МГУПИ

ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ

ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО
ОБРАЗОВАНИЯ

МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ПРИБОРОСТРОЕНИЯ И ИНФОРМАТИКИ

Кафедра ‘Персональные компьютеры и сети’

В.М.Баканов, В.А.Орлов

Программирование на языке С

УЧЕБНО-МЕТОДИЧЕСКОЕ ПОСОБИЕ

по выполнению лабораторных работ

Москва 2010

УДК 681.3.06

Рекомендовано к изданию в качестве учебно-методического пособия редакционно-издательским советом МГУПИ

Рецензент: профессор, к.т.н. Зеленко Г.В.

Баканов В.М., Орлов В.А.

Программирование на языке С: учебно-методическое пособие по выполнению лабораторных работ. -М.: МГУПИ, 2010. – 59 с.

Рассматриваются вопросы создания программ на языке высокого уровня С. В пособии рассматривается стандарт языка программирования С, соответствующий Kernigan & Ritchie (K&R, 1978).

Пособие может быть использовано студентами для подготовки к выполнению лабораторных, практических и курсовых работ. В качестве среды́ для выполнения предлагаемых в пособии заданий возможно использование как MS DOS или DOS-се́ссии под Windows (достаточно иметь компиляторы Borland Turbo C 2.0 / C++ 3.0), так и Xnix-подобных операционных систем.

Пособие соответствует Государственному стандарту для учебной дисциплины ‘Программирование на языке высокого уровня’.

Содержание

Введение

Язык программирования (ЯП) C (произносится ‘Си’) на сегодняшний день является наиболее популярным и весьма эффективным средством создания компьютерных приложе́ний. Разработанный значительно позднее Fortran’а и Algol’а язык C фактически вытеснил их (и многих других) из программистского обихода вследствие своей универсальности и демократизма стиля программирования. Множество языков программирования ‘заимствовало’ у C стандартные приемы создания программ (чего стоит лишь применение парадигмы #include… в современных диалектах тех же Fortran’а и Pascal’я; сам же классический C с момента создания изменился мало). На основе C-подобного синтаксиса построено множество *специализированных* языков программирования (Java, JavaScript, PHP, диалекты C для программирования микроконтроллеров и др.).

Создание C принято связывать с именем Денниса Ритчи, который в самом начале 70-х г.г. работал в компании Bell Telephone Laboratories над проектом (новой тогда) операционной системы (ОС) Unix. Фактически C и был разработан с целью создания переноси́мого варианта новой ОС (первый вариант Unix был написан на ассемблере для ЭВМ серии PDP-70, а затем перенесен на совмести́мую с ней PDP-11); дальнейшая миграция могла произойти только при переходе к ЯП высокого уровня, компиляторы с которого имелись для каждой аппаратной платформы. Серьезное влияние на C оказали ранее разработанные языки BCPL и B (‘Би’); в результате уже к середине 70-х г.г. Bell Lab. доказала всему миру, что ОС может быть успешно написана на ЯП высокого уровня.

Кла́ссикой описания языка программирования C является книга Брайана Кернигана и Д.Риччи ‘Язык программирования C’, впервые изданная издательством Prentice-Hill в 1978 г. [1]. С тех пор словосочетание ‘K&R’ служит маяком для указания на ‘чистый, истинный - *pure*’ язык C.

Язык программирования C хорошо структурирован, является модульным, компилируемым, универсальным и переноси́мым (на уровне компилятора) и сейчас повсеместно используется для системного и прикладного программирования. Серьезное влияние на распространение языка C среди персональных ЭВМ (ПЭВМ) оказала фирма Borland Int. (лине́йка интегрированных сре́д программирования Turbo C, Turbo C++ 3.0, C++Builder, библиотеки TurboVi-

sion, Object Windows Library, Visual Component Library, Component Library for Cross-Platform).

Язык программирования С является *языком программирования высокого уровня* (ЯПВУ). По определе́нию ЯПВУ – это язык программирования, *поня́тия и структура которого удо́бны для восприятия человеком*. Языки высокого уровня отражают потребности программиста, однако абстрагированы от возможностей конкретной системы обработки данных [2]. Наряду с этим современные интерпретации языка С допускают использование низкоу́ровневых включе́ний кодов ассемблера в исходный С-текст.

В настоящее время компиляторы с языка С являются обязательными для любой ОС (кстати, MS Windows *в основном* написана на С). Дальнейшее развитие С связано с объектно-ориентированным подходом (Бьерн Страуструп, 1979, язык С++, [3]) и созданием концепции визуального программирования (продукты лине́йки Borland C++Builder и др.).

Целью пособия является освоение особенностей построения программ на языке С. Практическое использование этих особенностей будут излагаться по ходу действий. Большая часть упражнений может быть проделана в домашних условиях. В примеры не вошли мощные (однако не столь часто используемые) возможности языка С – вызовы функций с переменным числом аргументов, создание ассемблерных процедур и использование прерываний BIOS’а в С-программах.

1 Лабораторная работа 1. Создание простейшей программы на С с использованием интегри́рованной среды́ фирмы Borland Int.

1.1 Цель работы

Целью работы является создание и отладка простейшей программы на языке С с использованием интегрированной среды́ программирования формы Borland.

1.2 Теоретическая часть

Одни́ми из наиболее удачных продуктов фирмы Borland Int. являются сре́ды разрабо́тки исполня́емых приложе́ний (*Integrated development environment* - IDE) Turbo C 2.0 (максимально соответствующей стандарту K&R, но приго́дной только для создания DOS-приложений) и (фактически перехо́дная от MS DOS к Windows) система Borland Turbo C++ 3.0. Для выполнения заданий данного пособия приго́дны оба варианта.

Интегрированная среда́ программирования Turbo C 2.0 (исполня́емый файл TC.EXE) включает текстовый редактор в стиле знаменитого Borland'овского Side Kick и (вызыва́емые при необходимости) : препроце́ссор, компилятор языка С и редактор свя́зей (*'ли́нкер, линко́вщик'*). Анализатор синтаксических ошибок связан с текстовым редактором и позволяет проводить отладку в интерактивном (согласно последовательности действий: *'подготовка исходного текста → компиляция и редактирование связей → исполнение → анализ и корректировка ошибок'* и да́лее в итерацио́нном ци́кле) режиме; таким же образом корректируются ошибки этапа редактирования связей и выполнения.

Turbo C 2.0 предназначен для создания объе́ктных и/или исполня́емых модулей для операционной системы MS DOS. Она обладает ра́звитыми сре́дствами управле́ния проектами и высокой скоростью компиляции, весьма́ хорошо совмести́ма со стандартом K&R. Комплект Turbo C 2.0 включает препроцессор (файл CPP.EXE), компилятор с командной строкой (TCC.EXE), редактор связей (TLINK.EXE), ассемблер (TASM.EXE) и другие ути́литы, позволяющие в пакетном режиме создавать и отлаживать сложные программы (часто называемые приложе́ниями) на языке С.

Вид экрана ПЭВМ с загруже́нной IDE Turbo C 2.0 приведен на рис.1.1. В верхней части IDE расположены управляющие элементы в виде выпада́ющих меню (см. табл. 1.1).

При редактировании исходного текста полезно помнить, что выделе́ние части текста происходит по Ctrl+K-B / Ctrl+K-K (начало и конец блока соответственно), копирование Ctrl+K-C, перемещение блока Ctrl+K-M и т.д.

Область экрана — Message — служит для вывода сообщений о результатах выполнения действий и списка предупреждений и ошибок. Многие особо часто используемые функции управления сконцентрированы в нижней части IDE и активизируются нажатием одной клавиши (особо ва́жной является клавиша F1, позволяющая получа́ть контекстную помощь).

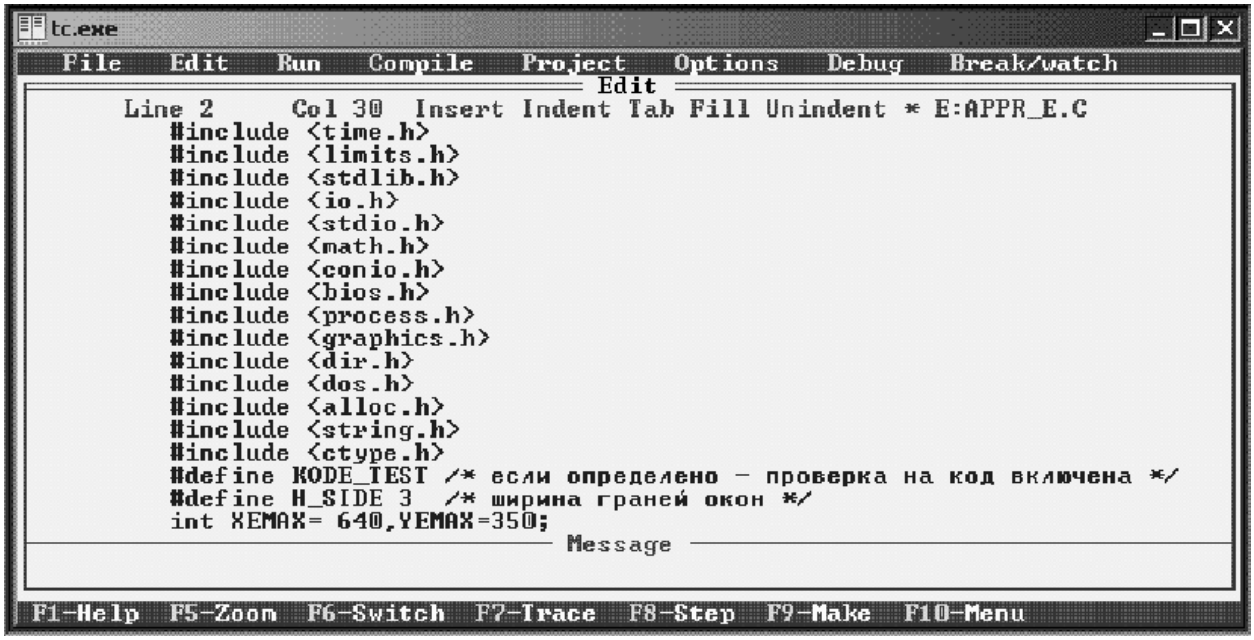

Рисунок 1.1 – Внешний вид экрана ПЭВМ с загру́женной IDE Borland Turbo C 2.0 и файлом исходного текста APPR_E.C

