МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ
«БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

КАФЕДРА ИНФОРМАТИКИ И ПРИКЛАДНОЙ МАТЕМАТИКИ

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

К ЛАБОРАТОРНЫМ РАБОТАМ ПО ДИСЦИПЛИНЕ

«Информатика, численные методы и компьютерная графика»

для студентов специальности
1 - 53 01 01 «Автоматизация технологических процессов и производств» (специализация 1 - 53 01 01 - 07 «Автоматизация технологических процессов и производств (промышленность строительных материалов)»)

БРЕСТ 2009

УДК 657.22(075)
ББК 65.052я73
М 92

В методических указаниях приводятся необходимые теоретические сведения по языку программирования С, который используется в качестве инструментального средства 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 указания предназначены для использования в ходе выполнения лабораторных работ по дисциплине «Информатика, численные методы и компьютерная графика» студентами специальности 1 - 53 01 01 «Автоматизация технологических процессов и производств» (специализация 1 - 53 01 01 - 07 «Автоматизация технологических процессов и производств (промышленность строительных материалов)»)

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

Рецензент: И.В. Котов, зав кафедры информационных технологий
БрГУ им. А.С.Пушкина, доцент, к.ф.м.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 «Сильно упрощенный АРМ «Баланс предприятия»», попробуем навести порядок в компьютерной терминологии с учетом специфики прикладного пользователя. Использовать в учебных целях модель обработки экономических данных 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, база данных. Как правило, термин «справочник» используется пользователем при наличии некоторой стабильности данных.

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, 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, запрос.

ШАБЛОН ВЫХОДНОЙ ФОРМЫ – специальное описание с указаниями, как формировать выходную форму с использованием понятия уровень отчета. Синонимы – отчет, описание отчета, **ОПИСАНИЕ ВЫХОДНОЙ ФОРМЫ.**

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

УРОВЕНЬ ОТЧЕТА – уровни иерархического расположения данных, которые «отплавиваются» программой формирования выходной формы при изменении указанных реквизитов для выполнения суммирования. Как правило, уровень отчета определен в описании формы, а сортировка по этим реквизитам определяется в описании запроса на формирование формы.

Как правило, выделяют списковые и итоговые отчетные формы. В качестве примера списковой формы из проекта, который будет создан в рамках лабораторных работ, можно привести «Оборотно–сальдовую ведомость по счету», соответственно, «Балансовая ведомость по счету» и «Журнал–ордер по счету» представляют собой итоговые формы.

Для понимания того, что и откуда берется в системе, используется графическое представление учетной модели, которое будем называть **КАРТОЙ СВЯЗЕЙ**. Пример карты связей для нашего проекта приводится на рис.1.1. Цифры в угловых скобках в реальной карте связей

отсутствуют, здесь же они указывают типовые работы по обработке данных, о которых говорится в главе 2.

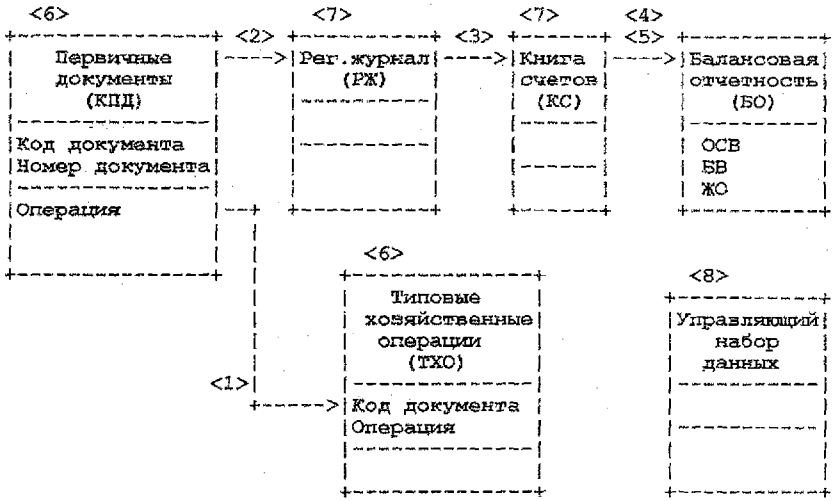

Рис.1.1.Карта связей упрощенной компьютерной бухгалтерии

Для **СХЕМАТИЧЕСКОГО ИЗОБРАЖЕНИЯ КАРТОТЕКИ** на карте связей 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 найдена дополнительная (расширенная) информация в карточке из другой картотеки. Обратим внимание на то, что в карте связей указаны только ключевые реквизиты и поля, используемые для связи картотек.

Необходимость использования карты связей вытекает из того факта, что для реквизитов, выбранных из других картотек, бухгалтер должен отрабатывать корректировку не на текущей картотеке, а на картотеках, где эти реквизиты содержатся. Карта связей 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, «Оборотно–сальдовая ведомость по счету» (ОСВ), «Балансовая ведомость по счету» (БВ) и «Журнал–ордер по счету» (ЖО).

