двойной точности, т. е. как имеющая тип double, и состоит из следующих частей:

целой части — последовательности цифр;
десятичной точки;
дробной части — последовательности цифр;
символа экспоненты e или E;
экспоненты в виде целой константы (может быть со знаком).

Любая часть, но не обе сразу, из нижеследующих пар может быть опущена:

целая или дробная часть;
десятичная точка или символ e (E) и экспонента в виде целой константы.

345. = 345 3.14 = 3.14 47e--2 = 0.47

3.2.4. Символьные константы

Символьная константа состоит из одного символа кода ASCII, заключенного в апострофы

'A' 'a' 'T' '$'

Специальные символьные константы

Новая строка (перевод строки)	HL (LF)	' \ n '
Горизонтальная табуляция	HT	' \ t '
Вертикальная табуляция	VT	' \ v '
Возврат на шаг	BS	' \ b '
Возврат каретки	CR	' \ r '
Перевод формата	FF	' \ f '
Обратная косая	\	'\ \'
Апостроф	'	' \' '
Кавычки	"	' \" '
Нулевой символ (пусто)	NUL	' \ 0 '

Любой символ представим последовательностью трех восьмеричных цифр \ddd.

!!! Символьные константы считаются данными типа int.

3.2.5. Строковые константы

Строковая константа представляется последовательностью символов кода ASCII, заключенной в кавычки: "последовательность символов ...".

"This is a character string" "строка — константа" "A" "1234567890"

Строковая константа — это заключенная в кавычки последовательность символов. Она имеет тип char [ ] и в конце строки компилятор помещает нулевой символ ' \ 0 ', отмечающий конец данной строки.

!!! Каждая строковая константа, даже если она идентична другой строковой константе, сохраняется в отдельном месте памяти.

Если необходимо ввести в строку символ кавычек ", то перед ним надо поставить символ обратной косой \. В строку могут быть введены любые специальные символьные константы, перед которыми стоит символ \ . Символ \ и следующий за ним символ новой строки игнорируются.

3.3. Операции и выражения

3.3.1.Выражения

Выражение состоит из одного или нескольких операндов и символов операций.

```
a++   b = 10    x = ( y * z ) / w
```

!!! Выражение, заканчивающееся точкой с запятой, является оператором.

3.3.2. Метаобозначения операндов

!!! Метаобозначение – это обозначение группы объектов с какой-то своей системой обознаний, например, в зарубежной литературе для обозначения программы первого уровня часто используют метаобозначение foo.

Некоторые операции требуют операндов определенного вида. Вид операнда обозначается одной из следующих букв:

e — любое элементарное выражение (elementary);

v — любое выражение, ссылающееся на переменную, которой может быть присвоено значение (variable). Такие выражения называются адресными.

Префикс указывает тип выражения. Например, ie обозначает произвольное целое выражение. Далее описываются все возможные префиксы:

i — целое число или символ (integer);

a — арифметическое выражение (целое число, символ или число с плавающей точкой, arithmetic));

p — указатель (адрес с указанием типа объекта, pointer);

s — структура или объединение (structure);

sp — указатель на структуру или объединение (structure pointer);

smem — имя элемента структуры или объединения (structure member);

f — функция (function);

fp — указатель на функцию (function pointer).

Если в выражении должно быть несколько операндов, то они отличаются номерами, например ae1 + ae2.

3.3.3. Арифметические операции (в том числе и арифметика с указателями)

ae1 + ae2	Сумма значений ae1 и ae2. `new_page = start_page + 20 ;`
pe + ie	Адрес переменной типа pe, больший на ie адреса, заданного указателем pe. Например, присваиваем переменной last адрес последнего элемента массива array. `last = array + array _size — 1 ;`
ae1 — ae2	Разность значений ae1 и ae2. `new_page = end_page — 20 ;`
pe — ie	Адрес переменной типа pe, меньший на ie адреса, заданного указателем pe. `first = last – array _size + 1 ;`
pe1 — pe2	Число переменных типа pe в диапазоне от pe1 до pe2. `array _size = last – first ;`

— ae — Изменение знака ae.

```
x = --- x ;
```

ae1 * ae2 — Произведение значений ae1 и ae2.

```
size_all = size1 * number ;
```

ae1 / ae2 — Частное от деления ae1 на ae2.

```
number_record = size_file / size_record ;
```

ae1 % ae2 — Остаток от деления (деление по модулю) ae1 на ae2.

```
time_clock = time_min % 60 ;
```

iv ++ — Увеличение iv на 1. Значением этого выражения является значение iv до увеличения.

```
j = i ++ ;
```

++ iv — Увеличение iv на 1. Значением этого выражения является значение iv после увеличения.

```
i = ++ j ;
```

pv ++ — Увеличение указателя pv на 1, так что он будет указывать на следующий объект того же типа. Значением этого выражения является значение pv до увеличения. Например, присвоить значение 0 переменной, на которую указывает ptr, затем увеличить значение указателя ptr так, чтобы он указывал на следующую переменную того же типа.

```
* ptr ++ = 0 ; // опасный эксперимент, можно ошибиться.
```

++ pv — Увеличение pv на 1. Значением этого выражения является значение pv после увеличения.

```
* ++ ptr = 0 ; // опасный эксперимент, можно ошибиться.
```

iv — — Уменьшение iv на 1. Значением этого выражения является значение iv до уменьшения.

```
i = j — — ;
```

— — iv — Уменьшение iv на 1. Значением этого выражения является значение iv после уменьшения.

```
i = — — j ;
```

pv — — Уменьшение указателя pv на 1 так, что он будет указывать на предыдущий объект того же типа. Значением этого выражения является значение pv до уменьшения.

```
position = p — — ;
```

— — pv — Уменьшение pv на 1. Значением этого выражения является значение pv после уменьшения.

```
position = — — p ;
```

!!! При выполнении операций ++ и — — появляется побочный эффект в том, что изменяется значение переменной, используемой в качестве операнда. То есть, по разному отрабатывают операторы «++ переменная» и «переменная ++», «— — переменная» и «переменная — —». Как правило, вполне достаточно использования операторов «переменная ++» и «переменная— —» с последующим вычислением выражения в отдельном операторе. Использование всех возможностей ++, — — чревато повышенной вероятностью ошибок.

3.3.4. Операция присваивания

v = e Присваивание переменной v значения e.

```
y = x + sqrt ( 4 ) / 2 ;
```

!!! Значением выражения, в которое входит операция присваивания, является значение левого операнда после присваивания.

```
y = ( x = 3 ) + ( z++ ) ;
```

v Операция = e Эквивалентно v = v Операция e .

```
number  += counter ;      // увеличение переменной
ptr     +=  10 ;          // сдвиг указателя вперед
x       —=  3 ;           // уменьшение переменной
ptr     —=  step ;        // сдвиг указателя назад
time    *=  x ;           // умножение
x       /=  2 ;           // деление
x       %=  10 ;          // остаток от деления по модулю
x       >>= 4 ;           // битовый (двоичный) сдвиг вправо
x       <<= 2 ;           // битовый (двоичный) сдвиг влево
switch  &=  ~sw_delete ;  // битовое И
control ^=  sw_on ;       // битовое ИСКЛЮЧАЮЩЕЕ ИЛИ
switch  |=  sw_delete ;   // битовое ИЛИ
```

3.3.5. Операции отношения

Ложь представляется целым нулевым значением, Истина представляется любым ненулевым значением. Значением выражений, содержащих операции отношения или логические операции, является 0 (Ложь) или 1 (Истина).

ie1 == ie2 Проверка на равно. Истина, если ie1 равно ie2; иначеЛожь.

```
if ( j ==  0 ) break ;
```

pe1 == pe2 Проверка указателей на равно. Истина, если значения адресов указателей pe1 и pe2 равны.

```
if ( * file_in == NULL ) goto end ;
```

!!! Не путайте операцию отношения для проверки равенства == и операцию присвоения переменной = .