Таблица 1.1 — Элементы пользовательского интерфейса, расположенные в верхней части IDE Turbo C 2.0

Набор клавиш выбора меню	*Название меню*	*Назначение*
Alt+F	**File**	Меню обеспечивает до́ступ к операциям с файлами (создание, открытие, сохранение), выбор каталога, выход в DOS и завершение работы
Alt+E	**Edit**	Переход в режим редактирования файлов
Alt+R	**Run**	Запуск программ на выполнение, отладка файлов и переход к пользовательскому экрану для просмотра результатов вы́дачи программы (*User Screen*)
Alt+C	**Compile**	Компиляция и редактирование связей исходного файла
Alt+P	**Project**	Работа с проектами (специальная возможность, позволяющая создавать программы, состоящие из многих исходных, объектных и библиотечных

		файлов)
Alt+O	**Options**	Конфигурирование проектов и само́й среды отладки.
Alt+D	**Debug**	Управление за́пуском и отладкой программ в интерактивном режиме.
Alt+B	**Break/ watch**	Позволяет устанавливать отслеживание значений переменных и точки останова программы для анализа критических ситуаций.

IDE Turbo C++ 3.0 обладает бли́зким (хотя и несколько усложне́нным по сравнению с Turbo C 2.0) интерфейсом пользователя, позволяет создавать приложения на C++ (вследствие этого имеет *излишнюю для программирования на языке С функциональность*) и может быть использован в данном пра́ктикуме.

Общий формат командной строки компиляции программ в паке́тном режиме следующий:

```
tcc  [режим режим режим…]  имя_файла  имя_файла…
```

Например, для компиляции исходного файла primer.c совместно с объектным файлом primer_add.obj в исполня́емый файл primer.exe следует зада́ть (*квалифика́тор* -IB определяет включа́емый каталог заголо́вочных файлов, здесь -LB суть ката́лог библиотек):

```
tcc  -IB:\include  –LB:\lib  –eprimer  primer.c  primer_add.obj
```

1.3 Необходимое оборудование

Для проведения работ необходима IBM PC-совмести́мый ПК с предустано́вленной интегрированной средо́й программирования фирмы Borland Int.

1.4 Порядок проведе́ния работы

Студент загружает IDE фирмы Borland Int. (в случае работы в Windows загру́зка происходит в DOS-сессии), подготавливает нижеприведенный исходный текст программы на языке С, компилирует ее, выявляет ошибки, запускает на выполнение. Качество выполнения программы проверяется преподавателем.

```
/* исходный текст программы 1_01.c */

#include <stdio.h>
```

```
main()
{
  printf("\nHello, students of (the BEST at MGUPI) kathedra IT-4 !..\n");

  getch(); /* ожида́ние ввода любого символа с клавиатуры */

} /* конец программы */
```

1.5 Описание программы

Для исходного текста на С принят *свободный формат* – отдельные операторы могут начинаться с любой позиции в строке и даже ‘набива́ться’ без перехода со строки на строку (хотя это крайне неудобно с точки зрения чита́емости текста). Любой оператор языка (вычисля́емое выражение, обращение к функции и т.д.) *заканчивается точкой с запятой*. Настоятельно рекомендуется с самого начала освое́ния языка программирования каждый оператор (включая символы-ограничители бло́ков ‘{’ и ‘}’) располагать на отдельной строке, подчеркивая тем самым бло́чную структуру алгоритма путем сдви́га блоков вправо по мере увеличения глубины вложе́ния.

Исходный текст (*‘исхо́дник’* в программи́стской терминологии; далее термины *программи́сткого сле́нга* будут выделя́ться *курси́вом*) начинается обычно с предписа́ния (*директи́вы*) #include (# означает, что это директи́ва *препроцессору* – выполняющейся перед компиляцией ути́литы, совершающей определенные действия). Директива include требует выполнения действий по подстановке указанного файла в место нахожде́ния само́й директивы; в данном случае это файл stdio.h, содержащий прототипы функций ввода/вывода. Без включе́ния этого заголо́вочного (‘*хи́дер*’а в англоязычной нота́ции русскоязы́чного термина *заголо́вок*) файла функция вывода данных printf работать не сможет, ибо не является включе́нной в язык С, а должна компоноваться с соответствующим библиотечным файлом на этапе редактирования связей (‘*линко́вки*’). Заключе́ние имени заголо́вочного файла в угловые скобки предпи́сывает препроцессору искать его в стандартном каталоге *хи́деров* (обычно это ./INCLUDE). Допустимо применение двойных кавычек (в этом случае поиск файла осуществляется в текущем каталоге). Реальные программы на С часто содержат десятки *хи́деров*, поддерживающих функции ввода/вывода, математические, работы с памятью и многие другие.

Кроме директив, являющихся указаниями препроцессору и игнорируемых компилятором, имеются (форма́льно похожие на них) инструкции компилятору – *пра́гмы*. Например, пра́гма #pragma inline сообщает компилятору, что программа содержит вста́вки на языке ассемблера.

В языке С изначально определены пять *пото́ков байтов* - стандартного ввода stdin, вывода stdout, вывода ошибок stderr, стандартный поток вывода на параллельный порт stdprn (обычно свя́зывается с принтером) и стандарт-

ный поток вывода на последовательный порт stdaux). В DOS'е stdout, stderr связаны с экраном терминала, stdin – с клавиатурой; в Windows подобного соответствия нет.

Данная программа состоит всего из одной написанной программистом функции (признаком функции – в *отличие от переменной* – является обязательное *наличие круглых скобок* после имени) с *предопределенным* именем main (такая функция должна обязательно присутствовать в программе и всегда *выполняется первой*); тело функции (декларации объектов программирования и составляющие функцию операторы) заключается в фигурные скобки.

Формально определение функции следующее (находящееся в угловых скобках необязательно):

тип_возвращаемого_значения имя_функции (<формальный_параметр_1>, <формальный_параметр_2>,<формальный_параметр_3>..);

где тип_возвращаемого_значения может принимать значения допустимых типов данных – int, float, double, struct_тип и т.п. (если возвращаемое значение несущественно, используется ключевое слово void),
формальный_параметр_1 – тип и имя формального параметра функции (если число и тип формальных параметров не конкретизированы, вместо их списка ставится *эллипсис* – '**...**').

Заметим, что в этом смысле описание функции main в приведенном примере упрощено (хотя и допустимо), полные возможности main будут описаны в последующих примерах. Описания (в т.ч. пользовательских) функций должны следовать *до вызова самой функции*; часто описания выносятся в отдельный заголовочный файл (*хидер*) с расширением '.h' ('.hpp' для C++).

Комментарии могут занимать несколько строк между ограничителями '/*' и '*/' ; для C++ комментарием является также и все, что следует за '//' (*двойной слэш*) – до конца строки). Хороший программист не стесняется писать как можно более обширные комментарии, что помогает разобраться в программе (комментарии не сохраняются в исполняемом файле и нисколько не увеличивают его объем).

Параметр (первый) оператора printf в данном случае фактически *является строкой* и заключается в двойные кавычки, перед специальными (*управляющими*) символами указывается символ обратного слэша '\'; например, последовательность '\r' означает *единственный символ* 'Возврат каретки' (код 13_{10}), '\n' – символ 'Новая строка' (10_{10}); сам символ '*обратный слэш*' должен выглядеть как '\\'. Любой символ может быть представлен в виде последовательности трех восьмеричных цифр '\ddd'.

Язык С *регистрозависим* – функций PRINTF и Printf не существует, а переменные First, first и FIRST являются различными.

После вывода текста на экран с помощью функции printf программа не заканчивается, а ожидает нажатия любой клавиши на клавиатуре вследствие выполнения функция getch (эта функция возвращает в программу код нажатой клавиши, в данном случае он несуще́ственен).

Перед компиляцией следует в меню Options → Directories корректно настро́ить рабочие ката́логи IDE. Если среда́ программи́рования (ежели она tc_2) установлена в каталог X:\tc_2, то значение Include Directories должно быть X:\tc_2\include (каталог заголо́вочных файлов), Library directories - X:\tc_2\lib (ката́лог библиотечных файлов), Output directory - X:\tc_2\work (объектные и исполня́емые файлы), Turbo C directory - X:\tc_2 (можно задать несколько путе́й через точку с запятой, указанные каталоги должны существовать и соответствовать своему назначению). Имя компилируемого файла следует зада́ть посредством Compile → Primary C file**.** После изменения настроек следует их сохрани́ть (выполнить Options → Save Options).

Компиляция и редактирование связей выполняется посредством Compile → Build All или просто F9; признаком успешной компиляции служит сообщение Success во всплывающем окне Making (в противном случае сообщения об ошибках выводятся в область — Message — главного экрана; переключение между областью редактирования и ошибок выполняется посредством клавиши F6). После успешной компиляции в каталоге Output directory появятся два файла с именем, соответствующем исходному, но с расшире́нием .OBJ (объектный) и .EXE (исполняемый).