На вышеуказанной карте связей можно видеть, что:

– на основании картотеки первичных документов формируется Регистрационный журнал;

– на основании Регистрационного журнала формируется Книга счетов;

– на основании Книги счетов формируется балансовая отчетность;

– занесение кода хозяйственной операции в картотеке первичных документов выполняется с использованием фильтра «код документа» из справочника «Типовые хозяйственные операции».

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

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). Пример описание картотеки приводится в табл.1.1

Картотека ТИПОВЫЕ ХОЗЯЙСТВЕННЫЕ ОПЕРАЦИИ Таблица 1.1.

Реквизит	Обозначение	Тип и значность
Код документа – ключ	txo_dokk	C3
Код операции – ключ	txo_txo	C10
Дебет	txo_db	C3
Кредит	txo_kr	C3

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** для нашего проекта в табл.1.2.

АРМ называется ФУНКЦИОНАЛЬНО ПОЛНЫМ АРМ'ом, если с его помощью на данном участке учета обеспечивается выполнение всех работ, выполняемых ранее вручную.

Описание работ АРМ'а «Баланс предприятия» Таблица 1.2.

Группа работ	Работы
Формирование первичных документов ДОКУМЕНТЫ	– Ввод текущей даты – Минимальный ввод и разноска первичных документов
Ведение рег.журнала и книги счетов РЖ	– Регистрационный журнал (РЖ) – Формирование книги счетов из рег.журнала – Просмотр книги счетов (КС)
Формирование балансовой отчетности БО	– Определение отчетных форм – Оборотно-сальдовая ведомость по счету – Балансовая ведомость по счету – Журнал-ордер по счету
Обслуживание картотек подсистемы «Баланс» КАРТОТЕКИ	– Типовые хозяйственные операции (ТХО) – 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:

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

– выделяются группы работ как единые методологически объединенные последовательности элементарных работ, например, группа работ «Месячный расчет»;

– выделяются в элементы меню верхнего уровня элементарные работы, связанные местом в технологии учетного процесса, например, обслуживание картотек первичных документов, сопровождение картотек, обеспечение архива и т.д.

При указании работы для ее корректного определения необходимо указывать элементы экранного меню верхнего уровня, например, указание «выполнить ежемесячный расчет» может выглядеть таким образом – «Месяц/Расчет».

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;

– функциональный вызов.

При этом могут использоваться поля специального вида, например, поле для обеспечения выборки из некоторого списка значений, переключатели и т.д., но такие поля сводятся к вышеуказанным базовым типам. Желательно, а в рамках лабораторных работ и обязательно, отдельно выделять ключевые поля (ключ – поле, однозначно определяющее карточку) и простые данные различной обводкой.

Типовые функциональные кнопки:

– **кнопки позиционирования** (выбрать, назад, вперед);

– **добавить** (возможны вариации – дублировать, добавить, вставить);

– **удалить** (возможны вариации – удалить, пометить к удалению, обход);

– **сортировка** (возможны вариации 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 работы обеспечиваются в виде функционального вызова в режиме «работа с карточкой». Для этого непосредственно на карточке (точнее, ее экранной форме), размещаются поля, «отработка» которых приводит к выполнению этой работы (так называемые кнопки или поля функционального вызова).

Попробуем определить некоторые термины, связанные с обработкой данных.

ТЕКСТ – поле на экранной форме с некоторым постоянным текстовым содержимым.

ОТОБРАЖАЕМОЕ (неизменяемое) ПОЛЕ – поле на экранной форме для отображения реквизита карточки без возможности его изменения.

РЕДАКТИРУЕМОЕ (изменяемое) ПОЛЕ – поле на экранной форме для отображения и корректировки (ввода) реквизита карточки.

ПЕРЕКЛЮЧАТЕЛЬ – поле на экранной форме для отображения реквизита типа ДА/НЕТ

ПОЛЕ (кнопка) ФУНКЦИОНАЛЬНОГО ВЫЗОВА – поле, выделенное на экранной форме, при отработке на котором мышкой происходит вызов некоторой функции (программы).

СБРОСИТЬ ПОЛЕ – установка переключателя в НЕТ (или 0, или [ ]), цифрового поля в 0, «денежного» поля в 0, символьного поля в пустую (пробельную) строку.

ВЫБРАТЬ ПОЛЕ – установить курсор мышки на поле и нажать (левую) кнопку мышки. В некоторых случаях требуется быстрое двойное нажатие кнопки мышки.

При выборке поля функционального вызова, как правило, используется двойное нажатие кнопки мышки, первое нажатие – установка поля, второе нажатие – вызов функции. В этом случае говорят ОТРАБОТАТЬ КНОПКУ или ВЫЗВАТЬ ФУНКЦИЮ.

Размещение и размер полей определяются разработчиком системы, но можно отметить, что наиболее выгодным местом для полей функционального вызова является место по кромке экрана. В этом случае легко выполняется позиционирование мышки на поле.