ie1 != ie2 Проверка на не равно. Истина, если ie1 не равно ie2.

```
i = 0 ; while ( i  != 100 )  { j = sin ( i * 3.14 ) ; i ++ ; } ;
```

pe1 != pe2 Проверка указателей на не равно. Истина, если значения адресов указателей pe1 и pe2 не равны.

```
if ( p != q )  printf ( " указатели не равны  \ n " ) ;
```

ae1 < ae2 Проверка на меньше. Истина, если ae1 меньше ae2.

```
If ( x < 0 )  printf ( " отрицательное число  \ n " ) ;
```

pe1 < pe2 Проверка указателей на меньше. Истина, если значение адреса pe1 меньше значения адреса pe2. Например, пока адрес, заданный p, меньше, чем адрес, заданный q, присваивать значение 0 переменной, на которую указывает p, и увеличивать значение p так, чтобы этот указатель указывал на следующую переменную.

```
while ( p  < q )  * p ++ = 0 ;
```

ae1 <= ae2 Проверка на меньше или равно. Истина, если ae1 меньше или равно ae2.

pe1 <= pe2 Проверка указателей на меньше или равно. Истина, если значение адреса pe1 меньше или равно значения адреса pe2.

ae1 > ae2 Проверка на больше. Истина, если ae1 больше ae2.

```
if ( x > 0 ) printf ( " положительное число  \ n" );
```

pe1 > pe2 Проверка указателей на больше. Истина, если значение адреса pe1 больше значения адреса pe2.

```
while ( p > q )  *p ---- = 0  ;
```

ae1 >= ae2 Проверка на больше или равно. Истина, если ae1 больше или равно ae2.

pe1 >= pe2 Проверка указателей на больше или равно. Истина, если значение адреса pe1 больше или равно значения адреса pe2.

3.3.6. Логические операции

! ae Логическое НЕТ. Истина, если ae ложно.

```
if  ( ! good ) printf ( " как не хорошо  \ n" ) ;
```

! pe Логическое НЕТ. Истина, если pe ложно.

```
if  ( ! file_in ) printf ( " плохой файл  \ n" ) ;
```

e1 || e2 Логическое ИЛИ. Вначале проверяется значение e1; значение e2 проверяется только в том случае, если значение e1 — Ложь. Значением выражения является Истина, если истинко значение e1 или e2.

```
if  ( x < 10  || x  > 30 )  printf ( " число вне  <10,30>  \ n" ) ;
```

e1 && e2 Логическое И. Вначале проверяется значение e1; значение e2 проверяется только в том случае, если значение e1 — Истина. Значением выражения является Истина, если значения e1 и e2 есть Истина. Например, если ptr не нулевой указатель и значение переменной, на которую указывает ptr, больше, чем 7, то увеличить number на 1. Обратите внимание, что если значение указателя ptr равно NULL , то выражение * ptr не имеет смысла.

```
if  ( ptr  !=  NULL  &&  * ptr  >  7 )  number ++ ;
```

!!! Не путайте логические операции НЕТ (!), ИЛИ (||), И (&&) и битовые операции НЕТ(~), ИЛИ (|), И (&).

3.3.7. Битовые операции

~ ie Инвертирование, дополнение до единицы, битовое НЕТ. Значение выражения содержит 1 во всех разрядах, в которых ie содержит 0, и 0 во всех разрядах, в которых ie содержит 1.

```
invert  =  ~ mask ;
```

ie1 & ie2 Битовое И. Значение выражения содержит 1 во всех разрядах, в которых и ie1 и ie2 содержат 1, и 0 во всех остальных разрядах.
!!! Может применяться для сброса и проверки битового флажка, как в нижеприведенных примерах

```
switch & =  ~ sw_delete ;    // сброс  признака
if ( ( x=switch ) &  sw_delete) != 0 ) printf ( "удалено" ); // проверка
```

ie1 | ie2 Битовое ИЛИ. Значение выражения содержит 1 во всех разрядах, в которых ie1 или ie2 содержит 1, и 0 во всех остальных разрядах. **!!! Может применяться для установки битового флажка,** как в нижеприведенном примере

```
switch |= sw_delete ;      // установка признака
switch = sw_delete | sw_edit ; // установка начальных значений
```

ie1 ^ ie2 Битовое ИСКЛЮЧАЮЩЕЕ ИЛИ. Значение выражения содержит 1 в тех разрядах, в которых ie1 и ie2 имеют разные двоичные значения, и 0 во всех остальных разрядах.

```
difference_bits = x ^ y ;
```

ie1 >> ie2 Битовый сдвиг вправо. Двоичное представление ie1 сдвигается вправо на ie2 разрядов. Сдвиг вправо может быть арифметическим (т. е. освобождающиеся слева разряды заполняются значением знакового разряда) или логическим в зависимости от реализации, однако гарантируется, что сдвиг вправо целых чисел без знака будет логическим, и освобождающиеся слева разряды будут заполняться нулями.

!!! Операцию можно использовать для более быстрого деления на 2 в степенях как в нижеприведенном примере

```
x = x >> 3 ;
```

ie1 << ie2 Битовый сдвиг влево. Двоичное представление ie1 сдвигается влево на ie2 разрядов; освобождающиеся справа разряды заполняются нулями.

!!! Операцию можно использовать для более быстрого умножения на 2 в степенях как в нижеприведенном примере

```
fourx = x << 2 ;
```

3.3.8. Адресные операции

& v Адрес переменной. Значением выражения является адрес переменной v.

```
name_ptr = & name ;
```

* pe Содержимое по адресу. Значением выражения является переменная, адресуемая указателем pe.

```
* array_ptr = array ;
```

* fpe Функция по адресу. Значением выражения является функция, адресуемая указателем fpe.

```
function_ptr = function_name ;
( * function_ptr ) ( arg1 , arg2 ) ;
```

3.3.9. Операции над массивами

pe [ie] Значением выражения является переменная, отстоящая на ie переменных от адреса, заданного указателем pe. Это значение эквивалентно значению выражения *(pe + ie). Например, присвоить значение 'q' 3—му элементу массива array.

!!! Первый элемент массива идет под номером 0.

```
array [ 2 ] = 'q' ;
```

3.3.10. Операции над структурами или объединениями

sv . smem — Элемент структуры или объединения. Значением выражения является элемент smem структуры или объединения sv. Например, присвоить значение 50 элементу cena структурной переменной product.

```
product . cena =  50 ;
```

spe —>smem — Элемент структуры или объединения по указателю. Значением выражения является элемент smem структуры (или объединения), на которую (ое) указывает spe. Это значение эквивалентно значению выражения (*spe) . smem. Например, присвоить значение 2 элементу cena структурной переменной, на которую указывает product_ptr.

```
product_ptr —> cena =  2 ;
```

!!! Использование указателей при работе со структурами более заметно, и в силу этого более надежно.

3.3.11. Другие операции

sizeof (e) — Число байт, требуемых для размещения данных типа e. Если e описывает массив, то в этом случае e обозначает весь массив, а не только адрес первого элемента, как во всех остальных операциях.

sizeof (Тип) — Число байт, требуемых для размещения объектов типа Тип. Например, вычислить число элементов в массиве целых чисел, определяемое как число байт в массиве, поделенное на число байт, занимаемых одним элементом массива.

```
number = sizeof ( array ) / sizeof ( int ) ;
```

(Тип) e — Значение e, преобразованное в указанный тип данных. Например, целое значение переменной number_int преобразуется в число с плавающей точкой перед делением на 3.

```
x =  ( float ) number_int  /  3 ;
```

3.3.12. Приоритеты и порядок выполнения операций. Преобразование типов. Порядок обработки операндов.

Смотри документацию.

!!! Не хочешь иметь проблем с ошибками из—за порядка выполнения операций, преобразования типов или порядка обработки операндов, тогда ВСЕГДА ставь скобки, указывай тип преобразования и не пиши длинные операторы присваивания.

3.4. Операторы

3.4.1. Формат и вложенность операторов

Один оператор может занимать одну или более строк. Два или большее количество операторов могут быть расположены на одной строке. Операторы, управляющие порядком выполнения (if, if—else, switch, while, do—while, for), могут быть вложены друг в друга.

3.4.2. Составной оператор

Составной оператор (блок) состоит из одного или нескольких операторов любого типа, заключенных в фигурные скобки.