Запуск (исполнение) программы осуществляется Run → Run или Ctrl+F9; для просмотра пользовательского экрана (скры́того под IDE) служит Alt+F5 (возврат в IDE – любая клавиша).

1.6 Вопросы для самопроверки

1. Из каких стадий состоит процесс получения исполня́емого файла?
2. Какие компоненты входят в состав IDE?
3. Что такое ‘препроцессор’ и каким образом обозначаются директивы управления им?
4. Для чего служат файлы, содержащие заголовки функций (файлы-*хи́деры*)?
5. Назвать стандартные файлы ввода/вывода. С какими внешними устройствами они (по умолчанию) связаны в DOS?
6. Каким образом обозначаются специальные (управляющие) символы?

2 Лабораторная работа 2. **Типы данных в языке С, массивы, область ви́димости, обработка строк и форматный вывод данных**

2.1 Цель работы

Целью работы является создание и отладка программы на языке С, *формально* использующей различные типы данных (в т.ч. объединенные в массивы), указатели, обработка строк, форматный вывод данных.

2.2 Теоретическая часть

В языке программирования С любой *идентификатор* является ***именем*** *переменной, функции* или *ме́тки* и может состоять из символов [a-z, A-Z, 0-9, подче́ркивание] в количестве до 32, причем первый символ должен быть только буквой или символом подчеркивания. Присвоение объектам программирования имен (идентификаторов) и типов данных называется ***декларацией***. Декларация должна быть описана ранее любых исполня́емых операторов программы, где эти объекты используются (обычно в начале функции или программы на языке С).

Основные типы данных приведены в табл.2.1.

Таблица 2.1 — Основные типы данных языка С для персональных ЭВМ на INTEL-совместимых процессорах

Тип данных	*Описание*
char	Символьная переменная (1 байт = 8 бит) может храни́ть любой символ из допусти́мого списка; символ представляется целым (диапазон от –128 до 127), соответствующим его коду. Тип unsigned char имеет диапазон 0-255.
int	Целое число со знаком (2 байта для Turbo C, диапазон от –32768 до 32767). Возможны short int и long int и (беззна́ковые) unsigned int, unsigned short, unsigned long.
float	Число с плавающей точкой одина́рной точности со знаком (4 байта, диапазон по модулю приблизительно от 3.14e–38 до 3.14e+38, точность – около 6–7 значащих цифр).
double	Число с плавающей точкой двойной точности со знаком (8 байт, диапазон по модулю приблизительно от 1.7e–308 до 1.7e+308, точность – около 14-15 значащих цифр).

От того, насколько проду́мано будут выбраны идентификаторы, зависит возможность ее понима́ния и модификации (и, в общем случае, надежность). Рекомендуется при назначе́нии идентификаторов стараться использовать имена, максимально близкие к именам переменных в *математической запи-*

си алгоритма (напр., для индексных переменных целесообразно выбирать i, j, k и т.д.).

Более полезный метод (т.н. *венге́рская нотация*) объявле́ния идентификаторов предложил Чарльз Симони (*Charles Simonyi*) - легендарный программист фирмы Microsoft Corp., дважды (2006 и 2009 г.г.) ‘космический турист’ на Международную Космическую Станцию – начина́ть имя с *пре́фикса* на нижнем регистре, который *несёт информацию о типе переменной* (это сильно упрощает понимание программы, явно указывая тип данных, с которыми данный идентификатор связан). Например, fnAlpha – функция с именем Alpha, sMyName – строка, iCounter – целое, cState – символ. Переменные при объявлении могут быть инициализированы.

Стро́ки хранятся как последовательность байт, причем имя строки является *адресом первого* (нулевого по счету) байта; строка может иметь любую длину, конец строки отмечается нулем (‘\0’) в последнем байте.

В некоторых случаях требуется явно (по указанию программиста) изменить тип данных при вычислениях (операция *приведе́ния типа*). Формат этой операции включает указание нового типа (в скобках) перед переменной (напр., int i; float f; i=(int)f;). В большинстве случаев С-компилятор выполняет приведе́ние типов автоматически по следующему правилу – типы переменных в сложном (включающем различные типы данных) выражении приво́дятся к наиболее сложному из присутствующих (напр., int к float). Во избежание недоразумений рекомендуется не забывать о типе данных даже при использовании оператора присвое́ния (напр., int i=5; float f=4.0e5; но не int i=5.0).

Массивы (массивом называется *группа элементов одного типа*) описываются в формате тип_данных имя_массива размер, где размер определяется значением в квадратных скобках (напр., декларация float a3[4]; объявляет a3 одномерным массивом вещественных чисел из 4-х элементов, int a4[10][11][12]; - трехмерный массив целых чисел из 1’320 элементов). Нумерация (индексация) элементов массивов в С/С++ всегда *начинается с нуля*. Элементы многоме́рных массивов в С/С++ всегда располагаются в оперативной памяти последовательно (линейно).

Близким к одноме́рным массивам типом данных являются перечисле́ния (объявление перечисле́ния начинается с ключевого слова enum).

В отличие от обычных переменных *указа́тели* представляют собой переменные, значения которых являются *адресами.* Признаком указателя при его декларации является символ ‘звездочка’ (‘*’) перед его именем. Например, декларации char *msg; и float *a3 объявляют указатели msg и a3 на переменные типа char и float, соответственно. Заметим, что *a3 является ***значением*** переменной типа float , имеющей адрес a3.

Над указателями определены 4 арифметических действия; следует только помнить, что эти манипуляция происходят с *адресами переменных* в оперативной памяти, а не с *самими переменными*. Существует обра́тная к ‘*’ *операция взя́тия адреса* переменной, признаком которой является символ ‘&’ перед именем переменной (например, &a_43 является *адресом переменной* a_43).

Класс памяти определяет способ обраще́ния к соответствующему объекту из других частей программы. Главное назначение класса памяти – разграниче́ние до́ступа к переменным из функций и отдельных файлов, которые может входить в С-программу. *Локальные переменные* (описываются и используются в функциях, автоматически создаются в момент входа управления в функцию и так же уничтожаются при выходе из нее (явным образом квалификатор auto указывать излишне). *Глобальные переменные* (глоба́лы) описываются до момента описания всех функций и одинаково доступны внутри функций (естественное применение глобальных переменных – обмен данными между функциями). Классы переменных не пересекаются – локальная переменная нико́им образом не взаимодействует с глобальной того же имени и является приоритетной внутри функции (изменения ее не затрагивает глобальную). Добавление квалификатора external делает локальную переменную внешней (т.е. имеющей область действия вне функции и всех файлов исходного текста). Описанные с ключевым словом static переменные являются внешними статическими и могут использоваться только функциями то же самого файла. Квалификатор register требует (если это возможно) от компилятора разместить переменную в одном из регистров процессора, что сильно ускоряет скорость до́ступа к ней. Размер локальной памяти внутри функций ограничен размером стека, поэтому очень длинные (по занимаемой памяти) данные целесообразно описывать как внешние.

Обобщением понятия массива являются *структуры* (struct) – объекты, объединяющие *элементы различных типов*. До́ступ к отдельным элементам осуществляется в форме ‘имя_структуры . имя_элемента’ или ‘указатель_на_структуру -> имя_элемента’. *Сме́си* (union) представляют собой утри́рованные структуры, все элементы которой перекрыва́ют друг друга (в каждый момент в смеси может храниться только одно значение); длина смеси равна максимальной длине составляющих смесь элементов. Структуры и смеси могут объединяться в массивы.

Структура может являться элементом *свя́зного списка*. В языке С структура не может содержать в качестве своего элемента структуру такого же типа, однако может включать *указатель на структуру* этого типа (при объявлении структуры необходимо указать имя и состав ее элементов (шаблон)). Например:

```
struct Info {
   char FIO[60];
   int Old;
   struct Info *p_left;
   struct Info *p_right;
   }
```

Присва́ивая указателю p_left ссылку на предыдущую в списке структуру Info, а указателю p_right – на следующую, получаем *кольцево́й список* с возможностью простого перемещения по объектам списка. Для организации *линейного списка* обычно присваивают p_left=NULL первой структуре в списке и p_right=NULL – последней (значения NULL проверяются при обращении к данным текущей структуры в списке).

С точки зрения программиста бывает удобно создавать короткие и более понятные имена для типов данных, уже определенных в языке С или более сложных, объявленных пользователем. Для этого используется ключевое слово typedef ; объя́вленный (опи́санный) с этим словом идентификатор становится новым именем типа и далее в программе мо́жет использоваться в качестве *спецификатора* (формально ничем не отлича́ясь от int, float, char) такого типа данных. Например, объявле́ние typedef char FIO [60] позволяет в дальнейшем использовать описа́ние данных в виде FIO persons[100], которое объявляет массив данных из 100 элементов типа FIO (что с точки зрения *обеспечения структурированности* определения данных логичнее, нежели описание char persons[60][100]).

В С имеется ра́звитый механизм форматирования данных при выводе (это необходимо для корректного вида печати). Можно задать число позиций для вывода числа, количество знаков после запятой, вывод данных в десятичной или 16-тиричной форме и т.д. Формат вывода данных определяется *строкой формата*, поэтому фактически формат функции printf следующий (строка формата определяет, в *каком виде* печатать данные объект_1, объект_2 и т.д.):

```
printf(строка_формата, объект_1, объект_2…);
```

В строке_формата должны находиться *спецификации формата*, начинающиеся со знака процента (см. табл. 2.2), которые *последовательно* применяются к объектам форматирования.