СОПРОВОЖДЕНИЕ ПРОГРАММНЫХ СРЕДСТВ есть модификация программного обеспечения исходя из изменения потребностей пользователей или обнаружения ошибок в его функционировании. Изменение потребностей пользователя могут определяться: а) изменением внешней среды функционирования системы, например, изменение текущего Законодательства, которое вызывает необходимость выполнять расчеты иным способом; б) желанием расширить функциональные возможности системы. **ОСНОВНЫМ ТРЕБОВАНИЕМ К ПРОГРАММНОМУ ОБЕСПЕЧЕНИЮ КОМПЬЮТЕРНЫХ СИСТЕМ С ИЗМЕНЯЕМОЙ ПРИКЛАДНОЙ ОБЛАСТЬЮ ЯВЛЯЕТСЯ НАЛИЧИЕ ГАРАНТИЙ СОПРОВОЖДЕНИЯ.**

ЭКСПЛУАТАЦИЯ ПРОГРАММНЫХ СРЕДСТВ есть ввод данных и запуск программ согласно соответствующим инструкциям.

Глава 2. Описание прикладной системы

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

И. Ньютон об изучении информатики.

В качестве примера системы для обработки данных будет использоваться модель упрощенного бухгалтерского учета. Как уже говорилось выше, в рамках данной модели отрабатываются все основные процедуры обработки данных, и, может быть, понимание учетных процедур пригодится нашим студентам в будущем. В данной главе предлагается упрощенная трактовка учетных процедур на примере взаиморасчетов между студентами, а именно, по каждому субъекту учета (студент) 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 каждого студента. 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** (см. рис.2.1). На основании Книги счетов легко получается самая простая отчетная форма под названием «Остатки в кармане на указанную дату».

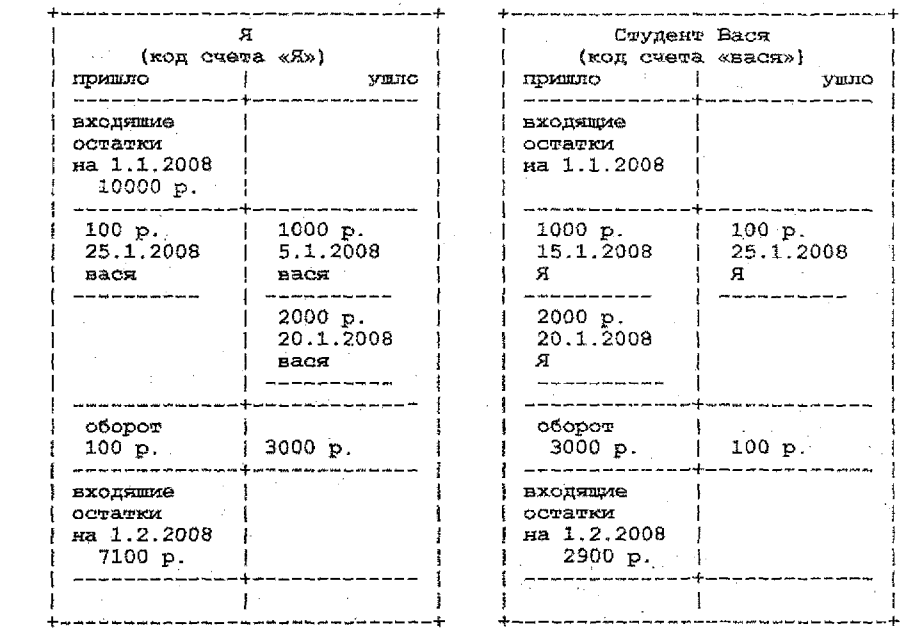

Рис.2.1. Пример самой маленькой в мире Книги счетов

В нашей системе (смотри карту связей на рис. 1.1) операция записывается сначала в картотеку первичных документов, причем при ее оформлении используется **<1>** выход на картотеку «Типовые хозяйственные операции» для выборки реквизита «Операция» с использованием в качестве фильтра код документа и с последующим занесением информации из выбранной записи «типовая операция» в запись «первичный документ». Завершающим аккордом обработки первичного документа является **<2>** формирование и занесение записи «проводка» в Регистрационный журнал на основании текущей записи «первичный документ». Далее выполняется **<3>** программное формирование всех записей Книги счетов из всех записей Регистрационного журнала с выборкой по фильтру «интервал + счет», который задан в управляющем наборе данных. В последствии из Книги счетов с помощью классических операций выборка, сортировка и формирование печатной формы по шаблону формируются **<4>** списковые и **<5>** итоговые печатные формы. Для сопровождения картотек создаются **<6>** классические экранные формы, **<7>** просмотровые экранные формы и **<8>** экранные формы для ввода управляющих данных. Все вышеуказанные работы (вызов экранных форм **<6>,<7>,<8>**; вызов программного модуля **<3>**; формирование печатных форм **<4>,<5>**) вызываются с помощью меню, для описания которого служит «Описание работ» (смотри табл. 1.2.). Реализация проекта (комплект программ в рамках одного программного вызова) и его запуск на выполнение – это просто завершающий технический штрих в нашей работе.

Из опыта разработки и сопровождения компьютерных систем можно сказать, что **ОПИСАНИЕ АРМ'а, которое достаточно и необходимо для его однозначного определение и может использоваться в качестве «Руководства пользователя», должно содержать:**

– карту связей между картотеками;

– описание картотек;

– описание выполняемых работ, которые сгруппированы технологически.

