МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ

Государственное образовательное учреждение
высшего профессионального образования
«Оренбургский государственный университет»

Колледж электроники и бизнеса

Л.А.ДЕЛЬ

ТЕХНОЛОГИЯ РАЗРАБОТКИ ПРОГРАММНЫХ ПРОДУКТОВ

МЕТОДИЧЕСКИЕ УКАЗАНИЯ К ЛАБОРАТОРЫМ РАБОТАМ

Рекомендовано к изданию Редакционно-издательским советом
Государственного образовательного учреждения
высшего профессионального образования
«Оренбургский государственный университет»

Оренбург 2009

УДК 681.3.06 (075.32)
Д-29
ББК 22.18я73

Рецензент
заместитель директора по научно методической работе Кузюшин С.А.

Дель, Л.А.
Д - 29 **Технология разработки программных продуктов[Текст]: методические указания к лабораторным работам/Л.А.Дель – Оренбург: ГОУ ОГУ, 2009. - 66 с.**

Методические указания предназначены для выполнения лабораторных работ, обеспечивающих учебный процесс по дисциплине “Технология разработки программных продуктов” в колледже электроники и бизнеса ГОУ ВПО ОГУ для студентов 3 курса в 5 семестре специальности 230105.51 “Программное обеспечение вычислительной техники и автоматизированных систем” очной формы обучения.

Рабочая программа составлена с учетом Государственного образовательного стандарта среднего профессионального образования по направлению подготовки дипломированных специалистов - утвержденного 08.02.2002 Министерством Образования Российской Федерации.

ББК 22.18я73

Содержание

Введение

Предмет “Технология разработки программных продуктов” является общепрофессиональной дисциплиной, устанавливающей базовый уровень знаний для освоения других общепрофессиональных и специальных дисциплин. В предмет “Технология разработки программного продукта” входят: основные понятия и определения, классификация программ, особенности создания программного продукта, жизненный цикл программы; проектирование программных продуктов; структура и формат, статические и динамические данные; модульное программмирование; стиль программирования, эффективность и оптимизация программ; отладка, тестирование, технологии программирования; защита программ; пакеты прикладных программ; коллективная разработка программных средств; экономические аспекты создания и использования программных средств.

Место дисциплины в учебном процессе

Предшествующие курсы, на которых непосредственно базируется дисциплина «Технология разработки ПП» являются:

- основы алгоритмизации и программирование;
- информационные технологии;
- операционные системы и среды;

Вместе с тем курс «Технология разработки ПП» является основополагающим для изучения дисциплины по выбору кафедры «Основы построения автоматизированных информационных систем», так же для дипломного проектирования и для практической деятельности молодых специалистов.

Особенности курса

Курс входит в число специальных дисциплин, определенных государственным образовательным стандартом среднего профессионального образования по направлению подготовки: «Информатика и вычислительная техника» специальности 230105.51 «Программное обеспечение вычислительной техники и автоматизированных систем».

Основная цель курса для студента: умение проектировать и разрабатывать программные системы

В результате изучения дисциплины студент должен:

а) знать принципы разработки и методы проектирования программного обеспечения, методы управления проектированием и организации коллективов разработчиков;

б) знать государственные стандарты и стандарты СТП;

б) уметь разрабатывать спецификации ПО, структуру ПО;

в) иметь представление о перспективах развития технологии ПО.

Для успешного изучения курса студенту необходимо знать курс основы алгоритмизации и программирования; объектно-ориентированное программирование.

Курс рассчитан на 62 часа лекций, 28 часов лабораторно-практических занятий, 30 часов курсового проектирования. Промежуточная оценка знаний и умений студентов проводится с помощью контрольных работ, которые включают в себя основные проблемы курса. Итоговый контроль в виде экзамена и зачета предусмотрен в 5-м семестре, в виде курсового проекта в 6-м семестре.

Лабораторные работы выполняемые в 5 семестре представлены в таблице 1

Таблица 1 - Лабораторные работы выполняемые в 5 семестре

№ Лаб. раб.	Наименование лабораторных работ	Кол-во часов
1	Построение меню. Разработка эскиза. Выбор режима диалога	2
2	Работа с модулем Unit. Составление программы.	4
3	Поиск ошибок с помощью отладчика (пошаговая отладка программы) Трансляция модулей. Создание выполнимой	4
4	Долговременно хранимые модули	2
5	Оверлейные программы	2
6	Динамическая память.	2
7	Линейные списки	4
8	Формирование списка с однонаправленным упорядочением его элементов	2
9	Исключение элемента из списка	2
10	Выполнение операций над списковыми структурами	4
Итого:		28 часов

1 Лабораторная работа №1 «Проектирование интерфейса пользователя»

Цель работы: Научиться разрабатывать сценарий диалогового режима, эскиз графического интерфейса

1.1 Ход работы

Диалоговый режим:

1) Разработать сценарий диалога, который представляется в виде:

- *меню* - диалог инициируется программой пользователя предлагается выбор альтернативных функций обработки из фиксированного перечня; меню может быть иерархическим и содержать вложенные подменю следующего уровня;
- действия *запрос-ответ* - фиксированный перечень возможных значений, выбираемых из списка ... Или ответы типа Да/Нет;
- *запрос по формату* - с помощью ключевых слов, фраз или путем заполнения экранных форм с регламентированным по составу и структуре набором реквизитов осуществляется подготовка сообщений.

2) Для сценария определить:

- *точки* (момент, условие) начала диалога;
- *инициатор диалога* - человек или программный продукт;
- *параметры и содержание диалога* - сообщения, состав и структура меню, экранные формы и т.д.;
- *реакция* программного продукта на завершение диалога.

3) Определить типы диалоговых окон, содержащие объекты управления:

- тексты сообщений;
- поля ввода информации пользователя;
- кнопки и т.д.

4) Разработать эскиз графического интерфейса пользователя.

1.2 Содержание отчета

1) Постановка задачи;
2) Исходные данные;
3) Сценарий диалогового режима;
4) Эскиз и спецификация используемых компонентов.

1.3 Вопросы на допуск к лабораторной работе

1) Что такое предметная область?
2) Дайте определение задачи и приложения;
3) В чем состоит постановка задачи?
4) Назовите основные этапы и работы по созданию программного продукта;
5) Каковы особенности диалогового режима работы программного продукта?
6) Укажите основные свойства диалогового режима;
7) Дайте определение графического интерфейса;
8) Что такое объект управления в графическом интерфейсе пользователя?

1.4 Методические указания

В диалоговом режиме под воздействием пользователя осуществляются запуск функций (методов) обработки, изменение свойств объектов, производится настройка параметров выдачи информации на печать и т.п. Системы, поддерживающие диалоговые процессы, классифицируются на:

- системы с *жестким сценарием диалога* — стандартизированное представление информации обмена;
- *дескрипторные системы* — формат ключевых слов сообщений;
- *тезаурусные системы* — семантическая сеть дескрипторов, образую-щих словарь системы (аналог — гипертекстовые системы);
- системы с *языком деловой прозы* — представление сообщений на языке, естественном для профессионального пользования.

Наиболее просты для реализации и распространены диалоговые системы с жестким сценарием диалога, которые представлены в виде:

- *меню* — диалог инициируется программой; пользователю предлагается выбор альтернативы функций обработки из фиксированного перечня; предоставляемое меню может быть иерархическим и содержать вложенные подменю следующего уровня;
- действия *запрос-ответ* — фиксирован перечень возможных значений, выбираемых из списка, или ответы *типа Да/Нет;*
- *запрос по формату — с* помощью ключевых слов, фраз или путем заполнения экранной формы с регламентированным по составу и структуре набором реквизитов осуществляется подготовка сообщений.

Диалоговый процесс управляется согласно созданному *сценарию,* для которого определяются:

- точки (момент, условие) начала диалога;
- инициатор диалога — человек или программный продукт;
- параметры и содержание диалога — сообщения, состав и структура меню, экранные формы;
- реакция программного продукта на завершение диалога.

Описание сценария диалога выполняют:

- *блок-схема,* в которой предусмотрены блоки выдачи сообщений и обработки полученных ответов;

- *ориентированный граф,* вершины которого — сообщения и выполняемые действия, дуги — связь сообщений; словесное описание;

- специализированные объектно-ориентированные языки построения сценариев.

Для создания диалоговых процессов и интерфейса конечного пользователя наиболее подходят объектно-ориентированные инструментальные средства разработки программ.

В составе инструментальных средств СУБД содержатся *построители меню,* с помощью которых создается ориентированная на конечного пользователя совокупность режимов и команд в виде *главного меню* и *вложенных подменю*. Конструктор *экранных форм* СУБД используется для разработки форматов экранного ввода и редактирования данных базы данных и входной информации, управляющей работой программного продукта.

В ряде СУБД и электронных таблиц, текстовых редакторов существуют различные типы *диалоговых окон,* содержащих разнообразные объекты управления:

- тексты сообщения;
- поля ввода информации пользователя;
- списки возможных альтернатив для выбора;
- кнопки и т.п.

В среде электронных таблиц и текстовых редакторов имеются возможности настройки главных меню (удаление ненужных, добавление новых режимов и команд), создания системы подсказок с помощью встроенных средств и языков программирования.

Графический интерфейс пользователя (Graphics User Interface—GUI)—ГИП является обязательным компонентом большинства современных программных продуктов, ориентированных на работу конечного пользователя. К графическому интерфейсу пользователя предъявляются высокие требования как с чисто инженерной, так и с художественной стороны разработки, при его разработке ориентируются на возможности человека.

Наиболее часто графический интерфейс реализуется в интерактивном режиме работы пользователя для программных продуктов, функционирующих в среде Windows, и строится в виде системы спускающихся *меню с* использованием в качестве средства манипуляции мыши и клавиатуры. Работа пользователя осуществляется с *экранными формами,* содержащими *объекты управления, панели инструментов* с *пиктограммами* режимов и команд обработки.

Пример. Средствами редактора диалогов Microsoft Word Dialog Editor построено диалоговое окно, обеспечивающее графический интерфейс пользователя (рисунок 1) В таблице 2 показана спецификация типовых объектов управления графического интерфейса спроектированного диалогового окна.

Таблица 2 - Спецификация типовых объектов управления графического интерфейса

Наименование объекта		Функциональная характеристика объекта
1	2	3
Метка	label	постоянный текст, не подлежащий изменению при работе пользователя с экранной формой (например, слова *Фамилия Имя Отчество);*
текстовое окно	text box	используется для ввода информации произвольного вида, отображения хранимой информации в базе данных (например, для ввода фамилии студента);
Рамка	frame	объединение объектов управления в группу по функциональному или другому принципу (например, для изменения их параметров);
Командная Кнопка	command button	обеспечивает передачу управляющего воздействия, например, кнопки <Cance1>, <ОК>, <0тмена>; выбор режима обработки типа <Ввод>, <Удаление>, <Редактирование>, <Выход> и др.;
кнопка-переключатель	option button	для альтернативного выбора кнопки из группы однотипных кнопок (например, *семейное положение);*
помечаемая кнопка	check button	для аддитивного выбора несколько кнопок из группы однотипных кнопок (например, *факультатив для посещения);*
окно-список	list box	содержит список альтернативных значений для выбора (например, «Спортивная секция»);
комбинированное окно	combo box	объединяет возможности окна-списка и текстового окна (например, «Предметы по выбору» — можно указать новый предмет или выбрать один из предлагаемого списка);
линейка горизонтальной прокрутки		для быстрого перемещения внутри длинного списка или текста по вертикали
линейка вертикальной прокрутки		для быстрого перемещения внутри длинного списка или текста по вертикали

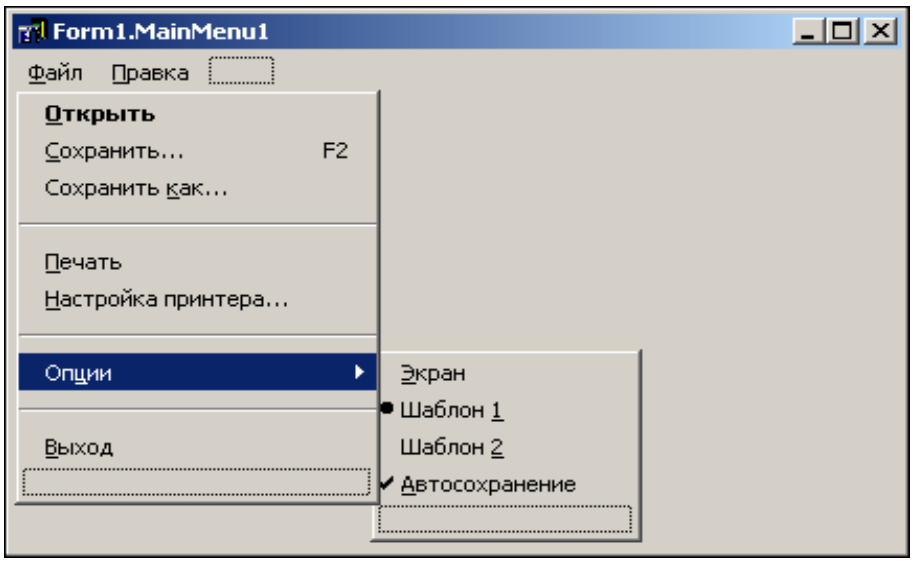


Рисунок 1 - Пример диалогового окна

1.5 Варианты индивидуальных заданий к зачету по теме «Проектирование интерфейса пользователя»

Разработать сценарий диалогового режима, а так же эскиз и спецификацию используемых компонентов к программам:

1) Программа «Помощник библиотекаря»;
2) Программа учета заказов на изготовление деталей на ПО «Стрела»;
3) Программа «Автострахование»;
4) Программа для автоматизации документов склада ЗАО ТПК «Орентекс»;
5) Электронный учебник по специальности «Оператор ЭВМ»;
6) Программа учета кредитов ОАО «Сельский дом»;
7) Тестируемая программа для студентов факультета ПОВТАС;
8) Электронный учебник по предмету «Теория вероятностей»;
9) Программа «Станция медицинской скорой помощи»;
10) Программа расчета расхода горючего на транспорт;
11) Программа «Растения в нашей жизни»;
12) Программа «Домашние животные»;
13) Программа «Моя семья»;
14) Программа «Электронный староста».

2 Лабораторная работа №2 «Создание модуля. Трансляция модуля. Создание выполнимой программы»

Цель работы: Научиться создавать модули (UNIT)

2.1 Ход работы

1) Для конкретного варианта реализовать в виде модуля набор подпрограмм для выполнения заданных в варианте операций;
2) Определить структуру модуля;
3) Набрать и оттранслировать программу пункта 3 методических указаний к лабораторной работе;
4) Создать файлы Unitlab.pas, lab2.pas;
5) Сохранить объектный модуль на диске;
6) Распечатать текст программы и результат;
7) Вывод.

2.2 Содержание отчета

1) Постановка задачи;
2) Исходные данные;
3) Текс программы и результаты ее выполнения;
4) Анализ допущенных ошибок.

2.3 Вопросы для допуска к лабораторной работе

1) Для чего внедряется модульное программирование?
2) Структура модуля?
3) Для чего нужна интерфейсная часть в модуле (INTERFACE)?
4) Что включает в себя раздел реализации (IMPLEMENTATION)?
5) Что нужно сделать для сохранения программы как объектный модуль?