Таблица 2.2 — Основные спецификации формата вывода языка С

Спефикация формата	*Описание*
%u	Целое число без знака
%d	Целое число
%f	Число с плавающей точкой

%e	Число с плавающей точкой в экспоненциальной форме (т.н. *научная нота́ция* – с порядком)
%c	Единичный символ
%s	Строка символов
%x	Число в 16-тиричной форме
%p	Значение указателя

Для большинства спецификаций формата можно задать число позиций, в которых будет помещено число – напр., %7d заставит *целое* печататься в 7 позициях, %8.3f – число с плавающей запятой в 8 позициях с 3-мя знаками после запятой (округление при выводе происходит по стандартным правилам). По умолчанию выводи́мое число ‘прижа́то’ к правой стороне поля вывода; для реализации обратного случая следует использовать символ *‘–’ (минус* или *дефи́с*) после ‘%’ (напр., спецификация поля ‘%–20.3e’ требует вывода вещественного числа в ‘научном’ формате с 3-мя цифрами после запятой, прижа́тому к *левой* стороне 20-ти символьного поля).

2.3 Необходимое оборудование

Для проведения работы необходима IBM PC-совмести́мый ПК с предустано́вленной интегрированной среды́ программирования фирмы Borland Int.

2.4 Порядок выполнения работы

Студент загружа́ет IDE фирмы Borland Int. (в случае работы в Windows загрузка происходит в DOS-сессии), подготавливает нижеприведенный исходный текст программы на языке С, компилирует ее, выявляет ошибки, запускает на выполнение. Качество выполнения программы проверяется преподавателем.

```
/* исходный текст программы 1_02.c */

#include <stdio.h>
#include <string.h>

int i1=1; /* i1 расположена во внешней памяти */

main()
{
 int i1=2;   /* локальная для функции MAIN целая переменная
                (автоматический класс памяти) */

 char *msg, /* указатель на объект типа char */
        fmt[120]; /* строка формата */

 static char string_1[80]="I am static string"; /* строковая переменная длиной
```

```
                                        не более 80 символов во внешней памяти */

 float array_1[4]={1.0, 2.0, 3.0, 4.0};    /* инициируемый значениями одномерный
                                              массив из 4 'плавающих' чисел */
 int matr_1[10][20];  /* двумерный массив, который соответствует матрице из 10
строк и  20 столбцов */

 struct complex
 {
  float real;
  float imag;
 } a1,  /* структура a1 типа complex */
 *a2=&a1; /* a2 суть указатель на структуру a1 типа complex */

 i1=i1+1;

 matr_1[2][3]=13; /* присвоение значения элементу i=2, j=3 матрицы matr_1 */

 a1.real=2.0;   /* целая часть комплексного числа a1 */
 a1.imag=3.0; /* мнимая часть комплексного числа a1 */

 msg="Hello, students of kathedra IT-4 !";

 printf("\n\n\ni1=%3d string_1=%s\n", i1, string_1);

  strcpy(fmt, ""); /* очистка строки fmt */
  strcat(fmt,"\na1=%6.3f+%6.3fi a2=%6.3f+%6.3fi\naddress(a1)=%p");
  strcat(fmt," address(a2)=%p\nsize(a1)=%d bytes size(a2)=%d\n");
  printf(fmt, a1.real,a1.imag, a2->real,a2->imag, &a1,a2, sizeof(a1),sizeof(a2));

 printf("\nmsg= %s\naddress(msg)=%p size(msg)=%d bytes\n",
         msg,msg,sizeof(msg));

 printf("\narray_1[3]=%.3e\naddress(array_1)=%p size(array_1)=%d bytes\n",
          array_1[3], array_1, sizeof(array_1));

 printf("\nmatr_1[2][3]=%3d\naddress(matr_1)=%p size(matr_1)=%d bytes",
          matr_1[2][3], matr_1, sizeof(matr_1));

 printf("\n\n\nВведите любой символ для продолжения...\n\n");

 getch(); /* ожида́ние ввода любого символа с клавиатуры */

}  /* конец программы */
```

1.5 Описание программы

При изучении данной программы следует обратить внимание на описание идентификатора i1 в начале программы (внешняя переменная) и внутри функции main; из полученного при выводе значения i1 предельно ясно, что локальная переменная имеет приоритет перед глобальной.

Представляет интерес описание конкретных идентификаторов структуры типа complex:

```
 struct complex
  {
   float real;
   float imag;
  } a1,  /* структура a1 типа complex */
   *a2=&a1; /* a2 суть указатель на структуру a1 типа complex */
```

Здесь a1 – имя структуры типа complex, a2 – *указатель на структуру* типа complex (при объявлении идентификатора a2 ему присваивается адрес идентификатора структуры a1).

Целая и мни́мая (a1.real и a1.imag соответственно) части структуры a1 типа complex выводятся отдельно по формату %6.3f каждая (в случае адреса́ции структуры указателем a2 используется запись a2->real и a2->imag). Данные выводятся в соответствие с форматной строкой fmt, функция strcpy копирует второй формальный параметр в первый, strcat конкатени́рует (*соединяет*) второй формальный параметр с первым (требуется включение файла заголовков string.h). Как видно из вывода программы, размер (возвращается выполняемым *на этапе компиляции* оператором sizeof) структуры a1 равен 8 байтам (два float-числа), размер же a2 равен 4 байтам. Структура типа complex удо́бна для хранения комплексных чисел; пользователь может разработать ряд функций работы с комплексными числами, представленными в подобной форме.

Переменная msg описана как *указатель на символ* (т.е. может хранить 4-х байтовый *адрес* некоторого символа), но при этом компилятор не выделяет никакого пространства для размещения символов и не инициирует msg конкретным значением. Оператор msg=”Hello, students of kathedra IT-4 !\n”; создает заданную строку в свободной части памяти (в данном случае - локальной), заканчивает ее нулевым символом и присваивает начальный адрес этой строки переменной msg. Адрес первого символа msg выводится на печать по формату %p в виде двух восьмеричных чисел.

Как и следовало предположить, размер массива array_1 равен 16 байтам (4 float-числа), размер массива matr_1 равен 400 байтам (200 целых чисел).

Строка string_1 описана как массив длиной 80 символов с квалификатором static – т.е. расположена во внешней памяти и будет доступна другим функциям С из того же файла.

2.6 Вопросы для самопроверки

1. Какие типы данных в языке С Вы знаете? С какой точностью может быть представлено целое число (число с плавающей точкой)?
2. Что такое приведе́ние типов данных? По какому правилу приво́дятся типы данных в сложном выражении?
3. Что такое указатель? Как соотносятся между собой операции ‘*’ и ‘&’?
4. Какие классы памяти и для чего определены в языке С?
5. В чем состоит различие обращения к элементам структуры в случаях использования ее имени и *указателя* на нее?
6. Каким образом располагаются в памяти ЭВМ элементы многоме́рных матриц в С-программах?
7. Что такое ‘*строка формата*’ при выводе данных? Каким образом указывается размер поля при выводе?

3 Лабораторная работа 3. **Управление последовательностью выполнения вычислений**

3.1 Цель работы

Целью работы является создание работоспособной программы на языке С, использующей операторы управления последовательностью вычислений (рассматривается на примере практической задачи – умножение матриц).

3.2 Теоретическая часть

Практически в любом ЯПВУ имеется набор операторов для управления последовательностью вычислений. Простейшими из подобных операторов являются операторы безусловного или условного передачи управления goto. В языках С/С++ структура этого оператора записывается так (в угловых скобках – необязательная часть):

```
<if(условие)>
  goto label;
…любая_последовательность_операторов…
<else>
…любая_последовательность_операторов…

label:
…любая_последовательность_операторов…
```

Здесь условие – последовательность операторов, результатом выполнения которой является число = 0 (интерпретируемое как false – *ложь*) или #0 (интерпретируемое как true - *истина*); в последнем случае происходит безусловная передача управления в точку label (здесь label – уникальный в пределах данной функции *идентификатор ме́тки*), передача управления с помощью goto → label возможна лишь в пределах одной функции. При невыполне́нии условия в if происходит переход к else (после которого могут находиться любые операторы).

Некоторые теоретики программирования настоятельно не рекомендуют использовать goto, мотивируя излишнюю сложность и запутанность (т.н. ‘*макаро́нный стиль*’) получаемой таким образом программы. Однако в реальной жизни часто именно goto существенно упрощает программу. Однако необходимо предупредить о ‘подводных камнях’ применения goto – например, в случае попытки передачи управления внутрь гнезда циклов или составного оператора (см. ниже).

В качестве логического выражения условие может выступать сколь угодно сложная комбинация *логических выражений*, объединенных логическим ‘И’ (&&), ‘ИЛИ’ (||) и др. Не следует забывать, что символом ра́венства в С/С++

является ‘==’ (а нера́венства – ‘!=’); инструкция a=b с точки зрения С-компилятора суть *присва́ивание, а не ра́венство* !

Если при ветвлении программы число возможных исходов априори известно и невелико, удобно употреблять оператор switch (в нижеприведенном примере ‘переключение’ происходит по значению переменной i):

```
switch(i)
{
 case 0: …операторы, выполняющиеся при i=0…
          break;
 case 1: …операторы, выполняющиеся при i=1…
          break;
 case 2: …операторы, выполняющиеся при i=2…
           break;
 ……..
 default: …операторы, выполняющиеся во всех остальных случаях…
}
```

В данном случае в конце каждой исполняемой ветви находится оператор break, приводящий к выходу из конструкции switch. Если убрать break перед case 1, то после окончания выполнения ветви i=1 выполнится и ветвь i=2 (и так далее).