!!! После закрывающейся фигурной скобки составного оператора не должно быть точки с запятой.

```
{ x = 1 ; y = 2 ; z = 3 ; }
```

3.4.3. Оператор—выражение

Любое выражение, которое заканчивается точкой с запятой, является оператором.

!!! Для обозначения пустого тела управляющего оператора используется пустой оператор, который состоит только из точки с запятой.

```
x = 2 + number ;   ++ number ;   fclose( file ) ;
if ( a <  0 ) ;  // пример классической ошибки в операторе if при
a = — a ;  // попытке вычисления абсолютного значения
```

3.4.4. Метка оператора

Метка : Оператор ;

Метка может стоять перед любым оператором, и служит для того, чтобы на этот оператор можно было перейти с помощью оператора goto. Областью определения метки является данная функция.

```
end :   fclose ( file ) ;
```

Оператор goto желательно не использовать с целью упрощения логики программы согласно принципам структурного программирования, тем более что без него можно легко обойтись. Использование goto может быть осмысленным при реализации конструкций типа оператора switch, как в нижеприведенном примере.

```
wk_start :
      if ( in == next )   { prg_next ( file_tmp ) ; goto wk_end ; }
      if ( in == back )   { prg_back ( file_tmp) ; goto wk_end ; }
wk_end :  printf ( " окончание работы \n" ) ;
```

3.4.5. Оператор безусловного перехода

goto Метка ;

Управление передается на оператор с указанной меткой. Область действия ограничена текущей функцией.

```
goto end_work ;
```

3.4.6. Оператор завершения внешнего оператора BREAK

break ;

Оператор прекращает выполнение ближайшего вложенного внешнего оператора switch, while, do или for. Управление передается следующему оператору.

```
cnt = 0 ;
while ( cnt < 200 )
      { cnt ++; if ( cnt == 100 ) break; printf ( "cnt= %i \n", cnt); }
```

3.4.7. Оператор продолжения внешнего оператора с новой итерации CONTINUE

continue ;

Оператор передает управление в начало ближайшего внешнего оператора цикла while, do или for, вызывая начало следующей итерации.

```
cnt = 0;
while ( cnt < 200 )
        { cnt ++; if ( cnt <= 100 ) continue; printf ( "cnt= %i \n",cnt); }
```

3.4.8. Оператор возврата RETURN

return ;
return (Выражение) ;

Оператор прекращает выполнение текущей функции и возвращает управление вызвавшей программе. При указании выражения вычисленное значение возвращается в точку вызова.

```
return ;            // выход без возвращаемого значения  в этом случае
                    // тип возвращаемого значения void в описании функции
return ( 0 ) ;      // выход с формированием кода возврата
return ( sqrt ( x ) ) ;  // выход с формированием  возвращаемого значения
```

3.4.9. Условный оператор IF—ELSE

if Условие Оператор ;
if Условие1 Оператор1 else Оператор2 ;

Если Условие истинно, то выполняется Оператор. Если выражение ложно, то оператор не выполняется.

```
if ( a = = x )  temp  =  3 ;
```

Если Условие1 истинно, то выполняется Оператор1 и управление передается на оператор, следующий за условным оператором (то есть Оператор2 не выполняется). Если Условие1 ложно, то выполняется Оператор2 и выполнение программы продолжается далее.

```
if ( x  > 1 ) z = 2 ; else  z =  6 ;
```

3.4.10. Оператор переключатель SWITCH

```
switch (Выражение)
{
case_константа1: Операторы1 ;
case_константа2: Операторы2 ;
. . .
default:_операторы ;
}
```

Сравнивает значение выражения с константами во всех вариантах case и передает управление оператору, который соответствует значению выражения.

!!! Легко, эффективно и надежно процедура выбора реализуется с использованием простого условного оператора if, то есть можно и нужно обходиться без оператора switch, если вы думаете о надежности программ.

3.4.11. Оператор цикла WHILE

while (Условие) Оператор ;

Если Условие истинно, то Оператор выполняется до тех пор, пока Условие не станет ложным. Если Условие ложно, то управление передается следующему оператору. Значение Условия определяется до выполнения Оператора.

```
cnt = 0 ;    // инициализация цикла
while ( cnt  <  20 )   // условие на выход из цикла
        { printf ( " cnt = % i   \ n", cnt );   cnt ++ ; } ;    // тело цикла
```

Если Условие истинно всегда, то имеем бесконечный цикл, из которого можно выйти только по break на некотором операторе if.

```
cnt = 0 ;
while ( 1 == 1 )
{ if ( cnt < 20 ) break ; printf ( " cnt = % i   \ n", cnt ) ;  cnt ++ ; } ;
```

3.4.12. Оператор цикла DO—WHILE

do Оператор while (Условие) ;

Оператор выполняется. Потом, если Условие истинно, то Оператор выполняется, и управление передается на начало цикла для вычисления Условия; это повторяется до тех пор, пока Условие не станет ложным. Если Условие ложно, то управление передается оператору, следующему за оператором цикла. Отметим, что значение Условия определяется после первого выполнения Оператора, поэтому Оператор выполняется хотя бы один раз.

```
x = 1 ;  // инициализация цикла
do  { printf ( " %d   \ n", power( x , 2 )); }  // тело цикла
 while ( ++ x   <=  7 ) ;              // условие на выход из цикла
```

!!! Так как, do—while сначала делает, а потом думает, что сделать, поэтому этот оператор лучше не использовать, а использовать классический оператор цикла while.

3.4.13. Оператор цикла FOR

for (Выражение1; Выражение2; Выражение3) Оператор ;

Выражение1 — инициализация цикла. Выражение2 — проверка условия завершения цикла. Если оно истинно, то выполняется Оператор (обычно его называют оператором тела цикла). Затем выполняется Выражение3. Все повторяется, пока Выражение2 не станет ложным. Если оно ложно, цикл заканчивается, и управление передается следующему оператору. Выражение3 вычисляется после каждой итерации.

Любое из выражений в операторе for может отсутствовать, однако разделяющие их точки с запятыми ; опускать нельзя. Если опущено Выражение2, то считается, что оно постоянно истинно. Оператор for (; ;) представляет собой бесконечный цикл, эквивалентный оператору while(1). Каждое из выражений может состоять из нескольких выражений, разделенных запятой.

```
for ( x = 1 , y = 3 , z = 4 ;      // инициализация цикла
        x <= 7;                    // условие на выход из цикла
        x ++ , y ++ , z ++ )    // действия по окончании обработки тела цикла
     printf ( "% d  \ n", power ( x , 2) ) ;    // тело цикла
```

!!! Оператор цикла for эквивалентен следующей последовательности операторов

```
Выражение1 ;
while (Выражение2)
        {
        Оператор ;
        Выражение3 ;
        }
```

поэтому можно обходиться без цикла for и, соответственно, вы избавляетесь от разделения операторов с помощью запятой, ибо запятую «правильный» программист использует только для разделения параметров при вызове функции.

3.5. Функции

3.5.1. Определение функции

Функция определяется описанием типа результата, описанием формальных параметров и составного оператора (блока), описывающего выполняемые функцией действия.

```
int     function ( int x , int a , int b )  // тип_возврата функции параметры
{
return ( a * x + b ) ; // возвращаемое значение
}
```

Значение выражения при необходимости преобразуется к типу результата функции. Оператор return может не возвращать значение в точку вызова функции.

!!! Функция, которая не возвращает значения, должна быть описана как имеющая тип возврата void.

```
void error_msg ( char *txt )
        {
        printf ( " *** error % s  \ n ", txt );
        }
```

3.5.2. Вызов функции

имя_функции (e1, e2, ... , eN)
(*указатель_на_функцию) (e1, e2, ... , eN)

Указатель_на_функцию — это переменная, содержащая адрес функции.

Адрес функции может быть присвоен указателю оператором

указатель_на_функцию = имя _функции ;

Аргументы (фактические параметры) передаются по значению, то есть каждое выражение e1, . . . , eN вычисляется и значение передается функции, например, загрузкой в стек.

Порядок вычисления выражений и порядок загрузки значений в стек не гарантируются.

Во время выполнения не производится проверка числа или типа аргументов, переданных функции. Такую проверку можно произвести с помощью программы lint до компиляции.