2.4 Методические указания

Модуль (UNIT) —программная единица, текст которой компилируется независимо (автономно). Она включает определения констант, типов данных, переменных, процедур и функций, доступных для использования в вызы-вающих программах. Однако внутренняя структура модуля (тексты программ и т.п.) скрыта от пользователя.

Термины UNIT и модуль являются синонимами **и обозначают** одно и то же понятие в разных языках.

Структура модуля (UNIT)

Модуль можно разделить на несколько разделов: заголовок, интерфейсная часть, реализационная часть, инициализационная часть.

```
    ЗАГОЛОВОК МОДУЛЯ
UHIT  Имя модуля;
        {$N+}                          Глобальные директивы  компилятора;
ИНТЕРФЕЙСНАЯ ЧАСТЬ
INTERFACE                                        Начало раздела объявлений;
USES                                  Используемые при объявлении модули;
LABEL                Подраздел объявления доступных глобальных меток;
 CONST          Подраздел объявления доступных глобальных констант;
TYPE                  Подраздел объявления доступных глобальных типов;
 VAR          Подраздел объявления доступных глобальных   переменных;
PROCEDURE                                    Заголовки доступных процедур;
FUHCTIOH                                     Заголовки доступных функций;
РЕАЛИЗАЦИОННАЯ ЧАСТЬ
IMPLEMENTATION                                   Начало раздела реализации;
 USES                                 Используемые при реализации модули;
 LABEL               Подраздел объявления скрытых глобальных меток;
 CONST            Подраздел объявления скрытых глобальных констант;
 TYPE                 Подраздел объявления скрытых глобальных типов;
 VAR           Подраздел объявления скрытых глобальных переменных;
PROCEDURE                          Тела доступных и скрытых процедур;
FUHCTIOH                           Тела доступных и скрытых функций;
ИНИЦИАЛИЗАЦИОННАЯ ЧАСТЬ
BEGIN
       Основной блок модуля;
 END.
```

Заголовок модуля

Заголовок модуля мало отличается от заголовка программы. В модуле вместо зарезервированного слова PROGRAM используется слово UNIT. Здесь же могут присутствовать директивы компилятору, дающие общие установки/соглашения для всего модуля.

При выборе имени модуля необходимо учитывать одну особенность. Выбирая имя программы, мы заботились о том, чтобы оно не совпадало с именами объектов (процедур, функций и т.п.) внутри программы. Здесь к этому требованию добавляется еще и требование совпадения имени модуля с именем файла, в котором он хранится. Поэтому имя модуля не может состоять более чем из восьми символов.

Интерфейсная часть

Новой по отношению к программе является в модуле интерфейсная часть. В этой части описываются все константы, типы данных и переменных, процедуры и функции, доступные в этом модуле для использования внешними программами. Интерфейсная часть модуля несет всю информацию, необходимую для использования процедур и функций, определенных в модуле. Любая другая информация о модуле для обычного его использования не нужна.

Примечание:

В рамках ИИО ТП поставляются следующие стандартные модули *System, Strings, Crt, Printer, dos, WinDos, Graph, Overlay, Turbo3.*

В интерфейсной части возможно, кроме всего прочего, сделать доступными для использования уже существующие готовые модули, указав их имена в операторе USES.

Следом за оператором USES в интерфейсной части описываются доступные извне и необходимые для описанных процедур и функций определения типов данных, констант и переменных.

Все процедуры и функции, доступные для общего пользования и определенные в данном модуле, должны быть описаны в интерфейсной части своей строкой-заголовком с указанием типов параметров. Сам текст программы этих процедур и функций находится (с дубликатом их заголовка) в реализационной части.

Нет необходимости упоминать в интерфейсной части все процедуры и функции, используемые в реализационной части. На практике возможна ситуация, когда интерфейсная часть вообще ничего не содержит, как показано в примере Unitlab.pas,, содержащем модуль Demo_unit3 . Этот модуль, конечно, лишен смысловой нагрузки, но по форме вполне корректен и работоспособен.

```
UNIT Demo_unit3;          { Unitlab.pas}
INTERFACE                 {Эта часть пуста}
IMPLEMENTATION
USES
      Crt ;
BEGIN
ClrScr ;
 END.
```

При написании интерфейсной части некоторого модуля нет необходимости напрямую набирать строки с заголовками процедур и функций, а можно воспользоваться возможностями копирования в рамках редактора текста. Это не только ускорит написание программы, но и снизит вероятность ошибки. При таком подходе обычно используют операции поиска, для того чтобы найти в тексте программы процедуры и функции, имена которых следует указать в интерфейсной части для обеспечения доступа к ним извне.

Реализационная часть

Реализационная часть — это часть, в которой определяются процедуры и функции. Точно так же, как и внутри обычной программы, Вы можете определить здесь глобальные (для модуля) переменные, типы данных и константы наряду с определением процедур и функций. Определенные здесь типы данных и структуры данных недоступны извне и могут использоваться для своих нужд только программами, входящими в реализационную часть.

Так же как и интерфейсная часть, часть реализационная может быть пустой. Такую возможность используют, если необходимо определить некоторые общие для многих программ типы данных, константы и переменные. Имея такое определение в рамках некоторого проекта, легко включить его во все программы (главную и подчиненные) с помощью оператора USES.

Инициализационная часть

Инициализационная часть представляет собой основной блок модуля. Приведенные в ней операторы выполняются первыми, т.е. они выполняются перед операторами основного блока главной программы, в которую включен данный модуль.

Вернемся к нашему *Demo_units-модулю.*. Он содержит в своей инициализационной части оператор **ClrScr,** заключенный в функциональные скобки BEGIN ... END. Если Вы подготовите и запустите следующую программу TUTWENIG.PAS, то вначале произойдет инициализация включенного модуля *Demo_units,* т.е. выполнится оператор **ClrScr,** вследствие чего будет очищен экран. Затем начнет выполняться главная программа, и на экран будет выдано сообщение "Привет!!!".

```
PROGRAM Uses_denio_unit3;  { lab2.PAS}
(Иллюстрирует правило исполнения операторов инициализационной части включенного модуля ПЕРЕД операторами, входящими в главную программу.}
USES
Demo_units;
BEGIN
WriteLn('Привет!!!');
END.
```

Трансляция модуля. Создание выполнимой программы

Результатом трансляции модуля является файл с тем же именем и расширением имени **.TPU,** который ввиду длительного использования заносится на диск, тогда как результат трансляции программы в целом **(.EXE-файл)** может оставаться в основной памяти. При его получении проверяется правильность обращения к блокам модуля (вот для чего нужна интерфейсная часть).

Общий объем модульной программы может быть много более 64, но каждый .TPU-файл не может превышать 64 К.

В подменю Compile, которое показано ниже:

Compile	A11-F9
Make	F9
Build	
Destination	Memory
Primary file ...	

Для запуска трансляции предназначаются верхние 3 пункта, однако пункт Compile требует установки значения "Disk" опции Destination (вместо "Memory") для трансляции модуля; вторым этапом проводится трансляция основной программы. Удобнее пункты Make (F9) и Build; кроме того, можно использовать непосредственно пункт Run ("выполнение") подменю Run или заменяющие его клавиши Ctrl+F9. Во всех 3 случаях программа и модули транслируются совместно, активным должно быть окно основной программ-мы, либо имя файла основной программы должно быть указано в опции Primary file.

Команда Make (клавиша F9) для каждого модуля проверяет:

- существование TPU-файла; если его нет, то TPU-файл создается путем трансляции исходного текста модуля;
- соответствие TPU-файла исходному тексту модуля, куда могли быть внесены изменения; если это так, TPU-файл автоматически создается заново;
- неизменность интерфейсного раздела модуля; если **этот** раздел изменился, то перекомпилируются также все модули, в начале которых данный модуль указан в предложении Uses.

Если модули давно проверены и неизменны, их исходные тексты в ЭВМ не нужны — команда Make в этом случае работает только с TPU-файлами, время компиляции минимально. Напротив, команда Build требует наличия исходных текстов модулей, ибо все они компилируются. Эта "перестраховка" влечет рост времени компиляции. Чтобы были понятны дальнейшие рекомендации, следует четко представить взаимодействие модулей и основной программы.

Изучите рисунок 2 (случай двух модулей M1 и M2; разделы инициализации не предусмотрены, локальные блоки в ней не отражены). В общем случае ссылки модулей на модули могут представлять сложную картину. В нашем примере основная программа косвенно использует модуль M2. Если бы потребовалось обращаться непосредственно в основной программе к объявлениям или блокам модуля M2, мы записали бы в основной программе Uses M1,M2 вместо Uses M1. Примите к сведению, что интерфейсная часть модуля M1 как бы становится началом \ описательной части основной программы, поэтому не только объявления, но и блоки модуля M1 "видны" из подблоков основной программы, где можно обращаться к ним. Однако нельзя, например, определить константу N в основной программе, а использовать в объявлении типа в модуле M1.

```
UNIT M2;
INTERFACE
<0бъявления, которые "видны" лишь из модуля M1>
IMPLEMENTATION
<Блоки, доступные для обращений из модуля M1>
END.

UNIT M1;
INTERFACE
USES M2;
<Объявления, которые "видны" из основной программы>
IMPLEMENTATION
<Блоки, используемые основной программой>
END.
{Основная  программа}
USES M1;
<0бъявления, которые "не видны" из модулей M1 и M2>
<Блоки, недоступные для модулей M1 и M2>
<Главная операторная часть>
END.
```

Рисунок 2 - Взаимодействия модулей и основной программы

Чтобы объявления типов, например структурных, были доступны всем частям программы, их следует поместить в отдельный модуль и его имя указать во всех предложениях Uses. Вас не должно смущать то, что какие-то программы или блоки не используют всех объявлений из его интерфейсного раздела. Хотя раздел реализации отсутствует, слово Implementation нужно записывать как границу интерфейсного раздела. Собрав вместе часто используемые блоки, вы можете оформить еще один модуль. Процесс вычленения модулей можно продолжить и далее. К этому нас могут вынудить и другие обстоятельства. Код основной программы размещается в сегменте памяти , следовательно, он не может занимать более 64 К памяти. Раздел реализации каждого модуля занимает отдельный сегмент. Большую программу вы, безусловно, сделаете блочной. Если не вкладывать большие блоки друг в друга и подчиненные блоки записывать прежде использующих их блоков, довольно просто распределить их для размещения в разные модули: выделив объявления в отдельный модуль, следующие за ними блоки помещаете во 2-й модуль, идущие далее по тексту — в 3-й модуль и тд. При этом 2-й модуль может ссылаться лишь на 1-й, 3-й модуль может обращаться и к 1-му, и ко 2-му и т.д.; тем самым "круговые" ссылки модулей предотвращаются. Поскольку 1-й модуль, содержащий одни объявления, ни на один модуль не ссылается, его можно (и нужно) указать в предложении Uses во всех модулях, где должны действовать эти объявления.

3 Лабораторная работа № 3 «Поиск ошибок с помощью отладчика»

Цель задания: Научиться отлаживать программы с помощью встроен-ного отладчика.

3.1 Ход работы

1) Набрать текст программы из примера lnput_integer
2) Произвести поиск ошибок в программе с помощью отладчика
из меню RUN
подменю Step Over - F8 (без захода в подпрограммы)
подменю Trace Into -F7 (с заходом в подпрограммы)
3) Выполнить программу до некоторой определенной строки Goto Cursor
4) Осуществить перезапуск программы, не закончив её выполнение.

3.2 Содержание отчета

1) Постановка задачи;
2) Текст программы и результат;
3) Нарисовать структуру программного продукта и функционально-модульную структуру программы;
4) Анализ допущенных ошибок.

3.3 Методические указания

Поиск ошибок с помощью отладчика

Несмотря на то что Интегрированная Инструментальная Оболочка Турбо Паскаля включает в себя ряд средств, облетающих разработку программ, в них все равно могут содержаться ошибки, не позволяющие корректно работать с данными программами.

В программе, написанной на ТурбоПаскале, равно как и в программе, написанной на любом другом языке программирования, могут быть допущены ошибки, каждую из которых можно отнести к одному из следующих трех типов:

1) Ошибки, проявляющиеся на этапе компиляции — ошибки, возникающие в связи с нарушением синтаксических правил написания предложений на языке Паскаль (к таким ошибкам в программах относятся пропущенные точки с запятой, ссылки на неописанные переменные, присваивание переменной значений неверного типа и т.д.).

Если компилятор встречает в тексте программы оператор или команду, которую он не может интерпретировать, то он позиционирует курсор на место этого оператора или команды и выводит сообщение об ошибке.

Если подобные ошибки возникают при использовании командно-строчного компилятора, то он выдает сообщение об ошибке, выводит исходную строку с ошибкой и ее номер, при этом символ (^) в выводимой исходной строке указывает местоположение ошибки.

2) Ошибки, проявляющиеся на этапе выполнения — ошибки, возникающие в связи с нарушением семантических правил написания программ на языке программирования Турбо Паскаль 7.0 (ярким примером ошибки данного типа является ситуация, когда Ваша программа пытается выполнить деление на нуль). Если написанная Вами программа обнаруживает ошибку такого типа, то она завершает свое выполнение и выводит сообщение следующего вида:

Run-time error <nnn> at **<xxxx:yyyy>**

Где:

nnn —номер ошибки выполнения,

xxxx:yyyy — адрес ошибки выполнения.

Если Вы выполняете программу из ИИО ТП, то она автоматически найдет вызвавший семантическую ошибку оператор (так же как и в случае синтаксических ошибок). Если же программа выполнялась вне ИИО и в ней появилась ошибка данного типа, то необходимо запустить ИИО ТП и найти вызвавший семантическую ошибку оператор, используя команду **Search/Find Error** (после того как Вы активизируете команду **Search/Find Error,** она запросит у Вас адрес сегмента и смещения (xxxx:yyyy) оператора, вызвавшего семантическую ошибку).

3) Логические ошибки — это ошибки, связанные с неправильным применением тех или иных алгоритмических конструкций. ИИО ТП 7.0 не позволяет автоматически обнаруживать ошибки данного типа. Она лишь обладает рядом средств отладки, которые могут значительно облегчить процесс поиска таких ошибок.

С точки зрения программиста одно из основных преимуществ отладки программ с помощью встроенного в ИИО ТП отладчика (Debugger) заключается в возможности осуществления контроля над ходом выполнения программы. Отследив таким образом выполнение каждой инструкции, Вы без особого труда сможете определить, в какой части программы находится ошибка.

Встроенный в ИИО ТП отладчик позволяет:

- выполнять команды языка Паскаль построчно (пошагово);
- трассировать команды языка Паскаль
- выполнять программу до некоторой определенной строки (до некоторой точки);
- осуществлять перезапуск программы, не закончив ее выполнение.

Эти возможности позволяют контролировать ход выполнения программы.

Рассмотрим все эти управляющие средства по очереди.

Пошаговая отладка программы и трассировка

Команды **Step Over и Trace Into** меню **Rim** позволяют осуществить построчную отладку программы. Активизация команды **Step Over** или нажатие функциональной клавиши [F8], так же как активизация команды **Trace Into** или нажатие функциональной клавиши [F7], вызывает выполнение отладчиком всех операторов, расположенных в строке, помеченной маркером (указателем строки выполнения). Единственное отличие между выполнением программы по шагам и ее трассировкой состоит в том, как реагирует отладчик на появление в тексте программы операторов вызова процедур или функций. При пошаговом выполнении программы вызов процедуры или функции интерпретируется как вызов простого оператора, т.е. Вы можете увидеть результаты работы процедуры, но не можете пошагово проследить, каким образом этот результат был получен, в то время как при трассировке программы такая возможность предоставляется.