В рамках лабораторных работ необходимо разработать упрощенную систему бухгалтерского учета. Карта связей для нашего проекта приведена на рис.1.1. Описание картотек самостоятельно оформляются студентами в формате, приведенном в табл. 1.1, на основании изучения аналога нашей системы, который размещен на сервере университета. Описание работ приведено в табл. 1.2.

Внешний вид экранных форм и меню проектируется по аналогии с формами системы на сервере. Печатные формы, формируемые учебной системой, приведены на рис.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).

Выходные формы должны быть ориентированы на типизацию, а также учитывать специфику использования вычислительной техники, а именно: а) используются отчетные формы по СДЕЛОЧНОМУ 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 правильности работы программы формирования Книги счетов из Регистрационного журнала. **Отметим, что в графах Дебет и Кредит в Регистрационном журнале указываются КОДЫ СЧЕТОВ, а в Книге счетов указываются СУММЫ ПО ДЕБЕТУ И КРЕДИТУ.** Оборотно–сальдовая ведомость является классической списковой формой с фильтром согласно некоторым управляющим данным, сортировкой и суммированием по полям – уровням отчета. Журнал–ордер является классической двухуровневой итоговой формой по уровням отчета счет, корреспондирующий счет. Балансовая ведомость представляет для нас интерес как форма, которая должна быть получена из формы Журнал–ордер как модификация с минимальными затратами, так как она рассчитывается аналогично Журналу–ордеру, но суммируется поле дебет вместо поля кредит.

Глава 3. Краткое описание языка программирования С

– Красиво – значит просто и со вкусом.
Шанель о моде и программировании на С.

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 рекомендуется 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 */. Комментарии разрешены везде, где допустимы пробелы.

```
/* Однострочный комментарий */
```

```
/*
Многострочный комментарий
Тяжела и неказиста жизнь тупого программиста
*/
```

Регистрационный журнал за январь месяц 2008 г.

Дата	Документ	Содержание операции	Дебет	Кредит	Сумма
05.01.08	ВБ/123 от 05.01.08	Оплата за машину	Вася	Я	1000

Книга счетов за январь месяц 2008 г.

Дата	Документ	Содержание операции	Счет	Корр. счет	Дебет	Кредит
05.01.08	ВБ/123 от 05.01.08	Оплата за машину	Вася	Я	1000	
05.01.08	ВБ/123 от 05.01.08	Оплата за машину	Я	Вася		1000

Оборотно-сальдовая ведомость по счету .....
за ..... месяц 200.. г.

Дата	Документ	Содержание операции	Корр. счет	Входящее сальдо Дб	Входящее сальдо Кр	Обороты Дб	Обороты Кр	Исходящее сальдо Дб	Исходящее сальдо Кр
..	..	.....	..			..	..		
		Итого по счету		..	..	..	..	..	..

Балансовая ведомость по дебету счета .....
за ..... месяц 200.. г.

С кредита счетов	Сумма
..	....
Итого по Дб сч.N_...	....

Журнал-ордер по кредиту счета .....
за ..... месяц 200.. г.

В дебет счетов	Сумма
..	....
Итого по Кр сч.N ...	....

Рис.3.1. Печатные формы, формируемые учебной системой

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.

3.1.2 Идентификаторы

Идентификаторы используются как имена переменных, функций и типов данных. Допустимые символы: цифры 0 – 9, латинские прописные и строчные буквы a – z, A – Z, символ подчеркивания _ . Первый символ не может быть цифрой.

Отметим, что:

– идентификатор может быть произвольной длины, но в некоторых системах не все символы могут учитываться компилятором и загрузчиком;

– прописные и строчные буквы – это разные символы. При использовании только строчных букв меньше вероятность ошибки.

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, unsigned), символы (char) и числа с плавающей точкой (float, double). На основе базовых типов данных строятся производные типы данных.

3.2.1. Целые константы

Десятичные: цифры 0–9; первой цифрой не должен быть 0.

2 111 956 1007

Восьмеричные: цифры 0 – 7; начинаются с 0.

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.

0x12 = 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), стоящей после константы.

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

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

Длинная шестнадцатеричная: 0x12l = 18 (десятичное) OXA3L = 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 (десятичное) 3.14 = 3.14 (десятичное) 47e–2 = 0.47 (десятичное)

3.2.4. Символьные константы

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

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

Специальные (управляющие) 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" "Это строковая константа" "A" "1234567890"

Строковая константа – это последовательность символов, заключенная в кавычки; она имеет тип char [ ]. В конце каждой 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 большего числа операндов и символов операций.

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

Выражение, заканчивающееся точкой с запятой, является оператором.