В некоторых случаях вместо if удобно использовать *условное выражение*, имеющее следующую форму:

```
выражение_1 ? выражение_2 : выражение_3;
```

В первую очередь вычисляется выражение_1. Если результат имеет ненулевое значение (*истина*, true), то значением условного выражения является выражение_2. Если значение выражение_1 равно нулю (*ложь*, false), то то значением условного выражения является выражение_3. Например, нахождение наибольшего из двух чисел x и y может быть записано так: max=(x>y)?x:y; Поиск абсолютной величины любого числа: abs=(x>0)?x:-x; замена наибольшего из двух чисел единицей: (x>y)?x=1:y=1;

Мощным оператором является *оператор цикла* for, формат которого тако́в (если …операторы_тела_цикла … состоит всего лишь из одного оператора, фигурные скобки могут быть опущены):

```
for(инициализация; условия; изменение)
{
 …операторы_тела_цикла…
}
```

Здесь инициализация используется для установки начального значения управляющей переменной цикла (можно инициализировать любое их число,

разделяя операторы запятыми или применять любые иные допусти́мые операторы – следует помнить, что инициализация *выполняется один раз* перед выполнением тела цикла); условия (любое выражение, значение которого может быть *преобразовано к логическому типу*) определяет условия невыхода из цикла (их также может быть несколько, причем отдельные выражения разделяются запятой); изменение задает изменение управляющих переменных цикла (также может быть несколько). Любые из описанных трех групп могут быть опущены при сохранении точки с запятой в качестве разделителя - напр., for(;;) задает бесконечный цикл. Общая схема выполнения for такая:

- вычисляется выражение инициализация,
- вычисляется выражение условия,
- если значение условия # 0 (true - *истина*), выполняются …операторы_тела_цикла…
- вычисляется изменение,
- заново вычисляется выражение условия,
- если условия = 0 (false – *ложь*), управление передается первому следующему за for оператору (осуществляется *'выход из цикла'*); в противном случае цикл повторяется.

Огромная мощь оператора цикла for заключается в возможности включения в состав любой их трех групп любого количества операторов, *разделяемых запятыми* (для разделяемых символом 'запятая' операторов *гарантируется последовательность выполнения их слева направо,* в качестве конечного результата принимается *результат самого правого выражения*); при этом часто бывает непросто понять функционирование цикла. Внутри тела оператора for могут быть использованы continue (принудительный переход к следующей итерации цикла) и break (принудительное окончание цикла).

Гнездо́м циклов называется конструкция, включающая несколько вло́женных циклов, напр., классическое умножение квадратных матриц [C]←[A]×[B] размерностью n требует гнезда циклов с *глубиной вло́женности* 3 (выполняемые в цикле операторы составляют *те́ло цикла*):

```
for(i=0; i<n; i++)
  for(j=0; j<n; j++)
    for(k=0; k<n; k++)
      c[i][j] += a[i][k] * b[k][j];
```

В языке C имеется и другой оператор цикла while (…операторы… выполняются в случае условие # 0, причем проверка этого условия производится *до выполнения* тела оператора while – это *цикл с предусло́вием*):

```
  while(условие)
```

```
{
 …операторы…
}
```

Имеется также оператор do (…операторы… выполняются минимум один раз и при условие # 0, т.е. это *цикл с постусло́вием*):

```
do
{
  …операторы…
}
while(условие);
```

В случае подстановки на место условие значений true (или любого *ненулевого числа*) циклы while и do становятся бесконечными.

Чаще всего используются *циклы с предусло́вием*, *циклы с постусло́вием* (do… while) составляют (по оценкам K&R) всего 5% общего числа используемых.

Оператор exit (status) позволяет немедленно выйти из программы с кодом завершения status (перед завершением программы все файлы закрываются с записью буферов на внешнее устройство). Оператор return (value) инициирует возврат из данной функции с возвращаемым значением value (конечно, тип этой переменной должен быть согласован с возвращаемым значением в описании функции в соответствующем файле заголовков).

3.3 Необходимое оборудование

Оборудованием является IBM PC-совмести́мый ПК с предустановленной интегрированной средой программирования Borland Int.

3.4 Порядок проведения работы

Студент загружает IDE фирмы Borland Int. (в случае работы в Windows загрузка происходит в DOS-сессии), подготавливает нижеприведенный исходный текст программы на языке С, компилирует ее, выявляет ошибки, запускает на выполнение. Качество выполнения программы проверяется преподавателем.

Заданием является перемножение неквадратных матриц [A] и [B] с помещением результата в [C]. При классическом методе умножения матриц значение элементов результирующей матрицы вычисляется как:

$$c_{ij} = \sum_{k=1}^{k=NCA} a_{ik} b_{kj}, \qquad (3.1)$$

где $i=1 \div NRA$ – строки матриц [A] и [C],
$j=1 \div NCB$ – столбцы матриц [B] и [C],
$k=1 \div NCA$ – столбцы матрицы [A] и строки матрицы [B].

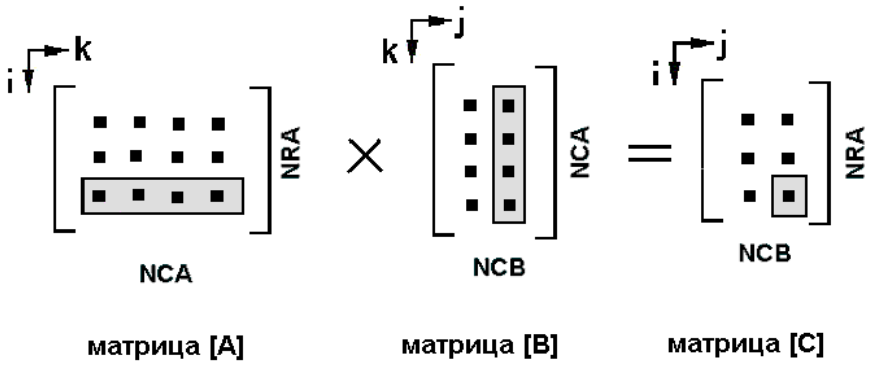

Рисунок 3.1. Умножения матриц по классической схеме
(на примере вычисления значения $c_{32} = \sum_{k=1}^{k=4} a_{3k} \times b_{k2}$)

С-реализация формулы (3.1) и схемы рис.3.1 (для всех значений NCA) приведена ниже:

```
/* исходный текст программы 1_03.c */

#include <stdio.h>

#define NRA  3 /* число строк матриц [A] и [C] */
#define NCB  2 /* число столбцов матриц  [B] и [C] */
#define NCA  4 /* число столбцов [A] и строк [B] */

main()
{
 float a[NRA][NCA],  /* описание рабочих массивов */
       b[NCA][NCB],
       c[NRA][NCB];

 int i,j,k; /* целые переменные для индексации элементов матриц */

again: /* метка для возврата к вводу матриц [A] и [B] */

/* ввод элементов матрицы [A] по строкам */
  for(i=0; i<NRA; i++) /* цикл по строкам [A] */
   {
    if(i==0) /* первая строка матрицы [A] */
    {
```

```
      printf("\n\nВведите 4 вещественных числа 1-й строки матрицы [A]…\n\n");
      scanf("%f %f %f %f", &a[0][0], &a[0][1], &a[0][2], &a[0][3]);
     }
     else
     if(i==1) /* вторая строка матрицы [A] */
     {
      printf("\n\nВведите 4 вещественных числа 2-й строки матрицы [A]…n\n");
      scanf("%f %f %f %f", &a[1][0], &a[1][1], &a[1][2], &a[1][3]);
     }
     else
     if(i==2) /* третья строка матрицы [A] */
     {
      printf("\n\n Введите 4 вещественных числа 3-й строки матрицы [A]…\n\n");
      scanf("%f %f %f %f", &a[2][0], &a[2][1], &a[2][2], &a[2][3]);
     }
   } /* конец цикла for(k=0; k<NCA; k++) по строкам матрицы [A] */

/* ввод элементов матрицы [B] по строкам */
  for(k=0; k<NCA; k++) /* цикл по строкам [B] */
   {
     if(k==0) /* первая строка матрицы [B] */
     {
      printf("\n\nВведите 2 вещественных числа 1-й строки матрицы [B]…\n\n");
      scanf("%f %f", &b[0][0], &b[0][1]);
     }
    else
    if(k==1) /* вторая строка матрицы [B] */
     {
      printf("\n\n Введите 2 вещественных числа 2-й строки матрицы [B]…\n\n");
      scanf("%f %f", &b[1][0], &b[1][1]);
     }
    else
    if(k==2) /* третья строка матрицы [B] */
     {
      printf("\n\nВведите 2 вещественных числа 3-й строки матрицы [B]…\n\n");
      scanf("%f %f", &b[2][0], &b[2][1]);
     }
    else
    if(k==3) /* четвертая строка матрицы [B] */
     {
      printf("\n\n Введите 2 вещественных числа 4-й строки матрицы [B]…\n\n\");
      scanf("%f %f", &b[3][0], &b[3][1]);
     }
   } /* конец цикла for(j=0; j<NCB; j++) по строкам матрицы [B]  */

  printf("\n\nВсе данные введены? (Y/*)\n\n");

  if(getch()!='Y') /* подтверждение правильности ввода */
   goto again;
```

```
/* собственно умножение матриц */
  for(i=0; i<NRA; i++) /* цикл по строкам [A] и [C] */
   for(j=0; j<NCB; j++) /* цикл по столбцам [B] и [C] */
    {
      c[i][j] = 0.0; /* обнуление cij */
      for(k=0; k<NCA; k++)
       c[i][j] += a[i][k] * b[k][j];
    }

/* вывод рассчитанных данных */
  for(i=0; i<NRA; i++) /* цикл по строкам матрицы [C] */
    switch(i) /* вывод в зависимости от номера строки [C] */
    {
     case 0: printf("\n\nЗначения элементов матрицы [C]:\n\n %10.3e  %10.3e\n",
                        c[0][0], c[0][1]);
                break;
     case 1: printf("%10.3e  %10.3e\n", c[1][0], c[1][1]);
                break;
     case 2: printf("%10.3e  %10.3e\n", c[2][0], c[2][1]);
                break;
    } /* конец  switch(i) */

  printf("\n\nЖелаете закончить работу (Y/*)\n\n");

  if(getch()=='Y') /* подтверждение продолжения работы */
   goto again;

} /* конец программы */
```

3.5 Описание программы

При изучении данной программы следует обратить внимание на операторы типа i++, j++. В C/C++ вместо записи i=i+m допускается i+=m (соответственно a*=4; b/=5.0e0 и т.д.); i++ равносильно i=i+1. Заметим, что i++ (*постфи́ксная* запись) в некоторых случаях отлична от ++i (*пре́фиксная* запись) – в первом случае текущее значение переменной i применяется по назначению и потом инкрементируется, во втором – сначала инкрементируется и потом уже используется как инкрементированное (ср., напр. a[i++] и a[++i] – это обращение к разным элементам массива a).