Рассмотрим более подробно работу команд **Step Over** и **Trace Into** на следующем примере

```
PROGRAM Procedure_and_Function;        {FUNKPROZ.PAS}
USES Crt;
VAR   a, b, Sum_nuinbers: INTEGER;
PROCEDURE Summing-up (VAR Sum:INTEGER; a,b:INTEGER);
BEGIN     Sum := a + b;      END;
FUNCTION Sum(a,b: INTEGER):INTEGERS;
BEGIN   Sum := a + b;      END;
BEGIN
    ClrScr;
       a := 12;       b := 15;
Summing_up (Sum_number3, a, b);
        WriteLn('Сумма равна: ',Sum_numbers);
 Sum_number3:=Sum(a,b);
      WriteLn('Сумма равна: ',Sum_numbers);
          WriteLn('Сумма равна: ',Sum(a,b)) ;
END.
```

Примечание:

Прежде чем проводить пошаговую отладку программы или ее трассировку, необходимо убедиться, что опция **Options/Debugger/Integrated** активизирована.

Введите с клавиатуры и откомпилируйте текст данной программы, после чего нажмите клавишу **[F8]** (пошаговая отладка). Обратите внимание на тот факт, что маркер сразу перемещается на оператор **BEGIN**, расположенный в основном блоке программы. Это происходит потому, что оператор **BEGIN** является первым оператором, который выполняется в теле программы. Нажимая клавишу **[F8]**, выполните эту программу до конца. Теперь вы можете попытаться оттрассировать эту же программу, нажимая клавишу **[F7]**. Обратите внимание на то, что трассировка программы во многом аналогична ее выполнению по шагам. Единственное отличие, как мы уже говорили, состоит в том, что когда встречается оператор вызова процедуры или функции, то при трассировке эти процедуры и

функции также выполняются по шагам, а при простом выполнении по шагам управление возвращается к Вам только после завершения выполнения подпрограммы.

4) Выполнение программы до определенной точки.

Иногда в процессе отладки возникает необходимость пошаговой отладки не всей программы, а лишь ее части. В ТП проблема выполнения программы до определенной точки может быть решена одним из следующих способов.

Во-первых, Вы можете воспользоваться командой **Go to Cursor** меню **Run** (или функциональной клавишей [F4]), предварительно установив курсор на так называемую строку останова (т.е. строку, до которой вы хотите выполнить программу). После активизации команды **Go to Cursor** программа будет выполняться до тех пор, пока не достигнет строки останова, в которой ее выполнение будет приостановлено. Начиная с этого момента, у Вас появится возможность управлять ходом событий. Например Вы можете продолжить выполнять данную программу пошагово до некоторой новой точки или перезапустить программу (о том, как перезапустить программу, не закончив ее выполнение, Вы можете прочитать в следующем разделе).

Во-вторых, Вы можете установить в некоторой строке (или сразу в нескольких строках) так называемую точку останова **(Breakpoint).** Запущенная программа будет выполняться до тех пор, пока не достигнет строки, в которой установлена точка останова.

После чего у Вас, как и в предыдущем случае, появится возможность управлять дальнейшим ходом событий.

Установить в тексте программы **Breakpoint** можно любым из ниже перечисленных способов:

а) Установите курсор на строке, в которую Вы хотите поместить **Breakpoint.** Откройте локальное меню и активизируйте команду **Toggle breakpoint;**

б) Установите курсор на строке, в которую Вы хотите поместить **Breakpoint,** и активизируйте клавиатурную комбинацию [Ctrl+F8];

в) Установите курсор на строке, в которую Вы хотите поместить **Breakpoint,** и активизируйте команду **Add breakpoints** меню **Debug.**

Если Вы активизируете команду **Breakpoints** меню **Debug,** перед Вами откроется окно **Breakpoints,** в котором содержится список всех использованных в программе точек останова. В этом окне для каждой из них имеется следующая информация:

- Breakpoint list: Имя файла, в котором установлена точка останова;

- Line: Номер строки, в которой установлена точка останова;

- Condition: Условие, при выполнении которого осуществляется останов программы. Например в качестве условия может быть использовано выражение следующего типа:

X > O или (X <=) OR (KeyPresaed)

- Pass: Счетчик числа проходов.

Задание для точки останова счетчика проходов сообщает отладчику, что останавливать программу нужно не при каждом достижении точки останова, а

только при N-ом ее проходе. Например **Breakpoint,** установленный в теле цикла **FOR ... TO ... DO**

В окне **Breakpoints** пользователь может модифицировать (кнопка **Edit),** удалить (кнопка **Delete** или **Clear all** — если Вы хотите удалить все точки останова) или просмотреть (кнопка **View)** местоположение какой-либо точки останова.

Примечание:

Чтобы удалить точку останова, установите курсор на содержащую ее строку и активизируйте комбинацию клавиш [Ctrl+F8].

3.4 Пример учебной программы lnput_integer

Постановка задачи:

Набрать предложенную программу Input_fixed_point_nuinber 2 и модуль UNIT My_units. Произвести поиск ошибок в программе с помощью отладчика из меню RUN. Сохранить программу Input_fixed_point_nuinber 2 в файле под именем INPINTEG.PAS, а модуль My_units в файле под имеем UEINGABE.PAS

Ранее мы уже установили, что попытка задать для целого числа значения, выходящие за допустимые пределы, приводит к ошибкам.

Для предупреждения таковых необходимо определить некоторую функцию контроля. Чтобы ее реализовать, перейдем к вводу действительных чисел вместо целых (диапазон их допустимых значений существенно шире). Введенная, как и раньше, строка ***Cbaracter_string*** теперь будет рассматриваться как запись действительного числа и будет переводиться в действительное число с помощью процедуры **VaL.** Далее мы сможем проверить, попадает ли значение этого действительного числа в диапазон допустимых значений целых чисел. И если это так, то преобразуем полученное значение в целое.

С помощью стандартной процедуры **Frac** можно установить, имеет ли действительное число значащие цифры после запятой. Если проверка даст положительный результат, то такое число, очевидно, не является целым и должно быть отвергнуто. (В то же время числа, имеющие цифры после запятой, но незначащие, как например 3.0 или 2.000, вполне могут быть преобразованы в целые). Проверенные и подходящие действительные числа могут быть преобразованы в целые с помощью функции **Round.**

В дополнение ко всему вышесказанному мы реализуем вывод сообщений об ошибках с помощью специальной процедуры ***Error,*** объявленной в создаваемом модуле. Место на экране, в которое эта процедура будет выдавать сообщения, определяется двумя глобальными константами ***Positioa_Xw. Positioa_Y,*** объявленными в модуле. Кроме того, нам понадобится процедура удаления с экрана сообще-ния об ошибке ***CIrError,*** которая пользуется теми же константами, что и *Error.*

Модуль My_units и его использование

Наряду с уже обсуждавшимися процедурами и функциями модуль ***My_units*** содержит ряд функций, ранее не упоминавшихся. Краткое описание каждой новой процедуры (как и "старой") приведено в комментариях интерфейсной части модуля или в комментариях в начале текста самой процедуры. Если же в тексте программы Вам встретится незнакомый оператор, то информацию о нем можно получить с помощью встроенной справочной подсистемы.

```
UNIT My_units;
{UEINGABE.PAS}
{$V-}
INTERFACE
TYPE  Character_string_type  =  STRING[60];
PROCEDURE CIrError;
        {Удаляет с экрана выведенное ранее сообщение об ошибке.}
PROCEDURE Error(String_variable: Character_string_type);
{Выдает сообщение об ошибке, передаваемое в качестве параметра, на экран.}
FUNCTION Input_integer_variable : INTEGER;
        {Вводит целое число. Ввод осуществляется в формате действительного
числа с целью проверки принадлежности введенного числа диапазону значе-ний,
допустимых для целых чисел. При неудовлетворительном результате та-кой
проверки введенное значение отвергается.}
PROCEDURE GetStr(VAR Character_set: STRING);
        {Ввод строки (альтернатива процедуре ReadLn)}
PROCEDURE Input_code_key(VAR CH:CHAR);  {Ввод кода клавиши}
        IMPLEMENTATION
        USES Crt;
        CONST   Coiniaand_character = ^G:     {Управляющий символ. Оператор}
                                              {Write(^G) подает звуковой сигнал}
             Position_X = 1;                  {Позиция по оси X и по оси Y, в}
             Position_Y = 25;          {которую выводится сообщение об ошибке}
        VAR  Invalid letter : INTEGER;
PROCEDURE Error(String_variable:haracter_string_type);
{Выводит сообщение об ошибке, передаваемое в качестве параметра, на экран.)
        BEGIN
              GotoXY(Position_X,Poaition_Y) ;
{Выдача сообщения об ошибке производится в нижнюю  строку экрана и
сопровождается звуковым сигналом.}
             Write('Ошибка:',"G,String_variable) ;
             CIrEol;                    {Очистка неиспользуемой части строки}
        END;
PROCEDURE CIrError;{Удаляет с экрана выведенное ранее сообщение об
ошибке.}
         BEGIN
              GotoXY(Position_X,Position_Y) ;
```

```
            CIrEol;
          END;

FUNCTION Input_integer_variable: INTEGER;
{Вводит целое число. Ввод осуществляется в формате действительного числа с
целью проверки принадлежности введенного числа диапазону значений, допус-
тимых для целых чисел. При неудовлетворительном результате такой проверки
введенное значение отвергается.}
            VAR
               Real_variable :REAL;
               Character_string :STRING;
               X_Position, Y_Position :BYTE;
PROCEDURE Overflow;
            BEGIN
               Error('Введенная величина лежит вне допустимых', пределов!!');
             Invalid_letter := 1;
  {Код сшибки устанавливается в произвольное значение > 0.}
          END;
PROCEDURE Control_letter;
 {Реакция на наличие значащих цифр после запятой или недопустимых символов
во введенной строке.}
          BEGIN
          Error('Введите целое число!!');
             Invalid_letter := 1;
          END;
   PROCEDURE Control_limit;
          BEGIN
             GotoXY(X_Position,Y_Po3ition) ;
          CIrEol;
              ReadLn(Character_String);
          Val (Character_string,Real_variable,Invalid_letter) ;
              WriteLn;
IF ((Real_variable <-32768.5) OR (Real_variable >= 32768.5)) THEN Overflow
              ELSE
 IF (Frac(Real_variable)oO) OR (Invalid_letter > 0)  THEN Control_letter
              ELSE CIrError;
          END;                          {Control_limit}
          BEGIN                 {Input_integer_variable}
           X_Position := WhereX;
           Y_Position := Wher
{Сохранение координат положения курсора. Это дает в  дальнейшем возмож-ность
вернуть курсор в позицию, в которой он находился на момент вызова функции}
           REPEAT Control_limit
           UNTIL Invalid_letter = 0;
 {Корректная величина преобразуется в целое число путем округления.}
```

```
        Input_integer_variable := Round(Real_variable);
        CIrError;                          {Удаление ранее выданного сообщения}
        GotoXY(l,Y_Position + 1);     {Переход к началу следующей строки}
        END;                               {Input_integer_variable}
PROCEDURE GetStr (VAR Character_set: STRING);
                           {Ввод строки (альтернатива процедуре ReadLn).}
        VAR  i, line, column : INTEGER;
        BEGIN
            Character_set[l] := # 0;
        IF Line > 24  THEN line := 24;
        Column := WhereX;
        REPEAT
              GotoXY(column,line) ;
              ReadLn(Character_set) ;
        UNTIL (Character_set[l] <> #0);
        END;
        PROCEDURE Input_code_key (VAR CH:CHAR);
        BEGIN
              REPEAT CH :=#2S5;                              {Начальная установка}
        IF KeyPressed                                  {Проверка нажатия клавиши}
        THEN BEGIN
           CH := ReadKey;          {Ввод кода символа нажатой клавиши из буфера,}
IF CH = #OTHEN CH := ReadKey;        {приводящий к сбросу флага  KeyPressed.}
                  END;
              UNTIL CH <> #255
    END;
  END.
```

Чтобы окончательно оформить эту программу как модуль, необходимо Сохранить объектный модуль данной программы на диске. Для опции **Compile/Destination** установим значение **Disk** и выполним операцию компиляции с помощью комбинации клавиш [Ctrl+F9]. ТП 7.0 распознает в начале текста модуля оператор UNIT и автоматически создает файл с расширением .TPU вместо .EXE (как для обычных программ). Выдаваемое при этом сообщение **Can not run a unit** просто информирует Вас о том, что модуль самостоятельно не выполняется.

Для демонстрации возможностей (и отладки) модуля *My_units* и функции *Iaput_iateger_variable* Вам потребуется еще одна программа, и если Вы не придумаете ничего лучше, то используйте приведенный вариант программы INPINTEG.PAS.

```
        PROGRAM Input_fixed_point_nuinber 2;
        {INPINTEG.PAS}
(Обратите внимание на отличия данного примера от INPINTEG.PAS)
        USES Crt,   My_unita;
        VAR   Integer_variable : INTEGER;
```

```
BEGIN
        CIrScr;
GotoXY(l,5);   WriteLn('Введите, пожалуйста, целое число! ' );
GotoXY(2,7); {Курсор позиционируется в точку экрана, предназначенную для
ввода чисел.}
Integer_variable : = Input_integer_variable;
GotoXY(l,9);  WriteLn(Integer_variable) ;
{Курсор позиционируется в точку экрана,  предназначенную для  вывода чисел.}
        END.
```

3.5 Варианты индивидуальных заданий к зачету по теме «Модули»

1) Реализовать в виде модуля набор подпрограмм для выполнения следующих операций над комплексными числами:

а) сложения;

б) вычитания;

в) умножения;

г) деления;

д) модуля комплексного числа;

ж) возведения комплексного числа в степень *n (n* — натуральное).

Комплексное число представить следующим типом:

```
Type Complex=Record
R: Real;
M: Real
End;
```

Используя этот модуль, решить следующие задачи 1.1 и 1 2.

1.1 Дан массив *A* — массив комплексных чисел. Получить массив С, элементами которого будут модули сумм рядом стоящих комплексных чисел.

1.2 Дан массив *A [M]* — массив комплексных чисел. Получить матрицу *B[N, M],* каждая строка которой получается возведением в степень, равную номеру этой строки, данного массива *A*.

2) Реализовать в виде модуля набор подпрограмм для выполнения следующих операций над обыкновенными дробями вида P /Q *(P*-целое, *Q* - натуральное):

а) сложения;

б) вычитания;

в) умножения;

г) деления;

д) сокращения дроби;

ж) возведения дроби в степень *n (n*- натуральное);

з) функций, реализующих операции отношения (равно, неравно, больше или равно, меньше или равно, больше, меньше).

Дробь представить следующим типом:

Type Frac=Record P: Integer; Q: 1..32767 End;

Используя этот модуль, решить задачи 2.1 и 2. 2.

2.1 Дан массив *A* — массив обыкновенных дробей. Найти сумму всех дробей, результат представить в виде несократимой дроби. Вычислить среднее арифметическое всех дробей, результат пред ставить в виде несократимой дроби.

2.2 Дан массив *A* — массив обыкновенных дробей. Отсортировать его в порядке возрастания.