3.3.2. Метаобозначения операндов

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

e – любое 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;

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

p – указатель (адрес с указанием типа объекта);

s – структура или объединение;
sp – указатель на структуру или объединение;
f – функция;
fp – указатель на функцию.
Обозначение smem указывает на имя элемента структуры или объединения.
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_ves = ves + 20;
```

pe + ie — Адрес переменной типа pe, больший на ie адреса, заданного указателем pe. Например, присваиваем переменной last адрес последнего элемента массива massive.
```
last = massive + mas_size – 1;
```

ae1 – ae2 — Разность значений ae1 и ae2.
```
new_ves = ves – 20;
```

pe – ie — Адрес переменной типа pe, меньший на ie адреса, заданного указателем pe.
```
first = last – mas_size + 1;
```

pe1 – pe2 — Число переменных типа pe в диапазоне от pe1 до pe2.
```
mas_size = last – first;
```

– ae — Изменение знака ae.
```
x = – x;
```

ae1 * ae2 — Произведение значений ae1 и ae2.
```
ves_2all = ves * nmb * 2;
```

ae1 / ae2 — Частное от деления ae1 на ae2.
```
ves1 = ves / nmb;
```

ae1 % ae2 — Остаток от деления (деление по модулю) ae1 на ae2.
```
minutes = time % 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 с точки зрения надежности */
```

++ pv — Увеличение pv на 1. Значением этого выражения является значение pv после увеличения.
```
*++ptr = 0;     /* опасный эксперимент с точки зрения надежности /*
```

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.*

```
tek_pos = p – –;
```

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

```
pred_pos = – – 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 .

```
nmb  += counter;        /* увеличение переменной */
ptr  += 10;             /* сдвиг указателя вперед */
x    –= 3;              /* уменьшение переменной */
ptr  –= step;           /* сдвиг указателя назад */
timesx *= 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 истинно значение e1 или e2.

```
if  (x < 10 ||  x >30)  printf (" вне интервала <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 – Истина. Например, если ptr не нулевой указатель и значение переменной, на которую указывает ptr, больше, чем 7, то увеличить nmb на 1. Обратите внимание, что если значение указателя ptr равно NULL , то выражение *ptr не имеет смысла.

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

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 ниже приведенных 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.

```
diff_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. **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. **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.

```
* ves_ptr = ves;
```

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

```
func_ptr = func_name;
(* func_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 massive. Обратите внимание, **первый элемент массива идет под номером 0.**

```
massive [ 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 (oe) 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;
```

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

3.3.11. Другие операции

ae ? e1 : e2

pe ? e1 : e2 Если истинно ae или pe, то выполняется e1; иначе выполняется e2. Значением этого выражения является значение выражения e1 или e2 в зависимости от значения pe.

```
abs = (i <= 0) ? – i : i;
```

Вообще говоря, лучше работать классически и «не грузить голову» нестандартными приемами программирования.

```
abs = i; if (i <= 0) abs = – i;
```

e1, e2 Сначала выполняется выражение e1, потом выражение e2. Значением всего выражения является значение выражения e2.

```
x = (i = 1, j = i * 2, z = j +20) + 12345;
for (i = 1, j = 40; i < 20; i++, j– –) p [ i ] = p [ j ];
```

Отметим, что запятая в качестве разделителя операций используется либо в сложных присвоениях, либо в цикле for. И то, и другое негативно сказывается на надежности программ.

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,

```
nmb = sizeof (massive) / sizeof (int);
```

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

```
x = (float) nmb_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, то 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 + nmb;  ++nmb;  fclose(file);
```

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, тем более что без него можно спокойно обойтись. Использование goto может быть осмысленным при реализации конструкций типа оператора switch как в нижеприведенном примере

```
        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;
return (0);
return (sqrt (x));
```

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 ничего не делается.

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

Если Условие1 истинно, то выполняется Оператор1 и управление передается на опера-

тор, следующий за условным оператором (т. е. 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 более контролируемо 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; while (cnt < 20) { printf («cnt = %i  \n", cnt);  cnt++; };
```

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;  do { printf (" %d  \n", power(x, 2)); }  while (++x <= 7);
```

Так как, do–while сначала делает, а потом думает, 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; x <= 7; x++, y++, z++)  printf ("%d  \n", power (x, 2));
```

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, формальных параметров и составного оператора (блока), описывающего выполняемые функцией действия.

```
double line_func (x, a, b)   /* тип результата   имя функции   список параметров */
        double x;            /* описание параметров */
        double a;
        double b;
{
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 void.**

```
void  error_msg(txt) 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, т. е. 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 func_xyz возвращает значение типа double, то эта функция должна быть описана до вызова:

```
extern double func_xyz ();
```

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 оболочки shell передаются программе через параметр env.

```
main (argc, aigv, envp)
        int argc;       /* число параметров*/
        char **argv;    /* вектор параметров-строк */
        char **env;     /* вектор переменных среды */
{
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 */
```

Параметры argv и envp могут быть описаны также следующим образом:

```
char   *argv [ ];
char   *env [ ];
```

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 (один байт);
int – целое (обычно одно слово);
unsigned – неотрицательное целое (такого же размера, как целое);
short – короткое целое (слово или полуслово);
long – длинное целое (слово или двойное слово);
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).

Символы (char) в зависимости от компилятора могут быть со знаком или без знака. Рассматриваемые как целые, символы со знаком имеют значения от –127 до 128, а символы без знака от 0 до 256. Описание типа unsigned эквивалентно описанию типа unsigned int. Описание unsigned может сочетаться с описанием типа char, short или long, формируя описания типов unsigned char, unsigned short, unsigned long. Описания типов short и long эквивалентны описаниям типов short int и long int. Диапазон данных типа long обычно в два раза больше диапазона данных типа short.

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 massiv_50 есть массив из 50 целых чисел.