Вызов функции – это выражение, значением которого является значение, возвращаемое функцией.

Описанный тип функции должен соответствовать типу возвращаемого значения. Например, если функция function_xyz возвращает значение типа int, то эта функция должна быть описана до ее вызова:

```
extern  int  function_xyz ( ) ;   // описание внешней функции
```

Такое описание не определяет функцию, а только описывает тип возвращаемого значения; описание функции не требуется, если функция определена в том же файле до ее вызова.

3.5.3. Функция main

Каждая программа начинает работу с функции main ( ).

Во время выполнения программы можно передавать аргументы через формальные параметры argc и argv функции main.

Переменные среды языка передаются программе через параметр env.

```
        // в программу  передаются число параметров ,
        //  массив параметров-строк , массив переменных среды
main (  int argc , char ** argv , char ** env)
        {
        register  int i ;
        register  char  ** p ;
        // печать значений параметров
        for ( i = 0 ; i < argc ; i ++ )  printf ( " arg  % i  :  % s   \ n " , i ,  argv  [ i ] ) ;
        // печать значений переменных среды
        for ( p = env  ; * p  !=  ( char * ) 0 ;  p++ )     printf ( " % s   \ n ", *p ) ;
} // конец программы
```

3.6. Описания

Описания используются для определения переменных и для объявления типов переменных и функций, определенных в другом месте. Описания также используются для определения новых типов данных на основе существующих типов.

!!! **Описание не является оператором.**

3.6.1. Основные типы

Основными типами являются:

char	— символ или целое значение размером один байт;
int	— целое ;
unsigned	— неотрицательное целое ;
short	— короткое целое ;
long	— длинное целое ;
long long	— двойное длинное целое (GCC);
float	— число с плавающей точкой ординарной точности ;
double	— число с плавающей точкой двойной точности ;
void	— отсутствие значения (используется для указания отсутствия возвращаемого из функции значения).

Описание типа unsigned эквивалентно описанию типа unsigned int. Описание unsigned может сочетаться с описанием типа char, short, long или long long, формируя описания типов unsigned char, unsigned short, unsigned long, unsigned long long. Описания типов short, long и long long эквивалентны описаниям типов short int, long int и long long int .

3.6.2. Указатели и массивы

Допустимо бесконечно большое число различных типов указателей и массивов. Ниже приводятся примеры использования указателей и массивов.

Указатель на основной тип Переменная char_ptr есть указатель на символ

```
char * char_ptr ;
```

Указатель на указатель Переменная char_ptr2 есть указатель на указатель символа.

```
char * * char_ptr2;
```

Одномерный массив Переменная array_50 есть массив из 50 целых чисел.

```
int array_50 [ 50 ];
```

Двухмерный массив Переменная array_7_50 есть массив из семи массивов, каждый из которых состоит из 50 символов.

```
Char array_7_50 [ 7 ] [ 50 ];
```

Массив из семи указателей Переменная vector массив из 7—ми указателей на символы.

```
char * vector [ 7 ];
```

Указатель на функцию Переменная function_ptr указатель на функцию, возвращающую целое значение.

```
int ( * function_ptr ) ( );
```

3.6.3. Структуры

Структура объединяет логически связанные данные разных типов и используется в функциях и операторах при работе со всей группой данных.

Определение структурного типа данных

```
struct имя_структурного_типа
        {
        описания_элементов
        } ;
```

Пример определения структурного типа данных

```
struct obed
        {
        char   * mesto ;
        float   cena ;
        struct  obed  * next ;
        } ;
```

Описание структурной переменной (структуры) выполняется с помощью ранее описанного структурного типа данных.

```
struct obed obed_week [7] ;      // массив структур
struct obed obed_best ;          // одна структурная переменная
struct obed * obed_ptr ;         // указатель на структурную переменную
```

В некоторых системах определение структурного типа данных и переменной может быть выполнено одновременно следующим образом:

```
struct имя_структурного_типа
        {
        описания_элементов_структуры
        } переменная_1, . . . , переменная_n ;
```

Более того, может иметь место описание вида

```
struct  element , element _2     // тег структуры
        {
        int     element ;       // элемент структуры  в памяти
        int     element _2;     // элемент структуры  в памяти
        } element  , element _2, element _3  ;  // структуры в памяти
```

!!! Такое описание будет правильно в силу того, что **определение структурных переменных** (структура, объединение, перечисление), **элементы структурных переменных,** размещаемые в памяти **переменные** и **метки** компилятор разносит по разным областям видимости.

3.6.4. Битовое поле в структурах

Битовое поле – это элемент структуры, определенный как некоторое число бит, обычно меньшее, чем число бит в целом числе. Битовое поле предназначено для экономного размещения в памяти данных небольшого диапазона, например, флажков или простых битовых переключателей, или же при работе с системными структурными переменными.

```
struct  bit_field
        {
        unsigned int switch1 :10 ;  // первые 10 битов в слове
        unsigned int switch2 : 6 ;  // следующие 6 бит
        } ;
```

Данная структура описывает 10-битовое поле, которое для вычислений преобразуется в значение типа unsigned int, и 6-битовое поле, которое обрабатывается как значение типа unsigned int.

3.6.5. Объединения

Объединение предназначено для назначения на одну область памяти различных типов данных.

Определение объединенного типа данных (объединения)

```
union имя_объединенного_типа
        {
        описания_элементов_объединения
        };
```

Пример определения объединения:

```
union big_word
        {
        long    big_long ;
        char    * big_char [ 4 ] ;
        };
```

Данные типа union big_word занимают память, необходимую для размещения наибольшего из своих элементов, и выравниваются в памяти к границе, удовлетворяющей ограничениям по адресации как для типа long, так и для типа char * [4]

Описание переменных типа «объединение» выполняется с помощью ранее описанного типа данных.

```
union big_word  bw ;
union big_word  * bw_ptr ;
union big_word  bw_array [ 200 ] ;
```

В некоторых системах определение объединенного типа данных и переменной может быть выполнено одновременно следующим образом:

```
union имя_объединенного_типа
        {
        описания_элементов
        } переменная_1, . . . , переменная_n ;
```

!!! Объединение можно расценивать как дань традиционным языкам программирования без использования указателей. Использование указателей позволяет реализовать механизм наложения переменных в одной области памяти без использования данных типа «объединение».

!!! Перевод union есть пример плохой работы переводчиков. Название оператора объединение противоречит классическим операторам алгебры логики. Более правильно union было бы перевести как «содружество» или «пересечение».

3.6.6. Перечисления

Данные перечислимого типа позволяют именовать объекты некоторого ограниченного множества данных «красиво и правильно», но при работе с этими объектами будут использоваться int форматы данных, что позволяет обеспечить эффективную реализацию как по памяти и по быстродействию.

Определение перечислимого типа данных

```
enum имя_перечислимого_типа
        {
        список значений
        }
```

Пример определения перечислимого типа данных:

```
enum color { red, green, yellow };  // определения типа данных «перечисление»
```

Описание переменной перечислимого типа (перечисления) выполняется с помощью ранее описанного перечислимого типа данных.

```
enum color dom_color ;          // определение переменной «перечисление»
enum color zabor_color [ 40 ] ;
```

В некоторых системах определение перечисляемого типа данных и переменной может быть выполнено одновременно следующим образом:

```
enum имя_перечислимого_типа
        {
        описания_элементов
        } переменная_1, . . . , переменная_n ;
```

!!! Именование перечислимых объектов без использования данных типа «перечисление» можно эффективно реализовать с использованием оператора препроцессора #define.

3.6.7. Переименование (переопределение) типов

typedef старый_тип новый_тип

Переименование (переопределение) типов используется для введения осмысленных или сокращенных имен типов, что повышает понятность программ, а также для улучшения переносимости (мобильности) программ, например, в случае, когда имена одного типа данных могут различаться на разных ЭВМ.

```
typedef long large ; // определяется тип large, эквивалентный типу long
typedef char * string ; // определяется тип string, эквивалентный типу char *
```

3.6.8. Определение локальных переменных

Постоянные переменные, сохраняемые в некоторой области памяти, инициализируются нулем, если явно не заданы начальные значения. Временные переменные, значения которых сохраняются в стеке или регистре, не получают начального значения, если оно не описано явно.