3) Реализовать в виде модуля набор подпрограмм для выполнения следующих операций с квадратными матрицами:

а) сложения двух матриц;

б) умножения одной матрицы на другую;

в) нахождения транспонированной матрицы;

г) вычисления определителя матрицы.

Матрицу описать следующим образом:

Const NMax=10;

Type Matrica=Array[1.-NMax,1..Nmax] Of Real;

Используя этот модуль, решить следующие задачи 3.1 и 3.2.

3.1 Решить систему линейных уравнений /V-го порядка ($2 < N < 10$) методом Крамера.

3.2 Задан массив величин типа Matrica. Отсортировать этот массив в порядке возрастания значений определителей матриц.

4) Реализовать в виде модуля набор подпрограмм для выполнения следующих операций над векторами:

а) сложения;

б) вычитания;

в) скалярного умножения векторов;

г) умножения вектора на число;

д) нахождения длины вектора.

Вектор представить следующим типом:

Type Vector=Record

X,Y: Real

End;

Используя этот модуль, решить задачи 4.1,и 4. 2.

4.1 Дан массив *A* — массив векторов. Отсортировать его в порядке убывания длин векторов.

4.2 С помощью датчика случайных чисел сгенерировать 2N целых чисел. N пар этих чисел задают N точек координатной плоскости. Вывести номера тройки

точек, которые являются координатами вершин треугольника с наибольшим углом.

5) Реализовать в виде модуля набор подпрограмм для выполнения следующих операций над натуральными числами в Р-ичной системе счисления ($2 < P < 9$):

а) сложения;

б) вычитания;

в) умножения;

г) деления;

ж) перевода из десятичной системы счисления в Р-ичную;

з) перевода из Р-ичной системы счисления в десятичную;

и) функции проверки правильности записи числа в Р-ичной системе счисления;

к) функций, реализующих операции отношения (равно, не равно, больше или равно, меньше или равно, больше, меньше).

Р-ичное число представить следующим типом:

Type Chislo=Array [1. .16] Of 0..9;

Используя этот модуль, решить задачи 5.1 и 5. 2.

5.1 Возвести число в степень (основание и показатель степени записаны в Р-ичной системе счисления). Результат выдать в *Р*-ичной и десятичной системах счисления.

5.2 Дан массив *А* — массив чисел, записанных в Р-ичной системе счисления. Отсортировать его в порядке убывания. Результат выдать в Р-ичной и десятичной системах счисления.

6) Реализовать в виде модуля набор подпрограмм для выполнения следующих операций над натуральными числами в шестнадцатеричной системе счисления:

а) сложения;

б) вычитания;

в) умножения;

г) деления;

д) перевода из двоичной системы счисления в шестнадцатеричную;

е) перевода из шестнадцатеричной системы счисления в десятичную;

ж) функции проверки правильности записи числа в шестнадцатеричной системе счисления;

и) функций, реализующих операции отношения (равно, не равно, больше или равно, меньше или равно, больше, меньше).

Используя этот модуль, решить следующие задачи 6.1 и 6.2.

6.1 Возвести число в степень (основание и показатель степени записаны в шестнадцатеричной системе счисления). Результат выдать в шестнадцатеричной и десятичной системах счисления.

6.2 Дан массив *A* — массив чисел, записанных в шестнадцатеричной системе счисления. Отсортировать его в порядке убывания. Результат выдать в шестнадцатеричной и десятичной системах счисления.

7) Определим граф как набор точек, некоторые из которых соединены отрезками, подграф — как граф, являющийся под множеством данного графа.

Реализовать в виде модуля набор подпрограмм, определяющих:

а) число точек в графе;

б) число отрезков в графе;

в) число изолированных подграфов в графе (подграфов, не соединенных отрезками);

г) диаметр графа — длину максимальной незамкнутой линии в графе (длина каждого звена — 1);

д) граф — объединение двух графов;

е) подграф — пересечение двух графов;

ж) подграф — дополнение данного графа до полного (графа с тем же количеством вершин, что и в данном, и с линиями между любыми двумя вершинами);

и) число отрезков, выходящих из каждой вершины графа.

При запуске должны инициализироваться переменные:

Full_Graph — полный граф с числом вершин NumberOfVertix, Null_Graph — граф без отрезков с числом вершин NumberOfVertix.

Граф представить как объект

```
Const  NumberOfVertix=50;
Type   Graph=Array[1..NumberOfVertix,
1..NumberOfVertix]   Of   Boolean;
```

Используя модуль, решить следующую задачу 7.1.

7.1 Найти все правильные графы из *N* вершин (граф правилен, если из всех вершин выходит равное количество отрезков).

8)Реализовать в виде модуля набор подпрограмм для выполнения следующих операций над длинными числами:

а) сложения;

б) вычитания;

в) умножения;

г) нахождения частного и остатка от деления одного числа на другое;

д)функций, реализующих операции отношения (равно, не равно, больше или равно, меньше или равно, больше, меньше).

Длинное число представить следующим типом:

```
Type   Tsifra=Q..9;
Chislo=Array[l..1000]   Of  Tsifra;
```

Используя этот модуль, решить задачи 8.1, 8.2.

8.1 Возвести число в степень (основание и показатель степени — длинные числа).

8.2 Дан массив длинных чисел. Упорядочить этот массив в порядке убывания.

9) Реализовать в виде модуля набор подпрограмм для выполнения операций с многочленами от одной переменной (первый многочлен степени *m,* второй — степени *n):*

а) сложения;

б) вычитания;

в) умножения;

г) деления с остатком;

д) операций отношения (равно, не равно);

ж) возведения в натуральную степень *к;*

з) вычисления производной от многочлена;

е) вычисления значения в точке x0.

Многочлен представить следующим типом:

Type Mnogochlen=Array[1..500] Of Integer;

Используя этот модуль, решить задачи 9.1 и 9. 2.

9.1 Найти наибольший общий делитель многочленов *P(x)* и *Q(x).*

9.2 Вычислить *Ps(x) — Qr(x).*

10) Разработать способ представления множеств, содержащих более 255 элементов (до 2000). Создать модуль, позволяющий выполнять следующие операции над элементами таких множеств:

а) объединение;

б) пересечение;

в) разность;

г) функция проверки принадлежности элемента множеству;

д) функция проверки, является ли данное множество подмножеством (надмножеством) другого.

Используя созданный модуль, решить следующие задачи 10.1.

10.1 Дан массив множеств. Упорядочить элементы массива в порядке возрастания количества компонент соответствующих множеств.

Разработать программу, которая вводит несколько множеств, выражение, операндами которого являются эти множества, с операциями объединения, пересечения и вычитания, вычисляет значение этого выражения и выводит результат.

3.6 Вопросы к защите лабораторной работы

1) Какие файлы сохраняются на диске после трансляции программы;
2) Что позволяет выполнить встроенный в TP отладчик;
3) Перечислить способы выполнения программы до определенной точки.

4 Лабораторная работа 4 «Долговременно хранимые и стандартные модули»

Цель работы: Научиться использовать стандартные модули Турбо Паскаля

4.1 Ход работы

1) Ознакомиться с работой стандартного модуля DOS (Windos), позволяющего датировать последовательные версии файла, распечатки программ, измерять время выполнения её частей;
2) Составить и отладить программу использования стандартных процедур: GetTime - дающую текущее время: час, минуты, секунды, доли секунды;
GetDate - дающие текущую дату, месяц, год, день недели;
3) Измерить время выполнения «бессодержательного» цикла.

4.2 Содержание отчета

1) Подготовка задачи;
2) Исходные данные;
3) Тексты программ и результаты их выполнения;
4) Анализ допущенных ошибок.

4.3 Вопросы для допуска к лабораторной работе

1) Какие стандартные модули имеются в Турбо Паскале;
2) Какие из них необязательно объявлять в предложении Uses;
3) Какие из стандартных модулей содержатся в файле Turbo.tpl.

4.4 Методические указания

Стандартный модуль System - единственный, который не нужно указывать в предложении Uses -содержит "библиотеку времени выполнения" (стандартные математические функции и т.п.). Имеются следующие стандартные модули: System, dos, Crt, Overlay, Printer, Graph, Turbo3 и Graph3. Они, кроме 3 последних, содержатся в файле Turbo.TPL ("библиотека модулей"), а Graph.TPU — в директории BGI. Тигbo3 и Graph3 даны для совместимости с ранней версией Турбо Паскаля.

Модуль DOS содержит блоки, обеспечивающие доступ ко всем средствам операционной системы PC DOS. Блоки "даты-времени", позволяющие датировать последовательные версии файла, распечатки программы, измерять время выполнения ее частей.

Модули Strings (библиотека блоков для работы со строками типа PChar) и

WinDos — с целью реализации возможностей системы MS DOS с использованием строк типа PChar. Модуль WinDos используется взамен модуля dos.

Пример. Процедура GetTime дает текущее время: час, минуты, секунды, доли секунды. Измерим время выполнения "бессодержательного" цикла:

```
……
Uses dos;
  Var
      hour, min, sec, dec: Word;
      j, t1: longint;
Begin
      GetTime(hour min,sec,dec);
          t1:= hour*3600 + min*60 + sec;
             For j:= 1 to 10000000 do;        {"Пустой" цикл}
        GetTime(hour,min,sec,dec);
        Writeln(h6ur*3600 + min*60 + sec - t1)
 End.
```

Библиотечный файл TURBO.TPL можно расширять. Длительно и коллективно используемые библиотеки блоков лучше хранить не в виде TPU-файлов, а "встроить" в систему путем помещения их в файл Turbo.TPL. Вам поможет это сделать утилита (вспомогательная системная программа) TPUMOVER. В результате упрощается использование библиотечных блоков.

Следует иметь в виду, что файл Turbo.TPL автоматически загружается компилятором в оперативную память и чрезмерное его увеличение нежелательно.

5 Лабораторная работа № 5 «Создание оверлейных модулей»

Цель работы: Научиться создавать и отлаживать оверлейные модули

5.1 Ход работы:

1) Набрать и отладить пример оверлейной программы, состоящей из главной программы и двух оверлейных модулей;

2) Оверлейная программа должна быть реализована как управляющая (резидентная) часть, которая постоянно будет находиться в памяти;

3) Оверлейные фрагменты должны быть оформлены в виде *модулей,* снабженных специальной директивой {$O+};

4) При трансляции оверлейной программы Turbo Pascal – компилятор работает следующим образом - управляющая (неоверлейная) часть (коды модулей объединяются в файл с тем же самым именем, но с расширением .ovr);

5) Основная часть оверлейной программы должна содержать описатель Uses, в котором перечисляются используемые оверлейные модули. Первым модулем в описании должен быть указан системный модуль Overlay. Кроме того, вслед за этим описанием должны размещаться директивы компилятора, указывающие, какие модули из перечисленных в описании Uses являются оверлейными;

6) Распечатать программу и модули.

5.2 Содержание отчета:

1) Постановка задачи;

2) Текст программы и результат;

3) Нарисовать структуру программного продукта и функционально-модульную структуру программы;

4) Сделать анализ ошибок допущенных при работе.

5.3 Методические указания

Использование языка Turbo Pascal для программирования реальных задач в ряде случаев осложняется ограничениями, связанными с размещением больших программ в оперативной памяти. В общем случае без использования специальных Средств максимальный размер программы не может превышать объем свободной оперативной памяти.

В целях преодоления этого естественного ограничения в системе Turbo Pascal имеется специальный механизм, который называется *оверлейным.*

Применение оверлейного механизма позволяет разрабатывать большие и сложные программы, размер которых значительно превышает объем оперативной памяти.

Базовым принципом оверлейного механизма является представление программы в виде совокупности фрагментов, которые попеременно занимают одну и ту же область оперативной памяти. При необходимости выполнения того или иного фрагмента он загружается в оперативную память, быть может, вытесняя из нее ранее выполнявшийся фрагмент.

Оверлейный механизм является известным и наиболее распространенным методом преодоления ограничений по оперативной памяти в отсутствие у операционной системы средств виртуализации памяти. Кроме системы Turbo Pascal, развитый оверлейный механизм используется, например, при формировании объектного кода в системе Clipper. Как правило, алгоритм размещения и уда-ления оверлейных фрагментом реализуется специальным системным монитором, который поменяется в код разрабатываемой программы.

Естественно, оверлейная программа будет выполняться медленнее, чем выполняется (или могла бы выполняться) эквивалентная ей обычная программа, за счет времени подкачки оверлейных фрагментов. Однако это замедление является необходимой платой за выигрыш в общем размере программы.

Средства построения оверлейных структур в языке Turbo Pascal достаточно просты для использования (сохраняя в этом отношении общий дух ясности и компактности языка Pascal), и базируются на следующих основных принципах.

1) Оверлейная программа должна быть реализована как управляющая (резидентная) часть, которая постоянно будет находиться в памяти, и несколько оверлейных фрагментов, коды которых будут попеременно загружаться в специальный оверлейный буфер в оперативной памяти, который автоматически выделяется между сегментом стека и динамической областью памяти. По умолчанию для оверлейного буфера выбирается минимальный возможный размер, но во время выполнения программы его размер может быть легко увеличен путем выделения дополнительной области;

2) Оверлейные фрагменты должны быть оформлены в виде МОДУЛЕЙ, снабженных специальной директивой ($0+);

3) Никаких дополнительных языковых конструкций для оверлейного механизма не предусматривается; все средства управления оверлеями сосредоточены в системном модуле Overlay;

4) При трансляции оверлейной программы Turbo Pascal-компилятор работает следующим образом: управляющая (неоверлейная) часть программы оформляется в виде EXE-файла; все оверлейные части (то есть коды модулей) объединяются в файл с тем же самым именем, но с расширением.ovr. Трансляции оверлейных модулей представлена на рисунок 3;

5) Все обращения к оверлейным процедурам и функциям должны осуществляться посредством дальнего типа вызовов. Для этого такие подпрограммы (или оверлейные модули в целом), а также основную программу необходимо компилировать с включенной директивой $F или добавив после заголовка подпрограмм служебное слово tar;

6) Основная часть оверлейной программы должна, естественно, содержать описатель uses, в котором перечисляются используемые оверлейные модули (в этом же описании могут быть указаны и неоверлейные модули, используемые программой). Первым модулем в описании должен быть указан системный модуль Overlay. Кроме того, вслед за этим описанием должны размещаться директивы компилятора, указывающие, какие модули из перечисленных в описании uses являются оверлейными. Каждая такая директива имеет вид:

($0 Имя_модуля)

Имя_модуля в этой директиве должно идентифицировать дисковый файл с кодом оверлейного модуля (tpu-файла). Допускается указание дисковода и/или цепочки объемлющих каталогов. Расширение имени (tpu) может быть опущено.

Заметим, что из всех системных модулей Turbo Pascal в качестве оверлейного может быть использован ТОЛЬКО модуль dos.

7) В теле главной программы перед первым обращением к какой-либо оверлейной подпрограмме должен быть вызов стандартной процедуры Ovrlnit из модуля Overlay. Эта процедура инициализирует подсистему управления оверлеями; единственным ее параметром является строка с именем файла, в котором собраны коды оверлейных модулей (ovr-файл).