```
int massiv_50 [50];
```

Двухмерный массив Переменная massiv_7_50 есть массив из семи массивов, каждый из которых состоит из 50 символов.

```
char massiv_7_50 [ 7 ] [ 50 ];
```

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

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

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

```
int (* func_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;
```

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.

```
struct  bit_field
        {
        unsigned int switch1:10;
        unsigned int switch2: 6;
        };
```

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. Заметим, что исходя из смысла объекта, Union было бы более правильно переводить не как объединение, а как содружество, пересечение, дружественный тип данных.

Определение объединенного типа данных

```
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_massiv [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.**

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 };
```

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

```
enum color dom_color;
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;
```

Отметим, что «красивое и правильное» именование объектов можно реализовать более эффективно с использованием #define без использования данных типа перечисление.

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.

Все описания в блоке должны предшествовать первому оператору.

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;   /* 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;
}
```

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  func (x);
        int  x;
{
...
}
```

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;
}
```

3.6.9. Определение глобальных переменных

Глобальные переменные

Определяются на том же уровне, что и функции, т. е. не локальны ни в каком блоке. 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;
```

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;
```

3.6.10. Инициализация переменных

Любая переменная, кроме формальных параметров или автоматических массива, структуры или объединения, при определении может быть инициализирована.

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.

В качестве начального значения может использоваться любое константное выражение.

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 = 3.145e – 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;
        char    pol;
        };
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;
extern char *name;
extern int  func ();
```

Можно опускать длину внешнего одномерного массива.

```
extern  float  num_array [ ];
```

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

```
int func();
```

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 остаток файла; внутри блока областью действия является данный блок. Обычно внешние описания располагаются в начале файла.

Некоторые компиляторы допускают 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 командная строка будет продолжена на следующую строку 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 для идентификатора 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 math.h будет взят из некоторого стандартного системного каталога. Текущий каталог не просматривается.

Если имя файла заключено в кавычки, то поиск производится в текущем каталоге (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 файла нет, то просматривается стандартный каталог.

```
#include  <math.h>          /* вставка  содержимого системного файла math.h  */
#include  "my_head.h"       /* вставка содержимого файла my_head.h  */
```

3.7.4. Условная компиляция

#if константное_выражение

```
  #if  ABC + 3     /* истина, если константное выражение ABC + 3 не равно нулю*/
```

#ifdef идентификатор

```
  #ifdef   ABC   /* истина, если идентификатор ABC определен ранее по #define */
```

#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>. 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
freopen – закрыть поток stream и открыть файл newfile, используя описание этого потока
fdopen – связать поток с дескриптором файла,открытым функцией open
fclose – закрыть открытый поток ввода-вывода stream
fflush – записать символы из буфера в выходной поток stream
fseek – изменить текущую позицию offset в файле stream
rewind – переставить указатель текущего байта в потоке на начало файла
setbuf – модифицировать буфер потока
setvbuf – модифицировать буфер потока

3.8.2. Доступ к каналам

pclose – закрыть поток, открытый функцией popen
popen – создать поток как канал обмена между процессами

3.8.3. Состояние файла

clearerr – обнулить признаки ошибки потока
feof – проверить состояние конца файла в потоке
ferror – проверить состояние ошибки в потоке
fileno – связать дескриптор файла, открытого функцией open с существующим потоком

3.8.4. Форматированный ввод-вывод

Функции printf, fprintf и sprintf описаны далее.
Функции scanf, fscanf и sscanf описаны далее.

3.8.5. Ввод-вывод строк

fgets – прочитать строку из входного потока, включая символ новой строки
gets – прочитать строку из стандартного файла ввода stdin
fputs – записать строку в поток stream
puts – записать строку в стандартный файл вывода stdout. В конце строки записывается символ новой строки

3.8.6. Ввод символа

fgetc – прочитать следующий символ из входного потока stream
Замечание. Функции getc(), getchar(), ungetc() являются *макроопределениями*.
getc – прочитать следующий символ из входного потока
getchar – прочитать следующий символ из стандартного файла ввода
ungetc – вернуть символ во входной поток

3.8.6. Вывод символа

fputc – записать символ в поток
Замечание. Функции putc() и putchar() являются макроопределениями.
putc – записать символ в поток
putchar – записать символ в стандартный файл вывода

3.8.7. 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>.

strcat – сцепить две строки
strncat – сцепить две строки, причем из второй строки копировать не более n символов
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
strlen – определить длину строки (число символов без завершающего нулевого символа)
strchr – найти в строке первое вхождение символа c
strrchr – найти в строке последнее вхождение символа c
strpbrk – найти в строке s1 любой из множества символов, входящих в строку s2
strspn – определить длину отрезка строки s1, содержащего символы из множества символов, входящих в строку s2
strcspn – определить длину отрезка строки s1, содержащего символы, не входящие в множество символов строки s2

3.9.2. Проверка символов

Замечание. Макроопределения, описанные в этом подразделе, определены в файле ctype.h, который должен быть включен в программы с помощью команды препроцессора #include < ctype.h>.

int имя_макроса (c) int c;
isalpha Символ c – буква
isupper Символ c – прописная буква (в верхнем регистре)
islower Символ c – строчная буква (в нижнем регистре)
isdigit Символ c – цифра (0 – 9).
isxdigit Символ c – шестнадцатеричная цифра (0 – 9), прописная (A – F) или строчная (a – f) буква
isalnum Символ c – буква или цифра
isspace Символ c – символ пробела, табуляции, перевода строки, новой строки, вертикальной табуляции или перевода формата
ispunct Символ c – символ пунктуации, т.е. не управляющий символ, не буква, не цифра и не пробел
isprint Символ c – печатный, т.е. имеющий значение (в кода ASCII) от 040 (пробел) до 0176 (тильда)
isgraph Символ c – графический, т.е. печатный, за исключением пробела
iscntrl Символ c – управляющий (0 – 037) или символ удаления (0177)
isascii Символ c – символ кода ASCII, т е его значение лежит в диапазоне от 0 до 0200

3.9.3. Преобразование символов

Замечание. Макроопределения, описанные в этом разделе, определены в файле ctype.h, который должен быть включен в программу, использующую эти макросы, с помощью команды препроцессора #include < ctype.h>.

int имя_макроса (c) int c;
toascu – преобразование целого в символ кода ASCII
tolower – преобразование буквы в нижний регистр
toupper – преобразование буквы в верхний регистр
_tolower – такая же, как tolower, но более быстрая и ограниченная функция
_toupper – такая же, как toupper, но более быстрая и ограниченная функция

3.9.4. Преобразование строки в число

strtol – преобразование строки в длинное целое число
atol – преобразование строки в длинное целое число
atoi – преобразование строки в целое число
atof – преобразование строки в число двойной точности с плавающей точкой
strtod – преобразование строки в число двойной точности с плавающей точкой

3.9.5. Доступ к аргументам

getenv – ввести строку, связанную с переменной оболочки
getopt – ввести следующий символ опций из списка аргументов

Для записи текущих значений индекса и указателя аргументов используются следующие внешние переменные