При вводе данных с клавиатуры используется функция sscanf со стандартными параметрами форматирования, однако в качестве ссылок на вводи́мые переменные *следует задавать их адреса́* (т.е. &a[0][0], &b[1][0] и др.).

3.6 Вопросы для самопроверки

1. Назвать известные операторы изменения естественного порядка следования вычислений.
2. По какой причине некоторые теоретики от программирования не рекомендуют использовать оператор goto?
3. В каких случаях удобно использовать конструкцию if… else if… else if… else, а в каких switch… ?
4. Каким образом вычисляется значение условия, если имеется несколько (разделенных запятой) отдельных условных выражений в операторе for?
5. ?Каким образом каждое из трех полей оператора if управляет выполнением циклов? Как записывается ‘бесконечный’ оператор if?
6. В каком случае удобнее использовать цикл for? В каком – while? Когда – do… while? Перепишите заданные преподавателем циклы while... и do… while в виде эквивалентного им цикла for.
7. Что такое гнездо циклов? Как определяется глубина гнезда? Как (в общем случае) определить число выполнений тела самого внутреннего цикла?

4 Лабораторная работа 4**. Работа с внешними носителями информации**

4.1 Цель работы

Целью является создание и отладка программы на языке С, использующей операторы ввода/вывода информации на внешние (дисковые) носители информации. Применение операторов ввода/вывода информации иллюстрируется программой, выполняющей функции простой системы управления базой данных (СУБД).

4.2 Теоретическая часть

Как было сказано, в языке С изначально определены пять *пото́ков байтов* - стандартного ввода stdin, вывода stdout, вывода ошибок stderr, стандартный поток вывода на параллельный порт stdprn и стандартный поток вывода на последовательный порт stdaux. В DOS'е по умолчанию stdout и stderr связаны с экраном терминала, stdin – с клавиатурой, stdprn – с принтером, stdaux – с последовательным портом.

Программист оперирует с потоками посредством указателей на (предопределе́нную) структуру типа FILE, содержащую всю необходимую информацию для работы с потоками.

Стандартными действиями работы с потоками в языке С является следующая последовательность:

```
#include <stdio.h> /* хи́дер описаний процедур работы с потоками (файлами) */
……
……
FILE *f_ptr;    /* указатель на структуру типа FILE */
……
if((f_ptr = fopen(…)) == NULL) /* при успешном открытии fopen возвращает #NULL */
  printf("Ошибка открытия…\n");
……
… работа с файлом, заданным указателем f_ptr …
……
fflush(f_ptr); /* запись (сбро́с) всех данных из буфера потока f_ptr в файл */
fclose(f_ptr); /* закрытие файла, заданного указателем f_ptr */
……
……
fflushall(); /* сбро́с данных из буферов всех потоков в файлы */
fcloseall(); /* закрытие всех файлов */
……
```

Файл открывается оператором fopen(…), который возвращает указатель типа FILE (в данном случае это f_ptr) или 0 (NULL, false) при неудаче открытия; в дальнейшем этот указатель используется в качестве идентификатора при

всех операциях с файлом. Файл закрывается вызовом fclose(f_ptr), перед закрытием (а также после вывода критичных в случае потери данных) рекомендуется принудительно ‘сброси́ть’ в файл внутренний буфер данных с помощью вызова fflush(f_ptr); термин ‘сбросить’ означа́ет ‘записать в файл часть данных, уже нако́пленных в буфере вывода, но физически еще не записанных на носи́тель средствами ОС). Вызовы fflushall и fcloseall принудительно сбрасывают все буферы в файлы и закрывают все файлы соответственно (хотя при *успешном окончании программы* гарантируется корректное закрытие всех открытых к данному моменту файлов, опытные программисты с пользой применяют fflush, fclose, fflushall, fcloseall).

После закрытия файла указатель на него считается неопределенным и в дальнейшем может быть связан с любым иным файлом. Указатели stdin, stdout, stderr, stdaux, stdprn суть также указатели на структуру типа FILE, однако они являются константами и не могут быть переопределены.

Полностью форма оператора fopen следующая:

указатель_на_файл = fopen(имя_файла, режим_открытия);

где имя_файла – текстовая строка, определяющая полное (*с указанием пути по фа́йловой системе*) имя файла, с которым предполагается работать, режим_открытия – текстовая строка, содержащая один или несколько определяющих режим работы с файлом символов (“‘r” – открытие для чтения, “w” – открыть файл для записи, “a” – открыть для дополнения, “r+” – для чтения и записи, “w+” – открыть пустой файл для чтения и записи, “a+” – для чтения и записи в конец файла). При “r+” файл в момент выполнения fopen должен существовать; если файл не существует, то при значениях “a”, “**a+**” он создается; если файл уже существует, то при “w”, “w+” содержимое файла теряется. С помощью freopen можно переадресовать поток (напр., freopen(“my_file.dat”, “w”, stdout) перенаправит выводимый в файл my_file.dat поток данных на stdout – т.е. на дисплей).

Запись в (открытый) поток (файл) по формату (*текстовый режим*) совершается оператором fprintf(указатель_на_файл, строка_формата, объект_1, объект_2, …), где формальные параметры полностью соответствуют ранее описанным (см. лабораторную работу 2 этого пособия); то же относится к fscanf(указатель_на_файл, строка_формата, &объект_1, &объект_2, …). Т.о. printf и scanf фактически являются вызовами fprintf и fscanf с указателями на файл в виде stdout и stdin соответственно.

Запись и чтение в/из открытый файл по формату в *двоичном режиме* реализуют функции fwrite и fread. Формат fwrite следующий:

fwrite(буфер, размер_элемента, число_элементов, указатель_на_файл);

где буфер – *указатель* на буфер, содержащий подготовленные для записи данные, размер_элемента и число_элемента – размер каждого элемента и их количество, находящихся в буфере; формат fread аналогичен. При удачном завершении обе функции возвращают реально обработанное *число элементов*, указатель в файле устанавливается *за записанными/прочитанными элементами*. Определены также функции чтения и записи как одиночного символа, так и строки символов.

При работе с потоками (файлами) определено понятие *текущей позиции* в потоке (файле) – байтовое смещение текущей точки в файле от его начала. Функции ftell и fgetpos получают значение текущей позиции в файле, fseek и fsetpos устанавливают текущую позицию, rewind принудительно устанавливают значение текущей позиции в 0 (т.е. на начало файла).

Формат функции fseek следующий (при успешном позиционировании указателя функция возвращает 0):

```
fseek(указатель_на_файл, смещение, откуда);
```

где указатель_на_файл – идентификатор файла, смещение – смещение текущей позиции в файле в форме длинного целого (тип long int); откуда может принимать значения констант SEEK_SET, SEEK_CUR или SEEK_END, которые определяют, относительной какой базы задана величина смещение – от начала файла, от текущей позиции в файле или от конца файла соответственно. Некоторые файловые функции также корректируют положение указателя (напр., при удачном завершении fread указатель позиции в файле устанавливается *за прочитанными элементами*).

В языке С кроме использования линейки функций fopen, fread, rwrite и fclose существуют и функции open, read, write, close, применяющие для идентификации файлов уникальные целые переменные (вместо структур типа FILE); использование их близко первым (правда, отсутствует возможность форматированного вывода). Заметим, что в общем случае *функции потока и функции нижнего уровня несовместимы* (что связано в первую очередь с управлением буферизацией), поэтому при работе с файлом следует использовать только один тип функций.

Число одновременно открытых файлов в операционной системе не безгранично (для MS DOS задается в системном файле CONFIG.SYS строкой вида FILES=60), поэтому рекомендуется использовать правило ‘*открыл файл → поработал с ним → закрыл файл*’. Однако не следует забывать, что время открытия файла на многие порядки превышает время выполнения процессорных инструкций, поэтому при выводе значительного количества относительно небольших блоков данных файл многократно закрывать и открывать нецелесообразно.

4.3 Необходимое оборудование

Необходимым оборудованием является IBM PC-совместимый ПК с предустановленной интегрированной средой программирования фирмы Borland Int.

4.4 Порядок проведения работы

Студент загружает IDE фирмы Borland Int. (в случае работы в Windows загрузка происходит в DOS-сессии), подготавливает нижеприведенный исходный текст программы на языке С, компилирует ее, выявляет ошибки, запускает на выполнение. Качество выполнения программы проверяется преподавателем.

В данной работе создается простая система управления базой данных (СУБД) с функциями просмотра содержимого БД, добавления *записи* и поиска *записи* по заданному критерию. *Запись* является минимально обрабатываемым объектом БД и обычно ассоциируется (в *реляционных* базах данных) со *строкой двумерной таблицы*.