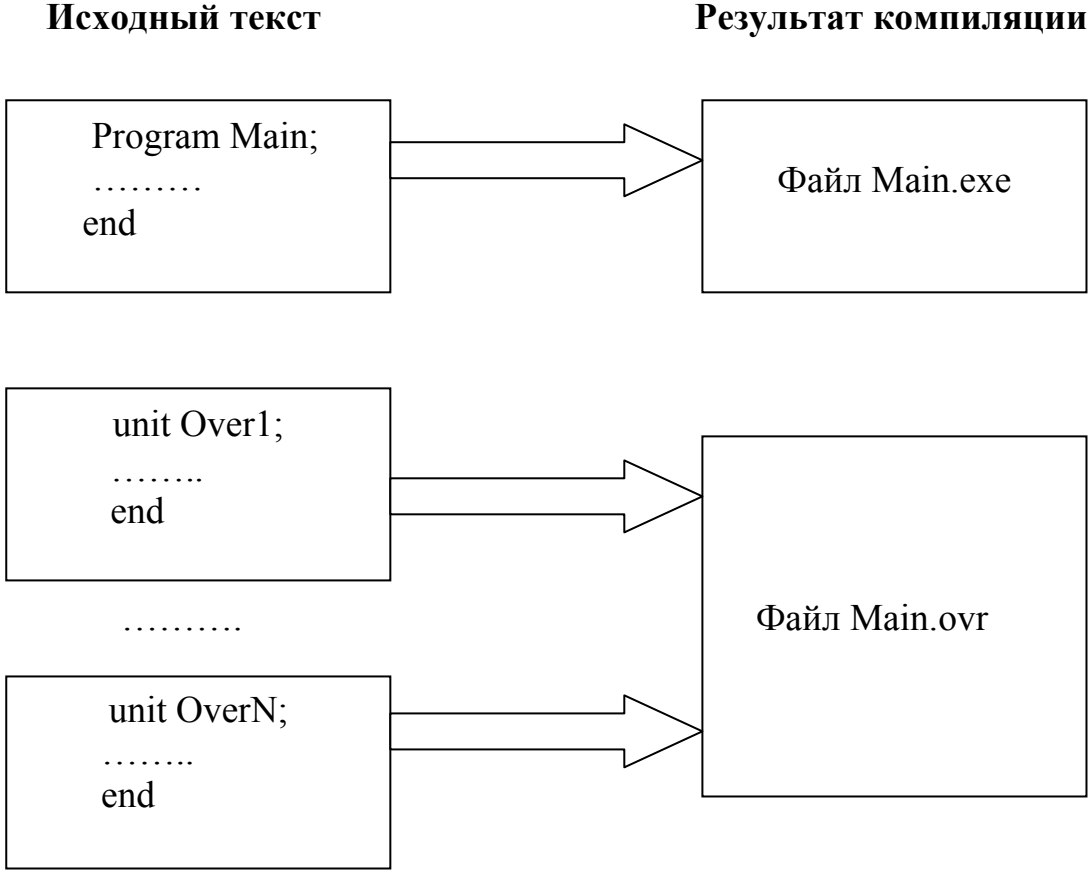


Рисунок 3 - Трансляции оверлейных модулей

Ниже приведен простой пример оверлейной программы, состоящей из главной программы и двух оверлейных модулей. Этот пример входит в группу демонстрационных файлов стандартной поставки системы Turbo Pascal.

```
{$F+,0+}                                    { директивы $F, $0 включены }
program OvrDemo;                  { главная часть оверлейной  программы}
{В uses-списке  должен быть указан модуль Overlay, а также оверлейные модули; указание модуля Crt показывает, что в списке должны быть все используемые в программе модули}
Uses Overlay, Crt, OvrDemol, OvrDemo2;
{$O OvrDemo1}                          {Указание оверлейных модулей из}
{$O  OvrDevo2}                         {числа  указанных  в  uses-списке}
begin
  TextAttr := White;              { Используем средства модуля Crt }
CIrScr;
Ovrlnit(OVRDEMO.OVR); {Инициализация системы управления оверлеями}
if OvrResult  <> 0 then  begin
                    Writeln('Ошибка: ', OvrResult) ;
                        Halt(l)
                        end;
{Коды процедур Write 1 и Write2, расположенные в оверлейных модулях, будут при выполнении программы попеременно загружаться в оверлейный буфер} показаны на рисунке 4.
repeat
      Writel;
      Write2
 Until KeyPressed
 end.
```

Файл OVRDEM01.PAS

```
($0+,F+}
  unit OvrDemol;
interface
      procedure Writel;
 implementation
   procedure Write1;
   begin
     Writeln('Один...')
    end;

end.
```

Файл OVRDEM02.PAS

```
($0+,F+)
  unit OvrDemo2;
interface
      procedure Write2;
 implementation
   procedure Write2;
    begin
      Writeln('Два...')
    end;

end.
```

Рисунок 4 - Оверлейные модули

5.4 Индивидуальное задание к зачету по теме «Оверлейные модули»

Составить программу для расчета треугольника при разных комбинациях исходных данных.

Алгоритм составления программы:

1) Сделать 2 оверлея;

- в оверлей TPAB поместим блоки процедуры TRa, TRb и функцию R;
- в оверлей TRCD поместим процедуры TRc, TRd.. Блоки из оверлея TRCD обращаются к блоку TRa в оверлее TRAB. Учтем это.

2) Откроем окно редактора и запишем в него оверлей.TRAB;

3) Поместим текст оверлея TRAB в файл TRAB.PAS. Данное окно редактора можно закрыть.

4) Откроем окно для записи оверлея TRCD:

5) Поместим текст оверлея TRCD в файл TRCD.PAS, окно закроем.

Откроем новое окно, занесем в него текст основной программы, сохраним этот текст в файле Tre.PAS (если вы измените имя этого файла, то должны изменить также имя файла в обращении к OvrInit):

5.5 Вопросы к защите лабораторной работы

1) Что является базовыми принципами оверлейного механизма представления программы?

2) Что содержит основная часть оверлейной программы?

3) Недостаток работы оверлейных модулей;

4) Схема распределения памяти в оверлейных программах.

6 Лабораторная работа № 6 «Динамическая память»

Цель работы: Карта памяти – Куча.

6.1 Ход работы:

1) Отладить программу которая выводит Сегмент (часть адреса) – начало ее кода, начало данных и Кучи. Сегмент границы свободной части кучи и стека.

1.1) Запустить программу из TP и выйти из него.

1.2) По полученным данным построить карту памяти.

2) Отладить программу с определением Const x: Longint = 2147483000; и вывести соответствующие элементы массивов MemL и MemW.

3) Отладить программу порождения динамических объектов используя процедуру New(A).

4) Создать динамический массив чисел типа Word, для проверки созданной структуры заполним массив числами 1,2….n, а затем выведем первый и последний элементы.

6.2 Содержание отчета

1) Постановки задачи;
2) Текст программы и результат;
3) Карта памяти. Куча;
4) Сделать анализ ошибок допущенных при работе.

6.3 Вопросы для допуска к лабораторной работе

1) Что означает A:=NIL?
2) Для чего предназначена процедура NEW(A)?
3) Что такое память?
4) Что такое параграф?
5) Что такое адрес и из чего он состоит?
6) Что такое «Куча»?
7) Что дает директива $M?
8) Что такое указатель?
9) Особенности объявления данных динамической структуры.
10) Процедуры GetMem, FreeMem.

7 Лабораторная работа № 7 «Линейные списки»

Цель работы:
Получить практические навыки работы с динамическими переменными и динамическими структурами данных.

7.1 Ход работы:

1) Разработать программы согласно индивидуального варианта задания.
2) Нарисовать структуру взаимодействия динамических данных используя:
а) простейшие действия с указателями
б) работу с очередью
в) добавление элементов очереди
г) удаление элементов очереди
3)Распечатать программу и результат

7.2 Содержание отчета

1) Постановки задачи;
2) Текст программы и результат;
3) Структурная схема.
4) Сделать анализ ошибок допущенных при работе

7.3 Методические указания

Для работы с динамическими структурами данных используются указатели. Указатели представляют собой специальный тип данных. Они принимают значения, равные адресам размещения в оперативной памяти соответствующих динамических переменных.

Списком называется структура данных, каждый элемент которой посредством указателя связывается со следующим элементом. На самый первый элемент (голову списка) имеется отдельный указатель.

Из определения следует, что каждый элемент списка содержит поле данных (оно может иметь сложную структуру) и поле ссылки на следующий элемент. После ссылки последнего элемента должно содержать пустой указатель (nil).

Число элементов связанного списка может расти или уменьшаться в зависимости от того, сколько данных мы хотим хранить в нем. Чтобы добавить новый элемент в список, необходимо:

1) получить память для него;
2) поместить туда информацию;
3) добавить элемент в конец списка (или начало).

Элемент списка состоит из разнотипных частей (хранимая информация и указатель), и его естественно представить записью. Перед описанием самой записи описывают указатель на нее:

```
Type     PList = ^TList;          { описание списка из целых чисел }
         TList = record
 Inf : Integer;   Next : PList;
  end;
```

Примеры: Создание списка.

Задача. Сформировать список, содержащий целые числа 3, 5, 1, 9.

Определим запись типа TList с полями, содержащими характеристики данных – значения очередного элемента и адреса следующего за ним элемента

```
PList = ^TList;
  TList = record
    Data : Integer;  Next : PList;
           end;
```

Чтобы список существовал, надо определить указатель на его начало. Опишем переменные.

Var Head, x : PList;

Создадим первый элемент:

New(Head); { выделяем место в памяти для переменной Head }

Head^.Next := nil; {указатель на следующий элемент пуст (такого элемента нет) } Head^.Data := 3; { заполняем информационное поле первого элемента }

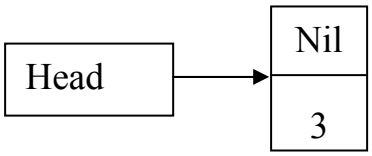


Рисунок 5 - Указатель на следующий элемент

Продолжим формирование списка, для этого нужно добавить элемент в конец списка.

Введем вспомогательную переменную указательного типа, которая будет хранить адрес последнего элемента списка:

x := Head; {сейчас последний элемент списка совпадает с его началом – рисунок 6}

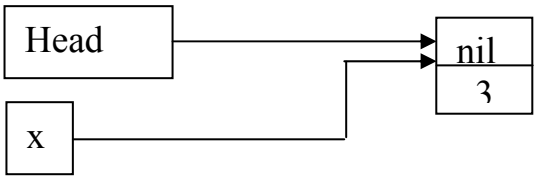


Рисунок 6 - Последний элемент списка совпадает с его началом

New(x^.Next); { выделим области памяти для следующего (2-го) элемента и поместим его адрес в адресную часть предыдущего (1-го) элемента рисунок 7}

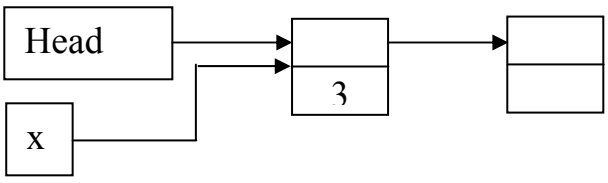


Рисунок 7 - Выделение области памяти для следующего (2-го) элемента

x := x^.Next ; { переменная x принимает значение адреса выделенной области. Таким образом осуществляется переход к следующему (2-ому) элементу списка рисунок 8}

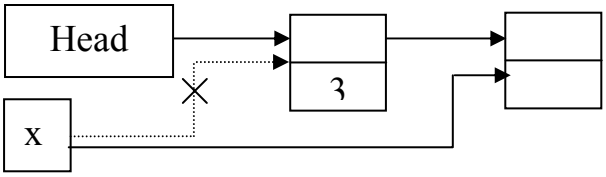


Рисунок 8 - Переменная x принимает значение адреса выделенной области

Таким образом осуществляется переход к следующему (2-ому) элементу списка

x^.Data := 5; { значение этого элемента }
x^.Next := nil; {следующего значения нет рисунок 9}

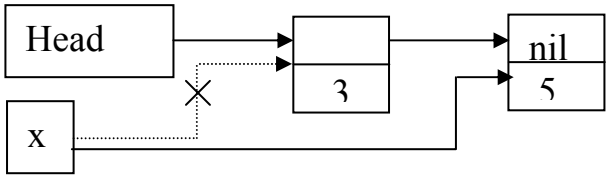


Рисунок 9 - Значение этого элемента

Остальные числа заносятся аналогично:

```
New(x^.Next); { выделим области памяти для следующего элемента }
x := x^.Next ; { переход к следующему (3-му) элементу списка }
x^.Data := 1; { значение этого элемента }
x^.Next := nil; {следующего значения нет }
New(x^.Next); { выделим области памяти для следующего элемента }
x := x^.Next ; { переход к следующему (4-му) элементу списка }
x^.Data := 9; { значение этого элемента }
x^.Next := nil; {следующего значения нет }
```

Замечание. Как видно из примера, отличным является только создание первого (Head) элемента – головы списка. Все остальные действия полностью аналогичны и их естественно выполнять в цикле.

Присоединение нового элемента к голове списка производится аналогично:

```
……..
   New(x);            { ввод значения элемента  x^.Data := … }
   x^.Next := Head;
    Head := x;
……..
```

В этом случае последний введенный элемент окажется в списке первым, а первый – последним.

Просмотр списка

Просмотр элементов списка осуществляется последовательно, начиная с его начала. Указатель List последовательно ссылается на первый, второй и т. д. элементы списка до тех пор, пока весь список не будет пройден. При этом с каждым элементом списка выполняется некоторая операция– например, печать элемента. Начальное значение List – адрес первого элемента списка (Head).

Digit – значение удаляемого элемента.

```
List := Head;
While List^.Next <> nil do
    begin
 WriteLn(List^.Data);
List := List^.Next; { переход к следующему элементу; аналог для массива i:=i+1 }
        end;
```

Удаление элемента из списка

При удалении элемента из списка необходимо различать три случая:

1) Удаление элемента из начала списка;
2) Удаление элемента из середины списка;
3) Удаление из списка.

Удаление элемента из начала списка

```
List := Head;            { запомним адрес первого элемента списка }
Head := Head^.List;      { теперь Head указывает на второй элемент списка }
Dispose(List);           { освободим память, занятую переменной List^ }
```

Удаление элемента из середины списка

Для этого нужно знать адреса удаляемого элемента и элемента, находящегося в списке перед ним.

```
List := Head;
While (List<>nil) and (List^.Data<>Digit) do
  begin
    x := List;
    List := List^.Next;
  end;
x^.Next := List^.Next;
Dispose(List);
```

Удаление элемента из конца списка

Оно производится, когда указатель x показывает на предпоследний элемент списка, а List – на последний.

```
List := Head; x := Head;
While List^.Next<>nil do
Begin    x := List;  List := List^.Next;   end;
x^.Next := nil;
Dispose(List);
```

7.4 Вариант индивидуальных заданий к зачету по теме «Динамические структуры данных. Связанные списки»

1) Сформировать список строк и

а) сохранить его в текстовом файле;

б) сохранить его в обратном порядке в текстовом файле. Использовать рекурсию;

2) Сформировать список строк из текстового файла;

3) Написать функцию, которая вычисляет среднее арифметическое элементов непустого списка;

4) Написать процедуру присоединения списка L2 к списку L1;

5) Написать функцию, которая создает список L2, являющийся копией списка L1, начинающегося с данного узла;

6) Написать функцию, которая подсчитывает количество вхождений ключа в списке;

7) Написать функцию, которая удаляет из списка все вхождения ключа;

8) Многочлен задан своими коэффициентами, которые хранятся в форме списка;

$$f(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_m x^m$$

Написать функции:

– Equal(p, q), проверяющую на равенство многочлены p и q;

– Summa(p, q, r), которая строит многочлен r = p + q.