!!! В большинстве компиляторов все описания в блоке должны предшествовать первому оператору. В GCC это не обязательно, но желательно выносить определение данных в начало блока с целью повышения надежности программирования.

Автоматические переменные

Автоматическая переменная является временной, так как ее значение теряется при выходе из блока. Областью определения является блок, в котором эта переменная определена. Переменные, определенные в блоке, имеют приоритет перед переменными, определенными в охватывающих блоках.

```
{
int x ;   // это автоматическая переменная
}
```

Регистровые переменные

Регистровые переменные являются временными, их значения сохраняются в регистрах, если последние доступны. Доступ к регистровым переменным более быстрый. В регистрах можно сохранять любые переменные, если размер занимаемой ими памяти не превышает разрядности регистра. Если компилятор не может сохранить переменные в регистрах, он трактует их как автоматические. Областью действия является блок.

!!! Операция получения адреса & не применима к регистровым переменным.

```
{
register int y ; // это регистровая переменная
}
```

Формальные параметры

Формальные параметры являются временными, так как получают значения фактических параметров, передаваемых функции. Областью действия формальных параметров является блок функции. Формальные параметры должны отличаться по именам от внешних переменных и локальных переменных, определенных внутри функции. В блоке функции формальным параметрам могут быть присвоены некоторые значения.

```
int function _sqrt ( int x )
{
return ( x * x) ;  // использование параметров в теле (блоке) функции
}
```

Статические локальные переменные

Статические локальные переменные являются постоянными, так как их значения не теряются при выходе из функции. Любые переменные в блоке, кроме формальных параметров функции, могут быть определены как статические. Областью действия статических локальных переменных является блок.

```
{
static int switch ; // это статическая переменная
}
```

3.6.9. Определение глобальных переменных

Глобальные переменные

Определяются на том же уровне, что и функции, т. е. определяются вне блоков. Постоянные переменные инициализируются нулем, если явно не задано другое начальное значение. Областью действия является вся программа. **Должны быть описаны во всех файлах программы, в которых к ним есть обращения.**

```
int global_switch ;      // это глобальная переменная
```

Статические глобальные переменные

Постоянные, так как их значения не теряются при выходе из функций. Областью действия является файл, в котором данная переменная определена. Должны быть описаны до первого использования в файле.

```
static int file_switch ;   // это статическая глобальная переменная
```

3.6.10. Инициализация переменных

Любая переменная, кроме формальных параметров или автоматических массива, структуры или объединения, при определении может быть инициализирована (присваивание начального значения).

Любая постоянная переменная инициализируется нулем, если явно не задано другое начальное значение. Это значит, что если переменная целая, то ее начальное значение равно 0, если символьная, то '\0', если это число с плавающей точкой, то 0.0.

Отметим, что **инициализация переменных может иметь смысл только в одноразовых программных входах**, а такие программы пишут только студенты в теме «поучимся». Более того, привычка к инициализации переменных при переходе к повторно входовым программам может привести к ошибкам.

Инициализация основных типов данных

```
int  I = 1 ;
float  x = 3.145 e – 2 ;
```

Инициализация массивов

```
int a[ ] = { 1 , 4 , 9 , 16 , 25 , 36 } ;
char s[ 20 ] = { 'a' , 'b' , 'c' } ;
```

Список значений элементов массива должен быть заключен в фигурные скобки. Если задан размер массива, то значения, не заданные явно, равны 0. **Если размер массива опущен, то он определяется по числу начальных значений.**

Инициализация строк

```
char hello[ ] = " привет " ;
```

Аналогичная реализация

```
char hello[ ] = { 'п', 'р', 'и', 'в', 'е', 'т', '\0' } ;
```

Инициализация структур

```
struct  persona
        {
        int     god ;   // год рождения
        char    pol ;   // признак  мужчина M – мужчина  Ж - женщина
        };
struct persona  shef = { 1950 , 'M' };
struct persona  deti [ ] =
        {
        { 2003, 'M' },
        { 1995, 'Ж' }
        };
```

Список значений для каждой структурной переменной должен быть заключен в фигурные скобки, хотя, если число значений соответствует числу структуры, это не обязательно. Значения присваиваются элементам структуры в порядке размещения элементов в определении структурного типа. Список значений может быть неполным, в этом случае неинициализированные элементы получают в качестве значения 0.

3.6.11. Описание внешних объектов

Тип внешних объектов (т.е. переменных или функций), определенных в другой компоненте программы, должен быть явно описан. Отсутствие такого описания может привести к ошибкам при компиляции, компоновке или выполнении программы.

При описании внешнего объекта используйте ключевое слово extern.

```
extern int  global_var , * name ;   // описание внешней переменной
extern int  function ( ) ;      // описание внешней функции
```

Можно опускать длину внешнего одномерного массива.

```
extern  float  num_array [ ] ;  // внешний одномерный массив
```

Поскольку все функции определены на внешнем уровне, то **для описания функции внутри блока прилагательное extern избыточно и часто опускается.**

```
int  function( ) ;      // описание внешней функции
```

Функция, не возвращающая значения, должна описываться как имеющая тип void. **Если тип функции явно не задан, считается, что она имеет тип int.** Областью действия описания на внешнем уровне является оставшаяся часть файла; внутри блока областью действия является данный блок. Обычно внешние описания располагаются в начале файла.

GCC допускает описание переменных на внешнем уровне без прилагательного extern. Многократные описания внешних переменных компоновщик (редактор связей) сводит к одному определению.

3.7. Препроцессор

Если в качестве первого символа в строке программы используется символ #, то эта строка является командной строкой препроцессора (макропроцессора).

Командная строка препроцессора заканчивается символом перевода на новую строку.

Если непосредственно перед концом строки поставить символ обратной косой (\ ), то эта строка будет объединена со следующей строкой программы.

3.7.1. Замена идентификаторов

```
#define идентификатор строка
#undef идентификатор
    #define ABC 100   // Заменяет каждое вхождение идентификатора
                      // ABC в тексте программы на 100
    #undef ABC        // отменяет предыдущее определение ABC
```

3.7.2. Макросы

#define идентификатор1 (идентификатор2,...) строка

Во избежание ошибок при вычислении выражений параметры макроопределения необходимо заключать в скобки.

```
#define abs ( A ) ( ( ( A ) > 0 ) ? ( A ) : — ( A ) )
```

Макрос abs обеспечивает замену каждое вхождения выражения abs(arg) в тексте программы на ((arg) > 0) ? (arg) : — (arg), причем параметр макроопределения A заменяется на arg

```
#define nmb_mem ( P , N ) \
    ( P ) —> p_mem [ N ] . u_long
```

Макрос nmb_mem уменьшает сложность выражения, описывающего массив объединений внутри структуры. Символ \ продолжает макроопределение на вторую строку.

3.7.3. Включение файлов

```
#include <имя_файла>
#include "имя_файла"
```

Командная строка #include может встречаться в любом месте программы, но обычно все включения размещаются в начале файла исходного текста.

Угловые скобки обозначают, что файл будет взят из некоторого стандартного системного каталога. Текущий каталог не просматривается.

Если имя файла заключено в кавычки, то поиск производится в текущем каталоге (каталог, в котором содержится основной файл исходного текста). Если в текущем каталоге данного файла нет, то поиск производится в каталогах, определенных именем пути в опции — I препроцессора. Если и там файл не обнаружен, то просматривается системный каталог.

```
#include < math.h >     // вставка содержимого системного файла
#include "my_head.h"    // вставка файла из текущего каталога
```

3.7.4. Условная компиляция

```
#if константное_выражение
    #if ABC + 3     // истина, если выражение ABC + 3 не равно нулю
#ifdef идентификатор
    #ifdef ABC      // истина, если идентификатор ABC определен ранее
#ifndef идентификатор
    #ifndef ABC     // истина, если идентификатор ABC не определен
#else
...
#endif
```

Командные строки препроцессора используются для условной компиляции различных частей исходного текста в зависимости от внешних условий

Если предшествующие проверки #if, #ifdef или #ifndef дают значение Истина, то строки от #else до #endif игнорируются при компиляции

Если эти проверки дают значение Ложь, то строки от проверки до #else (а при отсутствии #else — до #endif) игнорируются.