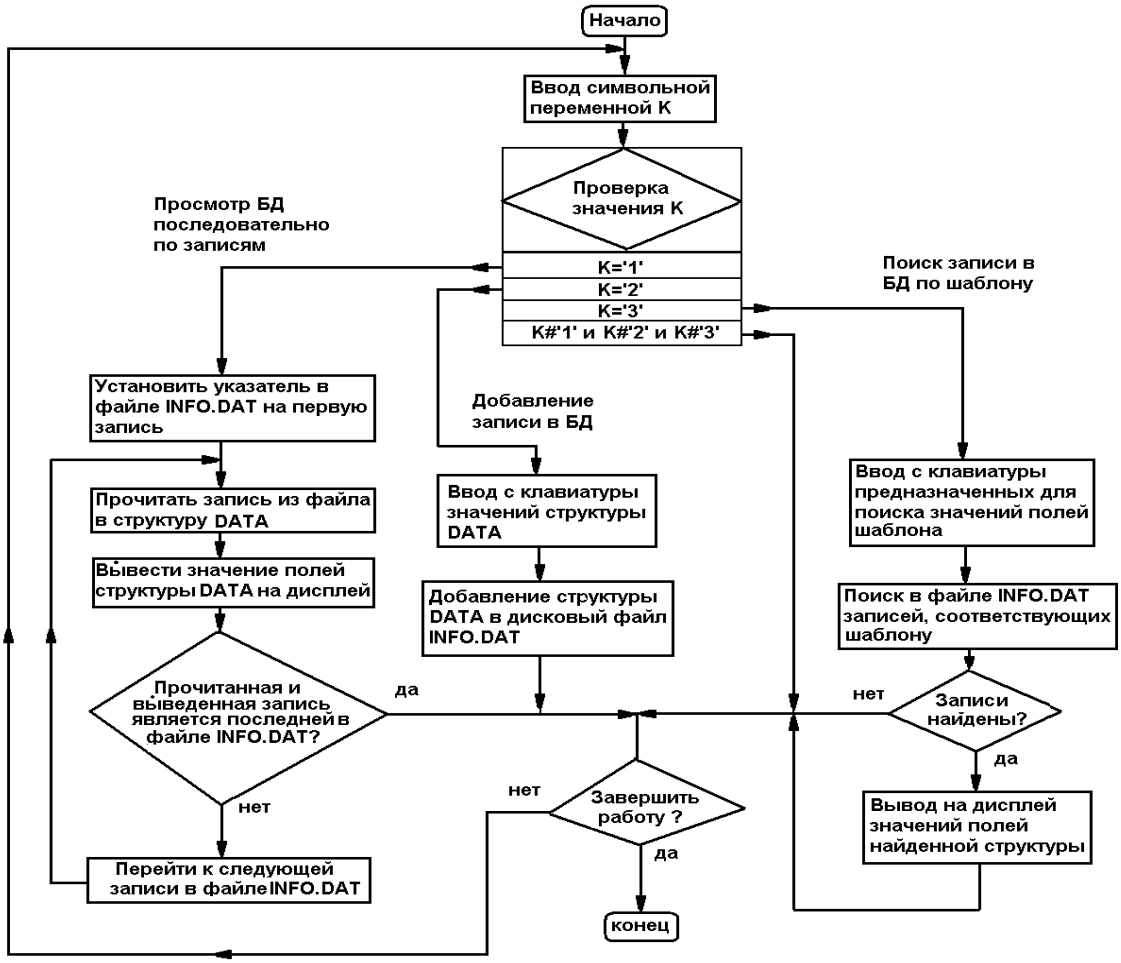

Рисунок 4.1. Схема функционирования простой СУБД

В данной БД каждая запись содержит данные о фамилии, имени, отчестве и возрасте работника в *поля́х* с именами Fam, Name, secName и Old соответственно, причем первые три поля имеют тип *си́мвольной строки*, а последнее – *целое число*. Схема СУБД приведена на рис.4.1. Поняв принцип ее функционирования, программист может расширить функциональность данной СУБД – напр., добавить функции редактирования и уничтожения выбранной записи.

Исходный текст программы на языке С, реализующей показанный на рис.4.1 алгоритм приведен ниже:

```
/* исходный текст программы 1_04.c */

#include <stdio.h>
#include <string.h>

main()
{
struct Info {
  char Fam[40], /* поле ФАМИЛИЯ */
         Name[20], /* поле ИМЯ */
         secName[50]; /* поле ОТЧЕСТВО */
  int Old; /* поле ВОЗРАСТ */
 } Data;  /* Data - структура типа Info */

char FileName[]="INFO.DAT", /* имя файла базы данных */
       Fam[40], /* строки для ввода шаблона при поиске в базе данных */
       Name[20],
       SecName[50],
       strOld[10];
FILE *fptr; /* указатель на стандартную структуру типа FILE */
int sizeData = sizeof(Data), /* размер структуры типа Info */
    sizeFile, i, nRecords,
    Old,
    found; /* переменная-флаг */

again: /* метка начала работы программы */

printf("\n\n\nВыберите режим работы:\n\n1 – вывести все записи базы данных,\n");
printf("2 – добавить новую запись,\n3 – найти запись по шаблону.\n\n");

 switch(getch()) /* выбор по введенному с клавиатуры значению */
 {
  case '1': /* вывести все записи в базе данных */
             fptr=fopen(FileName, "a+");
             if(fptr==NULL)
             {
              printf("\nОшибка открытия файла %s в режиме 'a+' ...\n", FileName);
```

```
            goto ended;
           }
           fseek(fptr,0,SEEK_END); /* текущ.позицию установить на конец файла */
           sizeFile=ftell(fptr); /* получить длину файла */
           nRecords=sizeFile/sizeData; /* число записей в базе данных */
           if(sizeFile==0) /* если размер файла нулевой… */
           {
            printf("\nФайл %s пуст ...\n", FileName);
            goto ended;
           }
           else
            printf("\nФайл %s содержит %d записей:\n", FileName, nRecords);

           for(i=0; i<nRecords; i++) /* циклы по всем записям */
           {
            fseek(fptr,sizeData*i,SEEK_SET); /* установить на начало i-той записи */
            fread(&Data, sizeData, 1, fptr); /* прочитать запись в структуру Data */
            printf("\nЗапись #%d: %s %s %s %d", i+1,
                    Data.Fam, Data.Name, Data.secName, Data.Old);
           } /* конец оператора FOR  */

           fclose(fptr);
           break;

case '2': /* добавить запись в конец файла */
           if((fptr=fopen(FileName, "a+"))==NULL)
           {
            printf("\nОшибка открытия файла %s в режиме 'a+' ...\n", FileName);
            goto ended;
           }
           printf("\nВводите Fam, Name, secondName, Old (разделитель -        \
                    пробел)\n\n");
           scanf("%s %s %s %d",
                   &Data.Fam, &Data.Name, &Data.secName, &Data.Old);
           if(fwrite(&Data, 1, sizeData, fptr)==sizeData)
            printf("\n\nЗапись успешно добавлена в базу данных...\n");

           fclose(fptr);
           break;

case '3': /* найти запись по шаблону */
           if(!(fptr=fopen(FileName, "a+")))
           {
            printf("\nОшибка открытия файла %s в режиме 'a+' ...\n", FileName);
            goto ended;
           }
           fseek(fptr,0,SEEK_END); /* установить текущ.позицию на конец файла */
           sizeFile=ftell(fptr); /* получить размер файла */
           nRecords=sizeFile/sizeData; /* число записей */
           if(sizeFile==0) /* если длина файла нулевая…*/
```

```
            {
             printf("\nФайл %s пустой...\n", FileName);
             goto ended;
            }
            else
             printf("\nФайл %s содержит %d записей:\n", FileName, nRecords);

            printf("\nВводите значение поля 'Fam' для поиска (‘*’ - любое) => ");
            scanf("%s", &Fam);
            printf("\nВводите значение поля 'Name' для поиска (‘*’ - любое) => ");
            scanf("%s", &Name);
            printf("\nВводите значение поля 'secName' для поиска (‘*’ - любое) => ");
            scanf("%s", &secName);
            printf("\nВводите значение поля 'Old' for search (‘*’ - любое) => ");
            scanf("%s", strOld);

           if(!strcmp(strOld,”*”)) /* если строка не равна “*” */
             Old=atoi(strOld); /* …конвертируем в целое */

            found=0; /* флаг найденных записей */

            for(i=0; i<nRecords; i++) /* циклы по всем записям */
            {
             fseek(fptr,sizeData*i,SEEK_SET); /* установить на начало i-той записи */
             fread(&Data, sizeData, 1, fptr); /* прочитать запись в структуру Data */

             if( (strcmp(Fam,"*")         && !strcmp(Data.Fam,    Fam))  ||
                 (strcmp(Name,"*")      && !strcmp(Data.Name, Name)) ||
                (strcmp(secName,"*") && !strcmp(Data.secName,secName)) ||
                 (strcmp(strOld,'*')       && Data.Old==Old) )
             {
              printf("\n3апись #%d найдена %s %s %s %d",
                        i+1, Data.Fam, Data.Name, Data.secName, Data.Old);
              found=1; /* установить флаг ‘Запись найдена’ */
             }

            } /* конец оператора FOR  */

            if(!found)
             printf("\n\nНи одна запись не найдена...\n");

            fclose(fptr);
            break;

   default: goto ended;

 } /* конец оператора SWITCH */

ended: /* ме́тка конца программы */
```

```
printf("\n\nЖелаете закончить работу (Y/*) ?\n\n");

 if(getch() != 'Y') /* …нет, хочу еще поработать ! */
  goto again;

}  /* конец программы */
```

4.5 Описание программы

Следует обратить внимание на эквивалентность нижеприведенных последовательностей процедур открытия файла и проверки корректности этого действия (причем предпочтение, без сомнения, следует отдать последней из трех как наиболее компактной):

- fptr=fopen(FileName, "…");
  if(fptr==NULL) {…}
- if((fptr=fopen(FileName, "…"))==NULL) {…}
- if(!(fptr=fopen(FileName, "…"))) {…}

Размер sizeFile идентифицируемого fptr файла определяется путем последовательного применения двух вы́зовов: fseek(fptr, 0, SEEK_END) и sizefile=ftell(fptr); это *профессиональный* подход.