9) Вычислить значение многочлена в целочисленной точке x.

P(x)=a0 + a1x2 +…+anxn

Коэффициенты вводятся с клавиатуры и динамически размещаются в памяти;

10) Сформировать список целых чисел и упорядочить их по неубыванию;

11) Сформировать список целых чисел и удалить из него все четные;

12) Сформировать список вещественных чисел и вычислить сумму;

13) Написать рекурсивную и нерекурсивную процедуры проверки наличия в списке заданного числа;

14) Написать функцию, которая проверяет, упорядочены ли элементы списка по алфавиту;

15) Написать функцию, подсчитывающую количество слов в списке, которые начинаются с той же буквы, что и следующее слово;

16) Определить симметричность произвольного текста любой длины. Текст должен оканчиваться точкой. Задачу решить с помощью двух списков;

17) Вычислить значение выражения

$$x_1 x_m + x_2 x_{m-1} + \ldots + x_m x_1$$

Значения $x_1, x_2, \ldots\ldots, x_m$ вводятся с клавиатуры и динамически размещаются в памяти;

18) Написать функцию, которая использует исходный список L и создает два новых списка L1 и L2. L1 содержит нечетные узлы, а L2 – четные;

19) Написать функцию, которая использует исходный список L и создает два новых списка L1 и L2. L1 содержит нечетные числа, а L2 – четные;

20) Сформировать два списка, отсортировать их объединить в один, не нарушая порядка;

21) Составить программу, которая вставляет в список *L* новый элемент *F* за каждым вождением элемента *E*;

22) Составить программу, которая вставляет в список *L* новый элемент F перед первым вхождением элемента *E,* если *E* входит в *L*;

23) Составить программу, которая вставляет в непустой список *L,* элементы которого упорядочены по неубыванию, новый элемент *E* так, чтобы сохранилась упорядоченность;

24) Составить программу, которая удаляет из списка *L* все элементы *E,* если таковые имеются;

25) Составить программу, которая удаляет из списка *L* за каждым вхождением элемента *E* один элемент, если таковой имеется и он отличен от *E*;

26) Составить программу, которая удаляет из списка *L* все отрицательные элементы;

27) Составить программу, которая проверяет, есть ли в списке *L* хотя бы два одинаковых элемента;

28) Составить программу, которая переносит в конец непустого списка *L* его первый элемент;

29) Составить программу, которая вставляет в список *L* за первым вхождением элемента *E* все элементы списка *L,* если *E* входит в *L*;

30) Составить программу, которая переворачивает список *L,* т.е. изменяет ссылки в этом списке так, чтобы его элементы оказались расположенными в обратном порядке;

31) Составить программу, которая в списке *L* из каждой группы подряд идущих одинаковых элементов оставляет только один;

32) Составить программу, которая формирует список *L,* включив в него по одному разу элементы, которые входят одновременно в оба списка *Lx* и *Eг*

33) Составить программу, которая формирует список *L,* включив в него по одному разу элементы, которые входят в список *L1,* но не входят в список *L2*

34) Составить программу для упорядочения в порядке возрастания элементов однонаправленного списка;

35) Составить программу, заполняющую список последовательностью случайных различных целых чисел и суммирующую те его элементы, которые расположены между минимальным и максимальным элементом (если минимальный элемент предшествует максимальному);

36) Дан список, содержащий целые числа. Сформировать другой список из элементов данного, абсолютные величины которых являются простыми числами;

37) Дан список, содержащий натуральные числа. Удалить те его элементы, которые кратны заданному числу *к*

38) Дан список, элементами которого являются векторы

(Const NMax=200; Type Vector=Array[1..NMax] Of Real;).

Сформировать список из длин этих векторов;

39) Элементами списка являются слова — имена существительные, записанные в именительном падеже (строки длиной не более 15 символов). Составить программу, которая добавляет за каждым словом все его падежные формы;

40) Дан список, содержащий целые числа. Определить количество различных элементов этого списка;

41) Даны упорядоченные списки £, и *Lj*. Вставить элементы списка *Ь?* в список *Lb* не нарушая его упорядоченности;

42) Дан список, содержащий запись неотрицательных целых чисел в двоичной системе счисления. Заменить каждый элемент списка на его запись в шестнадцатеричной системе счисления;

43) Дан список, содержащий обыкновенные дроби вида P / q, (P — целое, Q — натуральное). Составить программу для суммирования модулей этих дробей. Ответ представить в виде обыкновенной несократимой дроби;

44) Программа должна находить среднее арифметическое элементов непустого однонаправленного списка вещественных чисел, заменять все вхождения числа *х* на число *у,* менять местами первый и последний элементы, проверять, упорядочены ли числа в списке по возрастанию;

45) Дан список вещественных чисел. Написать следующие функции:

а) проверки наличия в нем двух одинаковых элементов;

б) переноса в начало его последнего элемента;

в) переноса в конец его первого элемента;

г) вставки списка самого в себя вслед за первым вхождением числа х;

46) Дан список строк. Написать следующие подпрограммы:

а) обращение списка (изменить ссылки в списке так, чтобы элементы оказались расположены в противоположном порядке);

б) из каждой группы подряд идущих элементов оставить только один;

в) оставить в списке только первые вхождения одинаковых элементов.

47) Даны два списка *L* и *L2* пар вещественных чисел. Написать подпрограммы, возвращающие новый список *L,* включающий в себя:

а) пары списка *Z*,, первая координата которых встречается как вторая координата у пар списка *L2;*

б) пары *(x, y)* списка *L\,* встречающиеся в виде *(y, x)* в списке *Li,*

в) пары *(x,y),* где *x<y* списка *Lv*

48) Даны два списка *Lx* и *L2* вещественных чисел. Написать под программы, возвращающие новый список *L,* включающий по одному разу числа, которые;

а) входят одновременно в оба списка;

б) входят хотя бы в один из списков;

в) входят в один из списков *L1* и *L2,* но в то же время не входят в другой из них;

г) входят в список *L1,* но не входят в список *L2.*

49) Целое длинное число представляется строкой цифр. Написать программу, упорядочивающую числа по неубыванию.

50) Дан список слов, среди которых есть пустые. Написать подпрограмму, выполняющую следующее действие:

а) перестановку первого и последнего непустых слов;

б) печать текста из первых букв непустых слов;

в) удаление из непустых слов первых букв;

г) определение количества слов в непустом списке, отличных от последнего.

7.5 Вопросы к защите лабораторной работы

1) Что такое указатели? Какие значения они могут принимать? Какие операции возможны над указателями?

2) Что представляют собой динамические структуры данных? Для чего они используются? Чем отличаются от данных статического типа?

3) Какие стандартные процедуры существуют в языке Pascal для работы с указателями?

4) Зачем различать типы указателей?

5) Какие операции требуется выполнить для вставки и удаления элемента списка?

6) Сколько элементов может содержать список?

7) Можно ли для построения списка обойтись одной переменной?

8 Лабораторная работа № 8 «Формирование списка с одновременным упорядочением его элементов»

Цель задания:

1) Ознакомление с Динамической структурой данных – однонаправленным списком;

2) Получение навыков работы с переменными ссылочного типа.

8.1 Постановка задачи

Для конкретного варианта составить входную информацию. Разработать программу, которая обеспечивает последовательное занесение информации в однонаправленный список с одновременным упорядочением по указанному в варианте признаку. По окончании формирования списка распечатать его.

8.2 Содержание отчета

1) Постановка задачи;
2) Исходные данные;
3) Текст программы и результаты ее выполнения;
4) Анализ допущенных ошибок.

8.3 Методические указания

Приведем в качестве примера работы со списками программу, которая вводит с терминала названия учебников (последнее название является фиктивным, оно характеризует конец списка и состоит из десяти букв Я), динамически отводит место в памяти под каждое название и строит из них связанный список, упорядоченный по алфавиту. По окончании формирования каталог книг выводится на терминал. Использование списковых структур при решении подобных задач имеет определенные т преимущества: исключает предварительное резервирование памяти, дает возможность сортировать (упорядочивать список одновременно с вводом, позволяет упрощать дальнейшую работу с данными, например достаточно просто реализовать операции удаления и вставки новых элементов. Такой упорядоченный список можно переписать во внешний файл для долговременного хранения.

```
Program список (input,output);
Type
Назв=packed array[1..10] of char;
     Ссылка=^элемент;
```

```
Элемент=record
 cc: Ссылка;  Дан: Назв;
    End;
Var  Книга: Назв;
       нач, тек, об, обпр : Ссылка;
       I: integer
Begin                                   {*формирование 1-го элемента списка*}
New (нач);
Writeln ('Вводите первое название');
For i:=1 to 10 do read (книга[i]);
  нач^cc:=nil;    нач^.дан:=книга;
                                        {*формирование списка*}
While книга<>'ЯЯЯЯЯЯЯЯЯЯ'  do
  Begin
  Тек:=нач;     i:=i+1;
 New (об);
Writeln ('вводите очередное название')
 For i:=1 to 10 do
      Read (книга[i]) ;
                                        {*Поиск подходящего места*}
While  (Тек <> NIL) and (Тек^.дан<книга) do
 Begin   Обпр:=тек;  Тек:= Тек^.cc;   End;
                                        {*Вставка нового элемента*}
об^.cc:= Тек;   об^.дан:= книга;
       if Тек =нач then нач:=об  else  Обпр^cc:=об;
 End;
                  {*Вывод на терминал упорядоченного каталога книг*}
writeln ('Каталог учебников: ');
Тек:=нач;
While  Тек^.cc < > NIL do
     begin
Writeln (Тек^.дан);  Тек:= Тек^.cc;
      End;
Dispose(нач);
  End.
```

В начале программы формируется первый элемент списка и задается значение начального указателя НАЧ. Затем, пока не будет введено название учебника состоящего из десяти букв Я, под каждое вводимое название динамически отводится место в памяти оператором NEW, после чего просматривается список с самого начала до те пор, пока не будет найдена соответствующая позиция для нового элемента или не будет достигнут конец списка. В найденное место вставляется новый элемент списка. В конце программы упорядоченный список распечатывается.

Протокол работы программы:
ВВОДИТЕ ПЕРВОЕ НАЗВАНИЕ
МАТЕМАТИКА
ВВОДИ ОЧЕРЕДНОЕ НАЗВАНИЕ
АЛГЕБРА
ВВОДИ ОЧЕРЕДНОЕ НАЗВАНИЕ
ФИЗИКА
ВВОДИ ОЧЕРЕДНОЕ НАЗВАНИЕ
ЯЯЯЯЯЯЯЯЯЯ
КАТАЛОГ УЧЕБНИКОВ:
АЛГЕБРА
МАТЕМАТИКА
ФИЗИКА

Данная программа правильно работает на тех ЭВМ, на которых множество русских букв упорядочено по алфавиту. В противном случае операции отношения над строками, содержащими русские буквы, дают неправильный результат. Для реализации операций отношения в соответствии с русским алфавитом на таких ЭВМ необходимо создавать вспомогательное программное обеспечение. Этот вопрос выходит за рамки данного учебного пособия.

8.4 Варианты индивидуальных заданий

Задание 1. Упорядочение списка

1.1 Составить список учебников для N-го курса, указав название, фа-милию автора, год издания, цену, тираж и упорядочить его по заданному признаку.

№ курса	Название поля
I.	Фамилия автора
II.	Название
III.	Год издания
IV.	Цена
V.	Тираж

1.2 Составить список кабинетов техникума для М этажа, указав название кабинета, номер комнаты, количество посадочных мест и упорядочить его по заданному признаку.

№ этажа	Название поля
I.	Название кабинета
II.	Номер комнаты
III.	Количество мест

9 Лабораторная работа №9 «Исключение элементов из списка»

Цель задания:

1) Ознакомление с возможностью выполнения операции исключения элементов из списка;

2) Закрепление навыков использования ссылочных типов данных.

9.1 Постановка задачи

Составить список учебной группы, содержащий не менее 15 учащихся. Указать для каждого учащегося оценки, полученные на последних четырех экзаменах. Разработать программу, которая вводит с терминала данные о каждом учащемся, заносит эту информацию в однонаправленный список. Обработать список согласно конкретному варианту.

9.2 Содержание отчета

1) Постановка задачи;
2) Список группы учащихся с оценками;
3) Текст программы и результаты ее выполнения;
4) Выводы.

9.3 Методические указания

При выполнении задания следует ознакомиться с приведенной ниже программой. Программа ИСКЛ вводит данные о каждом учащемся, строит список, а затем удаляет из списка элементы, относящиеся к неуспевающим учащимся.

```
Program искл {input,output};
Type    Данные=record
    ФИО: packed array[1..7] of char;
      Оценки:array[1..4] of integer;
              End;
        Ссылка=^Запись;
     Запись=record
    СС: ссылка
          Y:Данные;
                End;
      Var    Нач, s1,s2: ссылка;
             I,j,n: integer;
```

```
Begin
Writeln (‘Вводите количество фамилий в списке’);
Readln (n);
New (s1);  Нач:=s1;
For i:=1 to n do
     Begin
s1^.cc:=nil;
       Writeln (‘Вводите фамилию из 7 букв’);
For j:=1 to 7 do  read (s1^.y.ФИО [j]);
    For j:=1 to 4 do read (s1^.y.оценки [j]);
s2:=s1;  New (s1)
        If i<>n then s2 ^ .CC:= s1 Else dispose (s1);
     End;                                {*Закончено формирование списка*}
          s1:=нач;
 repeat
   For i:=1 to 4 do
If s1^.y.оценки [i]=2 then  if s1=нач then нач:= s1^.cc
     Else s2^.cc:= s1^.cc;  s2:=s1;     s1:=s1^.CC;
Until s1=nil;
  writeln (‘Список успевающих учащихся: ’);              {*Печать списка*}
s1=нач;
while s1<>nil do
begin
        write (s1^.y.ФИО [j]);
For i:=1 to 4 do
          write (‘  ’,s1^.y.оценки [i]:2);
Writeln;
             s1:=s1^.cc;
End;
   End.
```

Данную программу можно записать и с использованием оператора присоединения. При этом следует помнить, что в области действия оператора присоединения нельзя изменять элементы списка переменных-записей, указанных в заголовке.

Протокол работы программы ИСКЛ:

```
Вводи количество фамилий в списке: 3
Вводи фамилию из 7 букв и оценки
КОТОВ 4 4 5 2
Вводи фамилию из 7 букв и оценки
СОМОВ 5 5 5 5
Вводи фамилию из 7 букв и оценки
ТОКАРЕВ 3 4 4 3
Список успевающих учащихся
СОМОВ 5 5 5 5
  ТОКАРЕВ 3 4 4 3
```

9.4 Варианты индивидуальных заданий

Исключить из списка элементы, относящиеся к учащимся, у которых:
1) Средний балл меньше среднего балла группы;
2) Средний балл меньше 4,5;
3) Средний балл больше 4;
4) Все оценки 5;
5) Одна оценка 4, а остальные 5;
6) Оценка, полученная на первом экзамене, 2;
7) Оценка, полученная на втором экзамене, 5;
8) Нет удовлетворительных и неудовлетворительных оценок;
9) Больше одной оценки 2;
10) Одна оценка 3, а остальные 4 и 5;
Распечатать оставшийся список

9.5 Вопросы к защите лабораторной работы

1) В чем особенность объявления данных динамической структуры?
2) Что выполняет операция разименования?
3) С помощью каких процедур происходит распределение памяти под динамические переменные?
4) Какие состояния может принимать указательная переменная?
5) В каких случаях указатель может находиться в неопределенном состоянии?
6) В чем различие между состоянием nil и неопределенном состоянием.
7) Какие действия выполняют процедуры New и Dispose?