Команда #endif обозначает конец условной компиляции

3.8. Библиотека ввода-вывода

Программа, использующая перечисленные ниже функции ввода-вывода, должна включать в себя файл stdio.h с помощью команды препроцессора #include <stdio.h>. Здесь приведены только функции используемые в рамках проекта. Детальное описание функции лучше всего смотреть в help системах или непосредственно в заголовочных файлах.

3.8.1. Доступ к файлам

fopen — открыть поток ввода-вывода
fclose — закрыть открытый поток ввода-вывода stream
fseek — изменить текущую позицию offset в файле stream

3.8.2. Форматированный ввод-вывод

Функции printf, fprintf и sprintf описаны далее(смотри 3.10).
Функции scanf, fscanf и sscanf описаны далее (смотри 3.11).

3.8.3. Ввод символа

getc — прочитать следующий символ из входного потока
getchar — прочитать следующий символ из стандартного файла ввода

3.8.4. Блочный ввод-вывод

fread — прочитать из входного потока определенное число байт
fwrite — записать определенное число байт в выходной поток

3.9. Другие библиотеки

3.9.1. Обработка строк

Для выполнения описанных в этом подразделе функций необходимо включить в программу файл string.h с помощью команды препроцессора #include <string.h>.

strcmp — сравнить две строки в лексикографическом порядке
strncmp — сравнить первые n символов двух строк
strcpy — копировать строку s2 в строку s1
strncpy — копировать не более n символов строки s2 в строку s1
strlen — определить длину строки

3.9.2. Преобразование строки в число

atol — преобразование строки в длинное целое число long
atoi — преобразование строки в целое число int

3.9.3. Распределение памяти

malloc — выделение памяти
free — освобождение ранее выделенной памяти

3.10. Форматированный ввод

Для описания функций форматированного ввода scanf, fscanf, sscanf используются следующие метаобозначения:

П Пробел (символ П на самом деле не печатается!).

{ } Используется только один из перечисленных элементов.

[ ] Используется только один или не используется ни одного из перечисленных элементов.

/ Разделитель перечисляемых элементов.

Для использования функций printf, fprintf, sprintf в программу необходимо вставить команду препроцессора # include < stdio.h >.

Функции scanf, fscanf и sscanf могут иметь переменное число аргументов. Число и типы аргументов должны соответствовать спецификациям преобразования в строке определения формата ввода.

scanf — ввести данные из стандартного файла ввода stdin в соответствии со строкой **определения формата ввода,** присваивая значения переменным, заданным указателями на переменные

fscanf — ввести данные из потока stream в соответствии со строкой определения формата ввода, присваивая значения переменным, заданным указателями на переменные

sscanf — читать данные из строки в памяти в соответствии со строкой определения формата ввода, присваивая значения переменным, заданным указателями на переменные

3.10.1. Спецификация преобразования

%[*] [ширина] [дополнительные_признаки] символ_преобразования

Символ * обозначает пропуск при вводе поля, определенного данной спецификацией; вводимое значение не присваивается никакой переменной.

Ширина определяет максимальное число символов, вводимых по данной спецификации.

3.10.2. Пустые символы

Пробел или символ табуляции в форматной строке описывает один или более пустых символов. Пустые символы (пробел, символ табуляции, символы новой строки, перевода формата, вертикальной табуляции) во входном потоке в общем случае рассматриваются как разделители полей.

3.10.3. Литеральные символы

Литеральные символы в форматной строке, за исключением символов пробела, табуляции и символа %, требуют, чтобы во входном потоке появились точно такие же символы.

3.10.4. Спецификация ввода символа

% [*] [ширина] c

Ширина определяет число символов, которые должны быть прочитаны из входного потока и присвоены массиву символов. Если ширина опущена, то вводится один символ. По данной спецификации можно вводить пустые символы.

3.10.5. Спецификация ввода строки

% [*] [ширина] s

Ширина описывает максимальную длину вводимой строки. Строки во входном потоке должны разделяться пустыми символами; ведущие пустые символы игнорируются.

3.10.6. Спецификация ввода целого числа

% [*][ширина] [l / ll / h] { d / u / o / x }

Буква l определяет тип вводимых данных как long, ll – как long long (GCC) буква h - как short. По умолчанию принимается тип int.

Символы преобразования:

d — десятичное целое со знаком;

u — десятичное целое без знака;

o — восьмеричное целое без знака;

x — шестнадцатеричное целое без знака.

3.10.7. Спецификация ввода числа с плавающей точкой

% [*] [ширина] [l] { f / e / g }

Буква l определяет тип вводимых данных как double, по умолчанию принимается тип float. Символы преобразования f, e, g являются синонимами.

3.10.8. Спецификация ввода по образцу

% [*] [ширина] образец

Образец (сканируемое множество) определяет множество символов, из которых может состоять вводимая строка, и задается строкой символов, заключенной в квадратные скобки.

[a b c d]

Непрерывный (в коде ASCII) диапазон символов образца описывается первым и последним символами диапазона.

[a — z]

Если на первом месте в образце стоит символ ~, то вводиться будут все символы из входного потока, кроме перечисленных в образце, т.е. допустимое по этой спецификации множество символов будет дополнительным к описанному в шаблоне.

[~ 0 — 9]

По спецификации преобразования, заданной образцом, вводится строка символов, включая завершающий ее нулевой символ. Ведущие пустые символы не пропускаются.

3.11. Форматированный вывод

Для описания функций форматированного вывода printf, fprintf, sprintf используются следующие метаобозначения:

П Пробел (символ П на самом деле не печатается!).

{ } Используется только один из перечисленных элементов.

[ ] Используется только один или не используется ни одного из перечисленных элементов.

/ Разделитель перечисляемых элементов.

Для использования функций printf, fprintf, sprintf в программу необходимо вставить команду препроцессора # include < stdio.h >.

Функции printf, fprintf и sprintf имеют переменное число аргументов. Число и типы аргументов должны соответствовать спецификациям преобразования в строке определения формата вывода.

printf — записать аргументы в стандартный файл вывода stdout в соответствии со строкой определения формата вывода

fprintf — записать аргументы в поток stream в соответствии со строкой определения формата вывода

sprintf — записать аргументы в массив символов в соответствии со строкой определения формата вывода

Пример форматного вывода

```
printf ("error no. % d :  % s ", error , txt );
```

Печатается значение переменной error как десятичное целое и значение txt как строка. Результат форматированного вывода с точностью до значения переменных будет выглядеть следующим образом:

```
error no. 13 : cannot access file
```

3.11.1. Спецификация преобразования

%[выравнивание][ширина/ *] [доп_признаки]символ_преобразования

Выравнивание вправо по умолчанию. Выравнивание влево указывается — .

Ширина определяет минимальное число выводимых символов. Она может задаваться целым числом; если значение соответствующей переменной превышает явно заданную ширину, то выводится столько символов, сколько необходимо. Символ * обозначает, что число выводимых символов будет определяться текущим значением переменной.

```
printf ( " % *d " , number) ;  // вывод number в десятичном виде
```

3.11.2. Спецификация вывода символа

% [—] [ширина] с

```
%c    a
%3c   ППа
%—3c  аПП
```

3.11.3. Спецификация вывода строки

%[—] [ширина] [.точность] s

Точность определяет число печатаемых символов. Если строка длиннее, чем заданная точность, то остаток строки отбрасывается.

```
%10s      abcdefghij
%—10.5s   abcdeПППП
%10.5s    ПППППabcde
```

3.11.4. Спецификация вывода целого числа со знаком

%[—][+ / пробел][ширина] [l] d

Для отрицательных чисел автоматически выводится знак — (минус). Для положительных чисел знак + (плюс) выводится только в том случае, если задан признак +; если в спецификации задан пробел, то в позиции знака выводится пробел.