```
extern  char  *optarg;
extern  int     opind, opterr;
```

3.9.6. Распределение памяти

malloc – выделение памяти
calloc – выделение памяти и ее обнуление
realloc – изменение размера ранее выделенной памяти
free – освобождение ранее выделенной памяти

3.10. 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
```

3.10.1. 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", width, number);
```

3.10.2. Спецификация вывода символа

% [–] [ширина] c

```
%c    a
%3c   ППа
%-3c  аПП
```

3.10.3. 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   abcdeПППППП
%10.5s    ПППППabcde
```

3.10.4. 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;

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

```
%d    43
%+d   +43
% d   П43
```

3.10.5. Спецификация вывода целого числа без знака

%[–][#][ширина] [l] { 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.

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
%o    5626321721
%#o   05626321721
%x    2e59a3dl
%#X   OX2E59A3D1
```

3.10.6. 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
%.lf    1234.6
%E      1.234568E+03
%.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%

3.11. 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

3.11.1. 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.

3.11.2. 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.

3.11.3. Литеральные символы

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

3.11.4. 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.

3.11.5. Спецификация ввода строки

% [*] [ширина] s

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

3.11.6. Спецификация ввода целого числа

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

Буква l определяет тип вводимых данных как long, 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.

3.11.7. 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.

3.11.8. 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.

[abcd]

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.

[~ 0 – 9]

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

3.12. «ПРАВИЛЬНОЕ» ПРОГРАММИРОВАНИЕ НА С

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

Мобильность программ – это свойство, позволяющее выполнять программы на разных ЭВМ, работающих под управлением разных версий ОС с минимальными изменениями.

Использование в программе числовых констант, особенно когда смысл их неочевиден, является плохим стилем программирования. Числовые константы лучше определять в программе символическими именами, связанными с числовыми константами командой препроцессора #define. Такие определения легко находить и модифицировать, если они размещены в некотором стандартном месте. Обычно это начало программы или файл заголовка.

Все определения, связанные с конкретной операционной средой и конкретной ЭВМ, помещайте в файл заголовка.

Важнейшим средством разработки мобильных программ являются команды препроцессора #include и #define. Помещайте все определения типов данных, поименованных констант, макроопределения, используемые более чем одной программой, в единый файл заголовка так, чтобы все возможные изменения были локализованы.

Используйте файлы заголовка для общих определений одного проекта, т. е. множества связанных программ. Используйте локальные файлы заголовка для отдельных программ.

Используйте файл заголовка для локализации данных, зависящих от операционной среды, таких как имена файлов и режимы выполнения. Все, что может измениться при переходе на другую операционную систему или в рамках той же системы, помещайте в файлы заголовка, где эти данные легко могут быть модифицированы.

Для локализации программных фрагментов, зависящих от ЭВМ, используйте функции, условную компиляцию и команду #define. Используйте описание typedef для локализации определения типов данных, зависящих от ЭВМ.

Используйте #include <values.h>. С помощью этой команды в программу включается стандартный файл заголовка /usr/include/values.h , который содержит аппаратные константы.

Функции, зависящие от конкретной ЭВМ, объедините в отдельный исходный файл. Если таких файлов несколько, то соберите их в отдельном каталоге.

Фрагменты исходного кода, зависящие от аппаратуры, заключайте в команды условной компиляции.

Тщательно определяйте внешние имена. Максимальное число значимых символов в идентификаторах внешних переменных и функций зависит от операционной системы. Кроме того, некоторые операционные системы преобразуют все строчные буквы в прописные.

Глава 4. Особенности реализации проекта

– Проект – делу венец.
Программистская народная мудрость.

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, 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. Но все равно, классическое программирование – это достаточно сложная работа.