10 Лабораторная работа № 10 «Выполнение операций над списковыми структурами в среде программирования Delphi»

Цель задания:
1) Ознакомление с возможностями представления строк символов в виде списков;
2) Закрепление навыков выполнения операций над списками.

10.1 Постановка задачи

Ввести с терминала строку символов, формируя из ее элементов однонаправленный список. Обработать список согласно конкретному варианту и распечатать результат.

10.2 Содержание отчета:

1) Постановка задачи;
2) Входная строка символов;
3) Текст программы;
4) Выходная строка символов;
5) Анализ допущенных ошибок.

10.3 Методические указания

При выполнении задания следует ознакомиться с приведенной ниже программой. Программа ОБРАБ формирует однонаправленный список из строки. В поле данных каждого элемента списка записывается отдельный символ. В программе производится анализ первою символа входной строки: если это буква ‘А’, то в конец списка добавляется еще одна буква ‘А’, иначе из списка исключаются все буквы ‘А’. Полученный результат выводится на печать.

```
Program обраб(input,output);
Type  Ссылка=^элемент;
       Элемент=record
       W:ссылка; S:char;
                    end;
 Var Нач, тек, pr : ссылка;  PC, C:char;
Begin
      Write (‘=>’);  Read (c);  Pc:=c;
           New (нач);   Нач^.s:=c;  Нач^.w:=NIL;    pr:=нач;
 While  not eoln do  Begin  Read (c);
           New (тек);     Pr^.w:=тек; Тек^.w:=NIL;  Тек^.s:=c;  Pr:=тек;
 If pc=’a’ then  begin                    {*Добавление в конец списка буквы А*}
     Тек:=нач;
    While  Тек^.w <> NIL do
    begin
 Тек:= Тек^.w;    New (pr);   Тек^.w:=pr;   Pr^.w:=nil;   Pr^.s:=’A’
       End
  else   begin                                {*Удаление из списка всех букв*};
Тек:=нач;
While  Тек <> NIL do     begin
 If Тек^.s=’a’ then pr^.w:=Тек^.w  else      pr:=Тек;      Тек:=Тек^.w;
                              End;
End
 Тек:=нач;                                        {*Печать измененной строки*}
         Writeln (‘Изменённая строка’);
    While  Тек <> NIL do
```

```
Begin  Write (Тек^.s);  Тек:=Тек^.w; End;
Writeln;
End.
```

10.3.1 Динамические переменные

Динамической переменной называется переменная, память для которой выделяется во время работы программы.

Выделение памяти для динамической переменной осуществляется вызовом процедуры new.У процедуры new один параметр — указатель на переменную того типа, память для которой надо выделить. Например, если p является указателем на переменную типа real, то в результате выполнения процедуры new (p) , будет создана переменная типа real b ее адрес присвоится переменной-указателю p.

У динамической переменной нет имени, поэтому обратиться к ней можно только при помощи указателя.

Процедура, использующая динамические переменные, перед завершением своей работы должна освободить занимаемую этими переменными память или, как говорят программисты, уничтожить динамические переменные, для освобождения памяти, занимаемой динамической переменной, используется

процедура Dispose (p), которая имеет один параметр — указатель на динамическую переменную.

Например, если p указатель на динамическую переменную, память для которой выделена инструкцией new (p), то инструкция Dispose (p) освобождает занимаемую динамической переменной память.

Следующая процедура демонстрирует создание, использование и уничтожение динамических переменных.

```
procedure TForm1.Button1Click(Sender: TObject);
  Var  P1,p2,p3:^integer;        //  указатель на переменную типа integer
                                 //  создадим динамические переменные типа integer
Begin  new (p1); new (p2);  new (p3);
p1^:=StrToInt (Edit1.text);
    p2^:=StrToInt (Edit2.text);
    p3^:=p1^+p2^;
ShowMessage (‘Сумма введенных чисел равна ’+inttoStr (p3^));
        dispose (p1);                    //  уничтожение динамических объектов
dispose (p2);
  dispose (p3);
end;
```

Процедура создает три динамические переменные. Две переменные, на которые указывают p1 и p2, получают значения из полей редактирования (EDIT1 и EDIT2). Значение третьей переменной вычисляется как сумма первых двух.

10.3.2 Связанные списки

Указатели и динамические переменные позволяют создавать сложные динамические структуры данных, такие как связанные списки и деревья. Связанный список можно изобразить графически (рисунок 10).

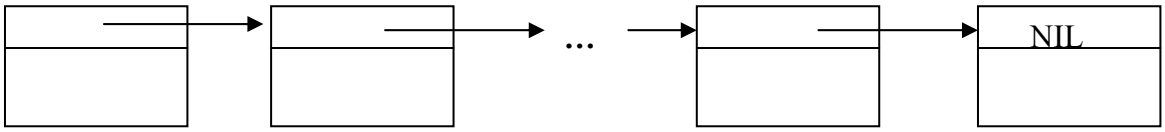
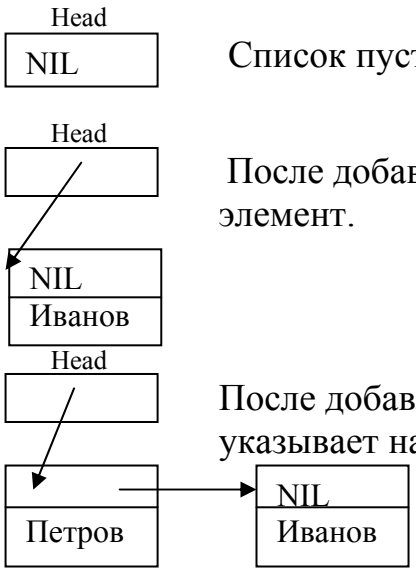


Рисунок 10 - Связанный список

Каждый элемент списка (узел) представляет собой запись, состоящую из двух частей. Первая часть — информационная. Вторая часть обеспечивает связь со следующим, и возможно с предыдущим, элементом списка. Список, в котором обеспечивается связь только со следующим элементом, называется односвязным. Чтобы программа могла использовать список, надо определить тип компонентов списка и переменную-указатель на первый элемент списка. Вот пример объявления компонента списка студентов:

```
Type TPStudent: ^TStudent;        // указатель на переменную типа TStudent
{Описание типа элемента списка}
TStudent = record
Surname: string [20];     name: string [20];     Group: integer;
address: string [60];     next: TPStudent;  // следующий элемент списка
          end;
var  head: TPStudent;   //  указатель на первый элемент списка
```

Добавлять данные можно в начало, в конец или в нужное место списка. Во всех этих случаях необходимо корректировать указатели. На рисунке 11 изображен процесс добавления элементов в начало односвязного списка.



Список пустой. Указатель head ни на что не указывает.

После добавления одного элемента head указывает на этот элемент.

После добавления второго элемента в начало списка head указывает на этот элемент.

Рисунок 11- Процесс добавления элементов в начало односвязного списка

Следующая программа формирует список студентов, добавляя фамилии в начало списка. Данные вводятся в поля редактирования (Edit1 и Edit2) диалогового окна программы (рисунок 12) и при щелчке по кнопке **Добавить** (Button1) добавляется в список.

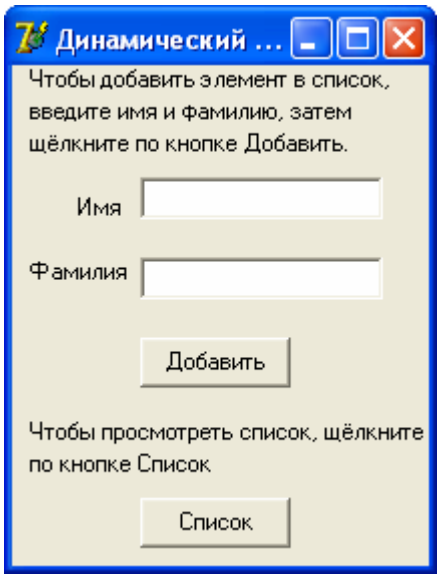


Рисунок 12- Диалогового окна программы с полями редактирования

Непосредственно добавление элемента в список выполняет процедура TForm1.Button1Click, которая создает динамическую переменную-запись – присваивает её полям значения – содержимое полей ввода диалогового окна, и корректирует значение указателя head.

```
procedure TForm1.Button1Click(Sender: TObject);
Var
   Curr : TPStudent; // новый, текущий элемент списка

begin
  new (curr);                    // создание нового элемента списка
  curr^.f_name:=Edit1.text;
  curr^.l_name:=Edit2.text;
                                 // добавление в начало списка
  curr^.next:=head;
  head:=curr;
  Edit1.Text:='';
  Edit2.Text:='';
end;
```

Процедура TForm1.Button2Click, которая запускается при щелчке на кнопке **Список,** перемещаясь по списку от начала, на которое указывает переменная head, формирует строковое представление списка до тех пор, пока значение поля next очередного элемента списка не будет равно nil.

```
procedure TForm1.Button2Click(Sender: TObject);
Var
   curr:TPStudent;              // текущий элемент списка
   n:integer;                   //  длина (кол-во) элементов списка
   st:string;                   //  строковое представление списка
begin
    n:=0;     st:='';     curr:=head;
 while curr <> NIL do              //  пока не конец списка
      begin
n:=n+1;    st:=st+curr^.f_name+' '+curr^.f_name+#13;    curr:=curr^.next;
      end;
if n <> 0  then ShowMessage ('Список: '+#13+st)
 else    ShowMessage ('В списке нет элементов');
       end;
```

Переменная head и тип TStudent используются обеими процедурами, поэтому их объявление следует поместить в раздел implementation перед объявлением приведенных выше процедур.

```
Type   TPStudent: ^TStudent;              //  указатель на тип TPStudent
         TStudent = Record
         f_name: string[20];              //  фамилия
         l_name:string[20];               //  имя
         next: TPStudent;                 //  следующий элемент списка
                    end;
var   head: TPStudent;                    //  начало (голова) списка
```

Как правило, списки упорядочены. Порядок следования элементов в списке определяется содержимым одного из полей. Например, список с информацией о людях обычно упорядочен по полю, содержащему фамилию.

Чтобы добавить узел в упорядоченный список, нужно найти узел, за которым должен следовать добавляемый элемент, и установить его указатель на новый элемент, а указатель нового узла установить на узел, перед которым он вставляется (рисунок 13).

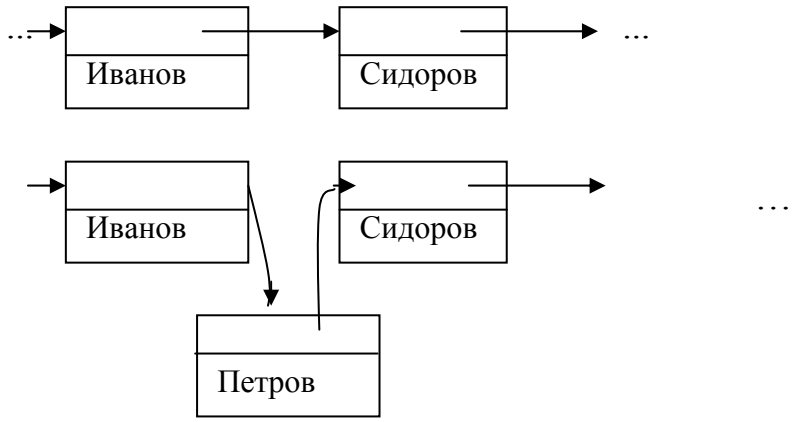


Рисунок 13 - Добавление узла в упорядоченный список

Следующая программа Упорядоченный список 2 формирует список упорядоченный по полю «фамилия». Вид диалогового окна программы рисунок 14 :

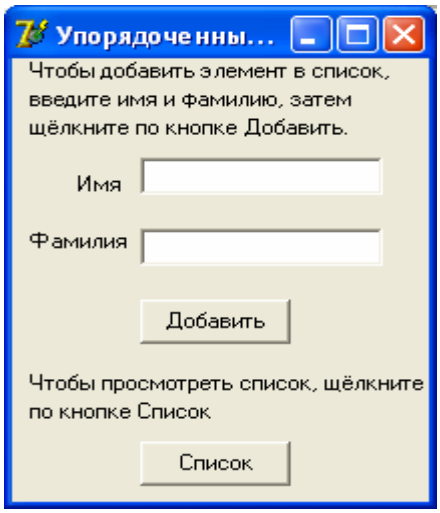


Рисунок 14 - Формирует упорядоченный список

Данные вводятся в поля редактирования (Edit1 и Edit2) и при щелчке на кнопку **Добавить** (Button1) добавляется в список таким образом, что список всегда упорядочен по полю «фамилия». Непосредственное добавление узла в список выполняет процедура TForm1.Button1Click.

```
Procedure TForm1.Button1Click (Sender: TObject);
Var
   Node: TPStudent;   //  новый узел списка
   Curr:  TPStudent;   //  текущий узел списка
   Pre: TPStudent;     //  предыдущий относительно curr узел
Begin
   New (node);  //  создание нового элемента списка
   node^.f_name:=Edit1.text;
   node^.l_name:=Edit2.text;
   //  добавление узла в список
   //  сначала найдем подходящее место в списке для узла
   curr:=head;
   pre:=nil;
    {Внимание!}
```

Если приведенное ниже условие заменить на (node.f_name > curr^.f_name) and (curr <> nil) то при проверке условия (node.f_name > curr^.f_name) во время добавления первого узла возникает ошибка времени выполнения, т.к. (curr = NIL) и, следовательно переменной (curr^._name) **нет!**

В используемом варианте условия ошибки не возникает, т.к. сначала проверяется условие (curr <> NIL), значение которого False и второе условие в этом случае не проверяется. }

```
While (curr <> NIL) and (node.f_name > curr^.f_name) do
  Begin
  pre:=curr;                    //  введите значение больше текущего
     curr:=curr^.next;          //  к следующему узлу
  end;
if pre = NIL then    begin        //  новый узел в начало списка
   node^.next:=head;
   head:=node;
   end
else
    begin                        //  новый узел после pre, перед curr
        node^.next:= pre^.next;
        pre^.next:=node;
     end;
Edit1.text:='';
Edit2.text:='';
 End;
```

Процедура TForm1.Button1Click создаёт динамическую переменную-запись, присваивает её полям значения – содержимое полей ввода диалогового окна, находит подходящее для узла место и коррекцией значения указателя *next* узла, после которого должен быть помещен новый узел, добавляет этот узел в список.