Символы преобразования

l — необходим для данных типа long ;

ll — необходим для данных типа long long (GCC);

d — определяет вывод данных типа int в десятичном формате со знаком.

```
% d   43
% +d  +43
% d   П43
```

3.11.5. Спецификация вывода целого числа без знака

%[—][#][ширина] [l / ll] { u / o / x / X }

Символ # определяет вывод начального нуля в восьмеричном формате или вывод начальных 0x или 0X в шестнадцатеричном формате. Символ l необходим для данных типа long, ll — для данных типа long long (GCC).

Символы преобразования:

u — десятичное без знака;

o — восьмеричное без знака;

x — шестнадцатеричное без знака;

X — шестнадцатеричное без знака с прописными буквами A — F.

```
% u   777626577
% lu  7626577
% o   5626321721
% x   2e59a3dl
%#X   OX2E59A3D1
```

3.11.6. Спецификация вывода числа с плавающей точкой

%[—][+ / пробел][#][ширина][.точность] { f / e / E / g / G }

Для отрицательных чисел автоматически выводится знак — (минус). Для положительных чисел выводится знак + (плюс), если задан признак +; если в спецификации задан пробел, то в позиции знака выводится пробел.

Завершающие нули не выводятся, если в спецификацию не включен признак #. Этот признак также обусловливает вывод десятичной точки даже при нулевой точности.

Точность определяет число цифр после десятичной точки для форматов f, e и E или число значащих цифр для форматов g и G. Округление "делается отбрасыванием. По умолчанию принимается точность в шесть десятичных цифр.

Типы аргументов float и double не различаются. Числа с плавающей точкой печатаются в десятичном формате.

```
%f     234.567890
%.3e   1.235e+03
%g     1234.57
```

Замечание. Чтобы вывести символ %, необходимо в форматной строке задать два символа %%.

```
printf("%5.2f %% \n", 99.44);
```

В результате выполнения данной функции будет напечатано 99.44%

Глава 4. Особенности реализации проекта

— Написать работающую программу не трудно,
трудно ее сопровождать.
Кто—то из великих, наверное, Питер Брукс.

Для разработки вышеописанного проекта необходимо установить инструментальную среду Geany/ GCC/ MinGW, являющуюся примером инструментальной среды на базе открытых инструментальных средств, т.е. с предоставлением исходных текстов разработчика и достаточно слабыми лицензионными ограничениями.

Предварительно отметим то, что выгодно разбивать физический диск вашего компьютера на логические диски с учетом функционального смысла хранимых данных, а именно, на следующие логические диски:

— система;
— типовые системные программы;
— программы и данные пользователя с использованием маленького кластера;
— программы и данные пользователя с использованием большого кластера;
— данные с хранением в течение некоторого периода;
— временные данные типа «поработал и удалил».

При работе на сервере БрГТУ используются следующие соглашения по использованию логических дисков:

— диск D рабочий;
— диски U и S системный архив (в качестве базового каталога преподавателя на данный момент времени используется каталог U : / VT&PM / EU_EI / PIN);
— диск R архив пользователя;
— диск Q используется в качестве почтового ящика.

Устанавливаем MinGW как типовую системную программу. Для этого скачиваем из Интернета с сайта MinGW загрузчик системы и запускаем его согласно размещенным на сайте инструкциям (можно посмотреть видео на YouTube). При выборе параметров установки обязательно указываем компилятор С и MinGW.

Устанавливаем инструментальную оболочку Geany. Процедура установки Geany реализована классическим копированием и, как правило, проблем не вызывает.

После этого модифицируем системную переменную Path в среде операционной системы, а именно, добавляем указание на каталоги с бинарниками MinGW.

Заметим, что при расширении системы до реализации с использованием графического режима необходимо установить дополнительные библиотеки, например, пакет GTK. Это выходит за пределы курса, но при желании будет приветствоваться.

В рамках лабораторных работ на основании описания упрощенной системы «Учет движения специальных жидкостей» (смотри 2.1) необходимо создать:

— картотеки (файлы для хранения линейно структурированной информации);

— программы для сопровождения этих картотек;

— программу формирования Книги счетов из Регистрационного журнала;

— программы для пошагового формирования печатных форм (выборка, сортировка, формирование печатной формы);

— меню для вызова соответствующих процедур (сопровождение картотеки, программа формирования картотеки, формирование печатной формы);

— пакетный bat—файл на запуск приложения и ярлык на запуск этого bat—файла;

— электронную версию описания системы (функциональную схему обработки данных, описание картотек, описание работ).

Описания структур данных в картотеках и настроечные константы проекта выносятся в отдельный заголовочный файл.

Требования к разрабатываемой системе:

— проект размещен в каталоге fioc_ верхнего уровня на личном диске, и все объекты именуются fioc_KTX, где fio есть первые буквы фамилии, имени, отчества; добавка с уточняет разрабатываемую подсистему (c — разработка на C). K есть код картотеки: p — первичные документы, o — объекты учета, j — регистрационный журнал, k — книга счетов, n — настройка системы. T есть тип объекта, который обозначается: s — экранная форма, q — запрос на выборку, u — сортировка (упорядочить), r — отчет, m — меню, p — программа. X есть модификатор объекта, например, x — fiof_psx стандартная процедура для ввода первичных документов, n — процедура fiof_csn для ввода данных при настройке системы, t — процедура fiof_cst для ввода данных при определении текущей даты;

— при создании проекта при именовании объектов руководствоваться следующими принципами: а) стандартное имя (8 символов – имя, 3 символа – расширение имени, используются только английские буквы), б) не более двух уровней каталожных структур, в) префиксация объектов, г) самоидентификация объектов, д) систематизированное именование объектов в проекте согласно некоторым ранее оговоренным соглашениям;

— обязательно использование для сохранения проекта архиватора от производителя, например, ARJ или ZIR/UNZIP.

Отметим, что для сопровождения картотек «Первичные документы» и «Объекты учета» используется классическая процедура сопровождения картотеки, для просмотра картотек «Регистрационный журнал» и «Книга счетов» используется классическая процедура «просмотр картотеки», для ввода данных при настройке системы, ввода текущей даты и пир определении выдаваемых форм используется классическая процедура ввода управляющих данных. Печатная форма «Оборотная ведомость по объекту» разрабатывается в упрощенном варианте, а именно, без расчета входящих и исходящих остатков, то есть выполняется только суммирование оборотов. Для указания месяца расчета используется более простой ввод интервального задания месяца (дата начала интервала, дата закрытия интервала). При выполнении работ необходимо максимально использовать процедуры копирования с последующим редактированием объекта, ибо новое, как правило, это хорошо отредактированное старое.

Отметим, что мы фактически имеем пять типов программ:

— меню, когда отрабатывается вывод элементов меню на монитор с подсказкой кода элемента, ввод кода элемента меню, выход на подпрограмму элемента меню и переход в начало программы на вывод элементов меню или на выход из программы;

— просмотр и корректировка картотеки настройка системы, когда отрабатывается открытие картотеки, отображение данных карточки картотеки, ввод данных карточки картотеки, закрытие картотеки и выход из программы. Отметим то, что картотека содержит всего одну карточку с реквизитами настройки системы и параметрами обработки данных;

— просмотр картотеки, когда отрабатывается открытие картотеки, отображение текущей карточки, далее ее редактирование или перемещение по картотеке или добавление карточки с последующим вводом данных или удаление карточки, закрытие картотеки и выход из программы;

— сопровождение картотеки, когда отрабатывается открытие необходимых картотек (картотека, картотека—справочник при необходимости, картотека—журнал при необходимости), отображение текущей карточки, далее ее редактирование непосредственно на поле данных или с использованием справочника, или перемещение по картотеке, или добавить карточку, или удалить карточку, или разноска данных, при необходимости, закрытие картотек и выход из программы;

— преобразование картотек, когда отрабатывается открытие исходной картотеки в режиме «читать», открытие формируемой картотеки (бинарный файл) или файла печатной формы (текстовый файл) в режиме «писать с начала», формирование выходного файла с использованием цикла пока не конец файла, закрытие файлов и выход из программы.

При наличии проблем с обеспечением собственной инструментальной среды при разработке проекта можно использовать среду TurboC с компилятором от Borland, размещенную на сервере университета с учетом особенностей TurboC.