СУБД за счет вынесенного определения данных в отдельные структуры позволили автоматически или с минимальными затратами на интеллект 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 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. Для этого будет использована в качестве инструментального средства среда Turbo C. Несмотря на то, что пакет достаточно старый, он достаточно хорошо подходит для первоначального изучения С. Пакет и документация размещаются на сервере университета. На сервере также размещены методические указания по выполнению типовых работ в среде Turbo C.

В рамках лабораторных работ на основании описания упрощенной системы «Баланс предприятия» (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);

– проект с базовым объектом меню и соответствующим приложением;

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

– электронную версию описания системы (карта связей, 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 – Turbo C). K есть код картотеки: p – первичные документы, o – типовые хозяйственные операции, j – регистрационный журнал, k – книга счетов, c – управляющий набор. 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 ZIP/UNZIP.

Отметим, что для сопровождения картотек «Первичные документы» и «Типовые хозяйственные операции» 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 при определении выдаваемых форм используется классическая процедура ввода управляющих данных. Печатная форма «Оборотно–сальдовая ведомость по счету» 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.

В дальнейшем данный проект должен 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 при именовании объектов необходимо использовать расширенный вариант, а именно, объекты именуются fiob_KKK_TXX, где fiob идентифицирует разрабатываемую подсистему на CBuilder, а KKK есть код картотеки: pd – первичные документы, txo – типовые хозяйственные операции, rj – регистрационный журнал, ks – книга счетов, cd – управляющий набор. 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. Пример приложения находится на сервере университета. В рамках реализации проекта на 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 предварительных материалов по проектированию системы. То есть, необходимо создать электронную версию карты связей, описания картотек и описания работ в текстовом формате, HTML формате и MS WORD формате.

При отработке материала по MS Excel также могут быть предложены задания с использованием вышеуказанных картотек, например, формирование и разноска первичного документа (лист КПД) с использованием справочника «Типовые хозяйственные операции» (лист ТХО) с последующей разноской в Регистрационный журнал (лист РЖ), формирование Книги счетов (лист КС) из Регистрационного журнала (лист РЖ), формирование «Оборотно–сальдовой ведомости по счету» (лист ОСВ) с использованием параметров определения форм балансовой отчетности (лист ПАРАМЕТРЫ).

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

Основные СВОЙСТВА КОМПЬЮТЕРА как свойства супербухгалтера:

– способность **быстро вычислять**;

– способность **хранить большие объемы данных** с последующим быстрым извлечением требуемой информации для обработки;

– способность выполнить с высокой точностью произвольное число раз некоторый расчет согласно ранее заданному описанию (**алгоритмическая обработка данных**).

Нетрудно заметить, что в компьютерной бухгалтерии исчезает трудоемкая и достаточно профессиональная работа бухгалтера по заполнению и формированию бухгалтерских книг 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, «комп» формирует, начальник контролирует, «программер» 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;

– 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» поставленные задачи;

– компьютерная бухгалтерия должна обеспечить: а) формирование первичных документов, б) выполнение расчетов и выдачу балансовой 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 колхоз без колхозника.

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 компьютерную бухгалтерию, понимание того, что такое компьютерные системы, гарантировано. Успехов и хороших оценок на экзамене.

ЛИТЕРАТУРА

1. Кольвах О.И. Компьютерная бухгалтерия для всех.– Ростов н/Д: Феникс, 1996.– 416 с.

2. Болски М.И. Язык программирования Си: Справочник. – М.:Радио и связь, 1988.– 96 с.

УЧЕБНОЕ ИЗДАНИЕ

Составители: Мухов Сергей Владимирович
Муравьев Геннадий Леонидович

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

К ЛАБОРАТОРНЫМ РАБОТАМ ПО ДИСЦИПЛИНЕ

«Информатика, численные методы и компьютерная графика»

для студентов специальности 1 - 53 01 01 «Автоматизация технологических процессов и производств» (специализация 1 - 53 01 01 - 07 «Автоматизация технологических процессов и производств (промышленность строительных материалов)»)

Ответственный за выпуск: Мухов С.В
Редактор: Строкач Т.В.
Компьютерная верстка: Кармаш Е.Л.
Корректор: Никитчик Е.В.

Подписано к печати 13.01.2009 г. Формат 60х84 1/16. Бумага «Снегурочка» Усл.печ.л. 2,33. Уч.изд.л. 2,5. Тираж 100 экз. Заказ №6. Отпечатано на ризографе учреждения образования «Брестский государственный технический университет». 224017, г. Брест, ул. Московская, 267