Вывод списка выполняет процедура, которая запускается щелчком по кнопке **Список** (Button2).

```
procedure TForm1.Button2Click(Sender: TObject);
Var
   curr:TPStudent;  // текущий элемент списка
   n:integer;   //  длина (кол-во) элементов списка
   st:string;  //  строковое представление списка
begin
    n:=0;
    st:='';
    curr:=head;
    while curr <> NIL do
      begin
        n:=n+1;
        st:=st+curr^.f_name+' '+curr^.f_name+#13;
        curr:=curr^.next;
      end;
    if n <> 0
      then ShowMessage ('Список: '+#13+st)
      else ShowMessage ('В списке нет элементов');
end;
```

Приведенное ниже описание типа узла списка и переменной head следует поместить в раздел implementation, перед текстом процедур.

```
Type
   TPStudent: ^TStudent;  //  указатель на тип TPStudent
   TStudent = Record
     f_name: string[20];  //  фамилия
      l_name:string[20];  //  имя
      next: TPStudent;  //  следующий элемент списка
end;
var
   head: TPStudent; //  начало (голова) списка
```

После запуска программы и ввода нескольких фамилий, например, в такой последовательности: Иванов, Яковлев, Алексеев, Петров, список выглядит так, как показано на рисунке 15.

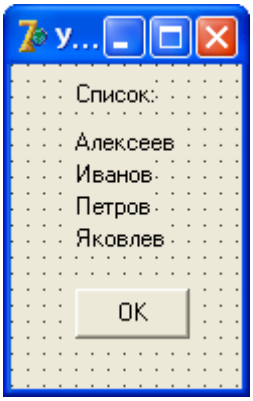


Рисунок 15 - Список после ввода фамилий

Чтобы удалить узел, надо скорректировать значения указателя узла, который находится перед удаляемым узлом, как показано на рисунке 16.

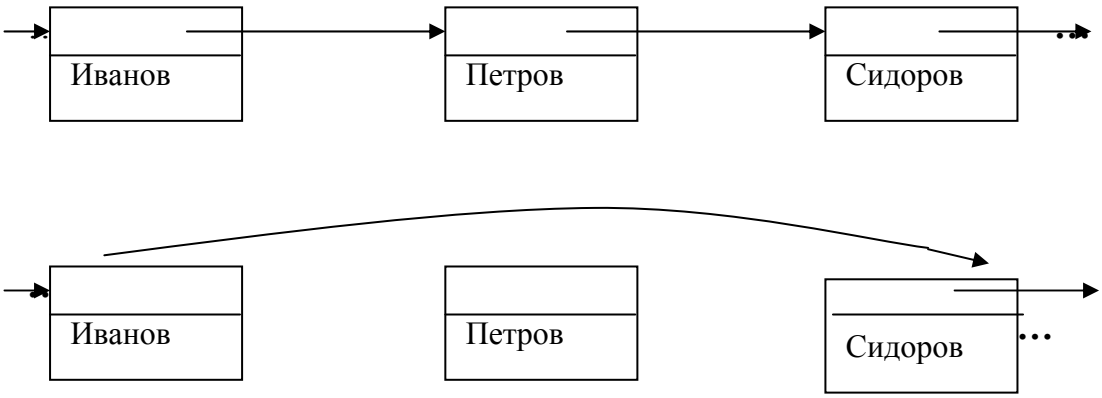


Рисунок 16 - Удаления узла который находиться перед удаляемым узлом

Узел – это динамическая переменная, которая теперь не нужна; поэтому, после исключения узла из списка, надо освободить память, занимаемую этой переменной. Освобождение динамической памяти, или, как иногда говорят, уничтожение переменной, выполняется вызовом процедуры Dispose. У процедуры Dispose один параметр — указатель на переменную, память, занимаемая которой, должна быть освобождена. Например, в программе: создается динамическая переменная, затем она уничтожается.

```
Var  P^:integer;
Begin
      New (p);
{ здесь инструкции программы}
dispose (p);
end;
```

Освободившуюся память смогут использовать другие переменные. Если этого не делать, то, возможно, из-за недостатка свободной памяти в какой-то момент времени программа не сможет создать очередную динамическую переменную.

Следующая программа позволяет добавлять и удалять узлы упорядоченного списка. Диалоговое окно программы приведено на рисунке 17:

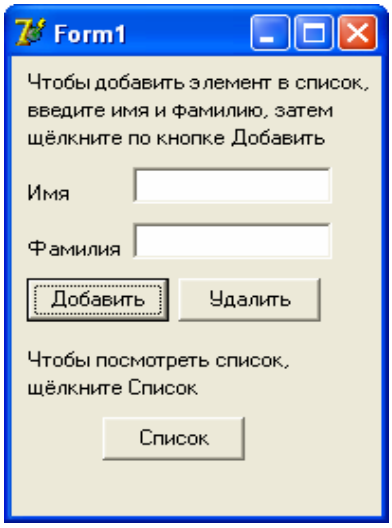


Рисунок 17 - Диалоговое окно программы добавления и удаления узла упорядоченного списка

Процедуры добавления узла в список и вывода списка, а также описание типа узла списка ни чем не отличаются от соответствующих процедур и описания рассмотренной ранее программы Упорядоченный список, поэтому они здесь не приводятся. Удаление узла из списка выполняет процедура, которая запускается при щелчке на кнопке Удалить (Button3)

```
procedure TForm1.Button3Click(Sender: TObject);
Var
   curr:TPStudent;                                  //  текущий, проверяемый узел
   pre:TPStudent;                                   //  предыдущий узел
   found:TPStudent;       //   TRUE – узел, который надо удалить, есть в списке
begin
 if head = NIL  then MessageDlg ('Список пустой', mtError,[mbOk],0)
 else        begin
          curr:=head;
          pre:=NIL;
          found:=false;                          //  найти узел, который надо удалит
 while (curr <> NIL) and (not found) do
   begin
if (curr^.f_name = Edit1.Text) and (curr^.l_name = Edit2.Text)  then found:=true
                                                          //  Нужный узел найден
else                                                      //  к следующему узлу
        begin
              pre:=curr;
              curr:=curr^.next;
           end;
         end;
 if found  then   begin                                   // удаляем узел
 if pre = NIL   then head:=curr^.next                     //  удаляем первый узел
 else      pre^.next:=curr.next;
           dispose (curr);
           Edit1.Text:='';
           Edit2.Text:='';
         end
 else                                  //  узла, который надо удалить, в списке нет
messagedlg
('Узел'+#13+'Имя:'+Edit1.Text+#13+'Фамилия:'+Edit2.Text+#13+'в списке не
найден.',mtError,[mbOk],0);
          end;
   end;
```

Процедура просматривает список от начала и сравнивает содержимое полей текущего узла с содержимым полей ввода диалогового окна. Если их содержимое совпадает, то программа исключает узел из списка и освобождает занимаемую этим узлом память. Если узла, который надо удалить, в списке нет, то программа выводит сообщение об ошибке.

10.3.3 Варианты индивидуальных заданий

1) Удалить первые две фамилии в списке;
2) Удалить последние три фамилии из списка;
3) Удалить все фамилии на букву К;
4) Удвоить все фамилии ;
5) Добавить в конец списка слово END;
6) Поменять местами первый и последний фамилии списка;
7) Поменять местами первую и вторую фамилию списка;
8) Поменять местами последнюю и предпоследнюю фамилию списка;
9) Подсчитать в списке число фамилий начинающихся на А и В, и если букв А больше, чем букв В, то удалить в строке все фамилии В;
10) Подсчитать число фамилий в списке, и если число нечетное, то удалить фамилии, стоящие посередине списка.

11 Контрольные вопросы для подготовки к экзамену

1) Основные понятия программного обеспечения: Программа, задача классификация задач, предметная область, схема процесса создания программ, постановка задачи, основные характеристики функциональных задач, структурная схема входной и выходной информации;
2) Категория специалистов: Схема взаимодействия специалистов связанных с созданием и эксплуатацией программ;
3) Характеристика программного продукта: классификация программного продукта, программный продукт подготовленный к эксплуатации должен иметь? Основные характеристики программ. Программный продукт имеет показатели качества;
4) Жизненный цикл программного продукта. Этапы жизненного цикла. Кривая продаж;
5) Методология проектирования программного продукта. Классификация методов проектирования. Что характерно для неавтоматизированного проектирование алгоритмов и программ. Структурное проектирование. Информационное моделирование предметной области. Объектно-ориентированный подход к проектированию ПП.
6) Этапы создание программных продуктов. Составление технического задания на программирование. Технический проект. Рабочая документация;
7) Структура программного продукта. Основные цели структуризации программных продуктов;
8) Проектирование интерфейса пользователя.
Особенности диалогового режима работы программного продукта. Классификация систем поддерживающих диалоговые процессы. Как представляются диалоговые системы поддерживающие жесткий сценарий диалога.

Что определяется для управления диалоговым процессом. Что выполняет описание сценария диалога. Дайте определения пакета прикладных программ;

9) Графический интерфейс пользователя. Требования предъявляемые к стандартному интерфейсу. Что такое объект управления в графическом интерфейсе пользователя;

10) Структурное проектирование и программирование.

Метод нисходящего проектирования. Функциональная структура приложения. Модульное программирование: Свойства программного модуля. Функционально-модульная структура приложения. В чем заключается метод информационного моделирования при разработке программных продуктов.

Структурное программирование. Базовые - управляющие структуры алгоритма.

11) Объетно-ориентированное проектирование: Основные понятия ООП. Методология ООП;

12) Усложненные структуры программ и данных. Структура модуля: Трансляция модуля. Создание выполнимой программы. Долговременно хранимые и стандартные модули. Оверлейные модули: Оверлейная структура. Основные принципы построения оверлейных структур.

13) Динамическая память: Карта памяти. Виды указателей динамической памяти. Списковые структуры.

Список использованных источников

1 **Рудаков, А.В.** Технология разработки программных продуктов: учебное пособие [Текст] /Рудаков А.В. – М.: ACADEMA, 2005.-207 с.(Среднее профессиональное образование)

2 **Макарова, Н.В.** Информатика: учебник [Текст] / Н.В.Макарова -М.: Финансы и статистика, 2004.-767с.

3 **Гофман В.Э.** Delphi: Руководство программиста [Текст] / В.Э.Гофман – СПб.: БХВ-Петербург, 2005.-1152с.

4 **Семагин И.Г.** Основы программирования [Текст] /И.Г.Семагин, А.П.Шестаков. –М.: Академия, 2004. –385 с.

5 **Попов В.Б.** TURBO PASCAL для школьников [Текст] /В.Б.Попов –М.: Финансы и статистика, 2007. -518 с.

6 **Житкова О.А.**Справочные материалы по программированию на языке Паскаль [Текст] /О.А.Житкова, Е.К.Кудрявцева–М. Интеллект-центр, 2002. -77 с.

7 **Меженный О.А.** TURBO PASCAL. Учитесь программировать [Текст] /О.А.Меженный. –М.: Диалектика, 2001.-88 с.

8 **Фаронов В.В.** TURBO PASCAL 7.0 [Текст] /В.В.Фаронов. –М.: Нолидж, 2007. -312 с.

9 **Марченко А.И.** Программирование в среде TurboPascal 7.0 [Текст] /А.И.Марченко. –М.: Бином Универсал, 2005. – 485с.

10 **Культин Н.** Delphi Программирование на Object Pascal [Текст] /Н.Культин. –М.: BHV–Санкт-Петербург, 2005. –297с.

11 **Зубов В.С.** Программирование на языке TP [Текст] /В.С.Зубов. –М.: ТТО «Филинъ», 2000. –301с.

12 **Васюокова Н.Д.** Практикум по основам программирования язык Паскаль [Текст] /Н.Д.Васюокова, В.В.Тюляева. –М.: Высшая школа, 2000. –с.

13 **Бондарев В.М.** Основы программирования [Текст] /В.М.Бондарев, В.И.Рублинецкий, Е.Г.Качко. –М.: ТТО «Филинъ», 2004. –363с.

Приложение А

(обязательное)

Пример оформления титульного листа

Министерство образования и науки Российской Федерации
Федеральное агентство по образованию

Государственное образовательное учреждение
высшего профессионального образования
«Оренбургский государственный университет»

КОЛЛЕДЖ ЭЛЕКТРОНИКИ И БИЗНЕСА

Кафедра вычислительной техники и математики

ЛАБОРАТОРНЫЕ РАБОТЫ

по дисциплине «Технология разработки программных продуктов»

КОГУ 220105.51 4005.03П3

порядковый номер по списку

Преподаватель
_____Дель Л. А.
«___»_____ 2009 г.
Исполнитель
Студент гр. 23П3
_____Дель В.Г.
«___»_____ 2009 г.

Оренбург 2009

Рецензия

Методической комиссии колледжа Электроники и бизнеса ОГУ на авторскую работу методические указания к лабораторно-практическим занятиям по дисциплине «технология разработки программных продуктов».

Автор - преподаватель кафедры «Вычислительной техники и математики» колледжа электроники и бизнеса ОГУ

Дель Людмила Анатольевна

Методические указания к лабораторным работам выполнены в соответствии с рабочей программой дисциплины «технология разработки программных продуктов» и государственными требованиями к минимуму содержания и уровня подготовки выпускников по специальности 230105.51 «программное обеспечение вычислительной техники и автоматизированных систем».

В вышеназванной работе собраны методические указания и рекомендации для выполнения лабораторных работ что позволяет студенту выполнить работы согласно требованиям .

Данные методические указания дают возможность студенту углубить знания, полученные на лекционных знаниях, развить мыслительную деятельность и способность к принятию самостоятельных решений, что является одной из основных характеристик программиста.

Из вышесказанного ясно прослеживается значимость, актуальность данных методических указаний.

Методические указания выполнены на 64 страницах

Методические указания рекомендуются к изданию необходимым тиражом.

Заместитель директора по
учебно-методической работе С.А. Кузюшин

УТВЕРЖДАЮ
Председатель редакционно-издательского
совета ГОУ ОГУ_____
(ФИО)
«_____» _____ 20 ___ года

Экспертное заключение
на учебно-методический документ

Наименование *Методические указания к лабораторная работам «технология разработки программных продуктов».*

Автор (ы) Дель Л.А.

1 Рассмотрев представленные авторами материалы и документы редакционно-издательский совет института (факультета) *Иформационных тех-Нологи й*провел научную, техническую, учебно-методическую, эргономическую и содержательную экспертизу учебно-методического документа и установил следующее:

1.1 *метод.указания соответствуют виду учебно-методического документа;*
(соответствие виду учебно-методического документа, дисциплине, виду учебного занятия)

1.2 *соответствует рабочей программе по специальности «ПОВТАС» __*
(соответствие рабочей программе дисциплины по направлению к специальности действующему
по дисциплине «технология разработки программных продуктов» для средних
классификатору)
учебных заведений

1.3 *Предназначены для более глубокого изучения теоретического материала,*
(методические достоинства и практическая ценность)
для самостоятельного рассмотрения ряда вопросов, которые представлены в сжатой, доступной для понимания форме_____

1.3 *соответствуют требованиям стандарта СТП 110-01 и нормоконтроля*
(качество содержания и оформления)

2 Заключение

На основании изложенного, редакционно-издательский совет считает, что
Методические указания «технология разработки программных продуктов. »
(учебно-методический документ может или не может быть рекомендован к изданию)
могут быть рекомендованы к изданию на базе колледжа Э и Б ОГУ в … экз.

Председатель редакционно-издательского совета института (факультета)
_____Информационныу технологии_____
(наименование института или факультета)
_____ д.т.н.,профессор А.М.Пищухин
(подпись) (ФИО, уч. степень, уч. звание)

Члены комиссии:
_____д.т.н.,профессор Н. Соловьев
(подпись) (ФИО, уч. степень, уч. звание)
_____д.т.н.,доцент Т.М.Зубкова
_____ст.преподаватель Н.Ю.Юркевская
Нач. отдела автоматизации библиотеки _____