В дальнейшем данный проект может быть реализован в среде CBuilder с использованием C++ с целью получения представления об объектно-ориентированном программировании в инструментальных средах с расширенной библиотечной поддержкой. В качестве отправной точки для внешнего представления экранных форм и образца для первоначального изучения, а также для копирования с последующей доработкой, необходимо использовать пример приложения на сервере университета. При реализации проекта на CBuilder при именовании объектов необходимо использовать расширенный вариант именования, а именно, объекты именуются fiob_KKK_TXX, где fiob идентифицирует разрабатываемую подсистему на CBuilder, а KKK есть код картотеки: pd — первичные документы, ou — объекты учета, rj — регистрационный журнал, ks — книга счетов, ns — настройка системы. Для реализации меню необходимо использовать закладки.

В дальнейшем данный проект может быть реализован с использованием СУБД, которые более эффективны при разработке таких систем. Именование объектов проекта выполнено по аналогии с разработкой на CBuilder. Поля таблиц максимально описаны согласно ранее сделанному предварительному описанию проекта. Примеры приложений находятся на сервере университета. В рамках реализации проекта на MS Access необходимо обратить внимание на проблемы с сопровождением проекта, которые обусловлены хранением программ и данных в одном файле, а также сложностью сопровождения программ на VBA Access.

В рамках курса информатики при выполнении работ, связанных с обучением работе с текстовыми и объектно—ориентированными редакторами, могут быть предложены работы по созданию материалов по проектированию системы. То есть необходимо создать электронную версию функциональной схемы обработки данных, описания картотек и описания работ в текстовом формате, HTML формате и MS WORD формате.

Материал также может быть использован при изучении MS Excel. В этом случае предлагается выполнить задание с использованием вышеуказанных картотек, например, формирование и разноска первичного документа (лист КПД) с использованием справочника «Объекты учета» (лист ОБЪЕКТЫ) с последующей разноской в Регистрационный журнал (лист РЖ), формирование Книги счетов (лист КС) из Регистрационного журнала (лист РЖ), формирование «Оборотной ведомости по объекту» (лист ОБОРОТ) с использованием параметров определения форм балансовой отчетности (лист ПАРАМЕТРЫ).

Глава 5. Особенности эксплуатации компьютерной бухгалтерии

— Эксплуатация «компа» безопаснее эксплуатации трудящихся.
Рокфеллер о компьютерных технологиях.

КОМПЬЮТЕР – это то, что:

— **быстро вычисляет**;

— **хранит большие объемы данных** с последующим быстрым извлечением требуемой информации для обработки;

— **алгоритмически обрабатывает данные** (способность выполнить с высокой точностью произвольное число раз некоторый расчет согласно ранее заданному описанию).

Нетрудно заметить, что в компьютерных системах исчезает трудоемкая и достаточно профессиональная работа сотрудников по заполнению и формированию отчетности согласно ранее сложившимся методикам. Вместо этого появляется работа по проверке (выверке) выполнения расчетных операций компьютерными программами.

Повышение производительности труда в компьютерной бухгалтерии обеспечивается за счет:

— использования при формировании первичных документов шаблонов (заготовок) при оформлении типовых, часто встречающихся или программно определяемых реквизитов (**принцип использования шаблонов**: новое — это хорошо скопированное и модифицированное старое);

— замены процедуры ввода с клавиатуры на процедуру выборки этих данных в соответствующих картотеках для ранее введенной информации (**принцип отсутствия вторичного клавиатурного ввода** — «мышка» дана для того, чтобы выбирать то, что введено ранее);

— замены ручного формирования бухгалтерской отчетности на программное формирование этой отчетности на базе картотек первичных документов с помощью соответствующих программных средств (**принцип программного формирования вторичных документов** — каждому свое, а именно, сотрудник вводит и выводит, компьютер формирует, начальник контролирует, программист делает непонятно что).

Основным результатом удачного внедрения компьютерных систем является следующее:

— **СИСТЕМАТИЗАЦИЯ РАБОТ.** Следствие: а) зачастую приходит понимание того, что же делается в организации, б) компьютер берет на себя часть контролирующих функций, т.к. требуется соблюдение определенного порядка работы, и неаккуратная или невыполненная работа, как правило, выплывает наружу;

— **РЕЗЕРВ ВРЕМЕНИ ДЛЯ СГЛАЖИВАНИЯ ПИКОВЫХ НАГРУЗОК** за счет производительности вычислительной техники. Простой исполнитель это может трактовать как уменьшение затрат ручного труда, которое выражается в более спокойной обстановке на работе и возможности сходить в магазин в рабочее время без срыва учетного цикла;

— **УМЕНЬШЕНИЕ ЗАВИСИМОСТИ РУКОВОДСТВА ОТ ПЕРСОНАЛА** за счет типизации выполнения работ и снижения требований к квалификации персонала за счет отнесения расчетной части на вычислительную технику.

Говорить о коммерческой выгоде в конкретных денежных суммах, которые рассчитываются согласно достаточно забавным методикам оценки эффекта от внедрения, не стоит, ибо на постсоветском пространстве в силу низкой заработной платы персонала основная компонента денежной экономии отсутствует.

При работе с компьютерными системами необходимо исходить из элементарных правил (заповедей):

— пользователь эксплуатирует компьютер, а не компьютер пользует пользователя;

— если не уверен, что завтра АРМ будет соответствовать твоим потребностям, то не забивай голову и не бей пальцы при внедрении системы;

— ничто так не украшает программы, как гарантии сопровождения;

— компьютерная система не обязана быть красивой, она должна быть надежной и эффективной;

— графика — для игрушек, текстовый режим — для эффективной работы;

— копирование есть основа восстановления;

— «комп» без вируса, что собака без блох;

— сеть — хорошо, а считать деньги надо в более надежном локальном режиме;

— если по понедельникам ходим в Интернет, а по вторникам на той же машине считаем зарплату, то расчет зарплаты – дело вероятностное;

— не бывает плохой компьютерной техники — бывают «криво» реализованные задачи;

— компьютерная система должна обеспечить: а) формирование первичных документов, б) выполнение расчетов и выдачу отчетности. Для раскладки пасьянсов она не предназначена;

— если при выборе типовой работы отрабатываешь более трех экранов, значит, у вас плохие проектировщики интерфейса;

— если долго не можешь найти поле на экранной форме во время расчета, то ваш АРМ можно использовать вместо «стрелялки»;

— Windows – хорошо, DOS – надежней, Linux – перспективней;

— ничто так не облегчает работу, как удачный АРМ, и ничто так не портит жизнь, как непредсказуемая компьютерная система;

— фирма без «компа», что рынок без «лоха».

Список изречений можно продолжить, но и вышесказанного вполне хватает для того, чтобы стать достаточно грамотным начальником в вопросах эксплуатации компьютерных систем.

И в заключение. После того, как вы самостоятельно создадите хоть и маленькую, но свою учетную систему, понимание того, что такое компьютерные системы, гарантировано. Успехов и хорошего сна перед экзаменом.

ЛИТЕРАТУРА

1. Болски, М.И. Язык программирования Си: Справочник. — М.:Радио и связь, 1988.
2. Уэит, М. Язык Си: руководство для начинающих / М. Уэйт, С. Прата, Д. Мартин — М.: Мир, 1988.

УЧЕБНОЕ ИЗДАНИЕ

Составители:
Мухов Сергей Владимирович
Муравьев Геннадий Леонидович
Парфомук Сергей Иванович

МЕТОДИЧЕСКИЕ МАТЕРИАЛЫ К ЛЕКЦИЯМ ПО ДИСЦИПЛИНЕ

Информатика и компьютерная графика

для студентов специальности
«Автоматизация технологических процессов и производств»

Программирование на С с использованием открытых инструментальных средств

Издание дополненное и переработанное

Ответственный за выпуск: Мухов С.В.
Редактор: Боровикова Е.А.
Компьютерная вёрстка: Соколюк А.П.
Корректор: Никитчик Е.В.

Подписано в печать 18.10.2017 г. Формат 60x84 1/16. Бумага «Performer».
Усл. печ. л. 2,55. Уч. изд. л. 2,75. Заказ № 1077 . Тираж 75 экз.
Отпечатано на ризографе учреждения образования «Брестский государственный технический университет». 224017, г. Брест, ул. Московская, 267.