При последовательном чтении из файла текущая позиция для считывания записи номер i (индекса́ция по i начинается с нуля) перед fread устанавливается с помощью fseek(fptr, sizeData*i, SEEK_SET), где sizeData – размер записи.

Конструкция if(fwrite(&Data, 1, sizeData, fptr)==sizeData) {…} служит для про́верки корректности записи. Заметим, что в случае вы́зова fwrite(&Data, sizeData, 1, fptr) правильно записанным условием корректности записи было бы if(fwrite(&Data, sizeData, 1, fptr)==1) {…}.

В данном (упрощённом) примере поиск записи в БД осуществляется по условию полного совпадения по тем полям, которые заданы в шаблоне поиска (шаблоном служат строки Fam, Name, secName и целое Old); при сравнении игнорируются поля, введённые как ‘*’. Собственно поиск проводится путем последовательного сравнения значений полей шаблона с соответствующими полями всех записей в БД, в результате выдается запись, соответствующая шаблону (при отсутствие найденных записей флаг found устанавливается в 0 и выдается информация о невозможности нахожде́ния записей по заданному шаблону); строка strOld дает возможность ввода ‘*’ (для реализации сравнения с цифровым полем Data.Old содержимое этой строки конвертируется в целое с помощью функции atoi). В промышленных БД вместо последовательного (чрезмерно медленного при числе записей в миллионы и больше) поиска

используются специальные *методы ускорения поиска* (напр., *двоичный поиск – дихотомия́*).

4.6 Вопросы для самопроверки

1. Дайте определение потоку. Какие существуют стандартные потоки? С какими устройствами они связаны?
2. Привести последовательность действий при работе с дисковыми файлами. С какой целью применяются функции fseek и flush?
3. Каким образом вызов fread влияет на положение текущей позиции в файле?
4. Какова длина записи (структуры типа Info, в байтах) в данной программе?
5. Предложить способ ввода в поле strOld шаблона только целых чисел или символа “*”.
6. Каким образом можно ввести в программу анализ диапазона вводимых пользователем значений возраста (поле Old, напр., от 0 до 100? Каким образом осуществить поиск записи по заданному диапазону (*от/до*) значений возраста?

5 Лабораторная работа 5. **Динамическое управление памятью. Передача параметров программе через командную строку**

5.1 Цель работы

Целью является создание работоспособной программы, использующей возможности языка С по управлению памятью и передачи аргументов посредством командной строки. Данная работа близка́ к ранее описанному примеру перемножения матриц, однако позволяет использовать матрицы произвольного размера, который задается посредством интерфейса командной строки.

5.2 Теоретическая часть

Язык С имеет развитые средства для управления требуемой программе оперативной памяти (ОП), причем выделение *пу́ла* (части, области) происходит из общей ОП (heap, общей ‘кучи’ памяти), управляемой операционной системой. Операции управления памятью включают функции *выделения, перераспределения и освобождения* (‘отда́че’ памяти ОС) пула памяти; причем корректное управление этими возможностями возложено на программиста (операции ‘уборки му́сора’ в С не определены). Использование выделенной памяти остается также на совести программиста – язык С не контролирует, например, ситуации выхода за границы выделенного участка памяти.

Участки памяти выделяются (‘аллоки́руются’) по запросам calloc и malloc (при вызове calloc выделенная память обнуляется):

```
указатель=calloc(число_элементов, размер_элемента);

указатель=malloc(размер);
```

где указатель – указатель на выделенную область памяти, размер_элемента и размер – величина области памяти (в байтах), число_элементов – число элементов длиной размер_элемента. Реальный захват памяти в N элементов типа int может быть осуществлен вызовом pMem=calloc(N, sizeof(int)); прототипы функций для работы с памятью находится в хи́дере alloc.h.

Изменение длины выделенного и адресуемого указателем указатель участка памяти осуществляется вызовом realloc:

```
realloc(указатель, новый_размер);
```

где новый_размер – новый размер выделенного участка (может быть больше или меньше предыдущего). Функции calloc, malloc и realloc возвращают указатель на начало (младший байт) выделенного участка памяти; при невоз-

можности выделения памяти возвращается нулевой указатель. Возврат выделенного участка памяти операционной системе осуществляется функцией free(указатель).

Работа с динамической памятью часто служит источником трудно определяемых ошибок (обычно связанных с некорректно используемыми указателями). Рекомендуется перед применением вышеприведенных функций проверять значение используемых указателей – напр., так:

```
char *pMem; /* указатель на область памяти */
…..
if(!pMem) /* если pMem ни на что не указывает (т.е. == NULL) */
  if(pMem=calloc(1000, 2)) /* если удачно выделены  2000 байт… */
{
…работа с pMem…
}
if(pMem) /* если указатель # NULL… */
  free(pMem); /* отдать память системе */
```

После этого указатель pMem можно использовать снова. В конце работы программы захваченная память ‘отдаётся’ ОС, но при некорректном окончании выделенные блоки памяти могут ‘зависнуть’, затрудняя этим работу иных приложений.

Использование интерфейса *командной строки* особенно удобно, если есть необходимость передать данные (обычно небольшого размера – напр., ключи, управляющие работой конкретной утилиты – как часто делается в Xnix-системах) откомпилированной программе (без повторной ее компиляции). Механизм передачи данных с использованием командной строки столь удобен, что был введён даже в диалекты Pascal’я (где он изначально не предполагался).

Формат вызова исполняемого файла program.exe с параметрами командной строки –v1000, -n, r24 таков (при работе с интегрированной средой Turbo C командную строку можно задать путем Options → Arguments из верхнего выпадающего меню):

```
C:\tc_2\work\>program.exe –v1000 –n r24
```

Полный формат функции main (с нее всегда начинается выполнение С-программы) следующий:

```
int main(int argc, char *argv[], char *envp[]);
```

где argc – число аргументов командной строки (которые должны разделяться пробелами), *argv[] – массив ссылок на отдельные строковые аргумен-

тов (которые адресуются как argv[0], argv[1], argv[2]…, argv[argc-1]); ∗envp[] – массив ссылок на переменные среды́ данной ОС (оба массива ссылок завершаются NULL). В argv[0] всегда содержится *полный путь* к файлу данной программы (для вышерассмотренного случая это c:\tc_2\work\program.exe), в последующих строках – то, что введено после имени программы (т.е. argv[1]="-v1000", argv[2]="–n", argv[3]="r24"); при этом argc=4. Другим путем считывания и изменения переменных среды является применение функций getenv и putenv. При желании представить символ пробела внутри аргумента сам аргумент заключается в кавычки. Равносильным форматом функции main является и такой: int main(int argc, char **argv, char **envp); где int – тип возвращаемого функцией main значения (фактически *код возврата*, анализируемый ОС после выполнения программы).

Длина командной строки ограничена, однако никто не запрещает, например, передать с ее помощью имя файла, где находятся данные для программы.

5.3 Необходимое оборудование

Оборудованием является IBM PC-совмести́мая ПЭВМ с предустано́вленной интегрированной среды́ фирмы Borland Int.

5.4 Порядок проведения работы

Студент загружает IDE фирмы Borland Int. (в случае работы в Windows загрузка происходит в DOS-сессии), подготавливает нижеприведенный исходный текст программы на языке С, компилирует ее, выявляет ошибки, запускает на выполнение. Качество выполнения программы проверяется преподавателем.

Заданием является создание программы перемножения неквадратных матриц [A] и [B] с помеще́нием результата в [C], причём размеры матриц заранее неизвестны и передаются скомпилированной программе через элементы командной строки; изначально элементы исходных матриц заполняются случайными числами.

```
/* исходный текст программы 1_05.c */

#include <alloc.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

float *pA, *pB, *pC; /* указатели на участки памяти расположения элементов
```

```
                              матриц [A], [B], [C] */

#define pA(i,k,k_Max) pA[i*k_Max+k] /* ма́крос для адресации матрицы [A] */
#define pB(k,j,j_Max)  pB[k*j_Max+j]  /* …то же для матрицы [B] */
#define pC(i,j,j_Max)   pC[i*j_Max+j]   /* …то же для матрицы [C] */

void main(int argc, char *argv[], char *envp[])
{
 int NRA=10, NCB=10, NCA=10, /* значения размеров матриц по умолчанию */
     i,j,k;

 printf("\n\n\nАргументы командной строки (всего %d):\n\n", argc);
 for(i=0;i<argc;i++) /* цикл по всем argc элементам */
  printf("%s\n", argv[i]);

 printf("\nНажми любую клавишу для продолжения...\n");
 getch();

 printf("\nПеременные системы:\n");
 for(i=0;envp[i];i++) /* цикл пока envp[i] != NULL */
  printf("%s\n", envp[i]);

 printf("\nЛюбая клавиша – продолжить…\n");
 getch();

 NRA=atoi(argv[1]); /* число строк матриц [A] и [C] */
 NCB=atoi(argv[2]); /* число колонок [B] и [C] */
 NCA=atoi(argv[3]); /* число колонок [A] и строк [B] */

 pA=(float *)calloc(NRA*NCA, sizeof(float)); /* аллокируем память для [A] */
 pB=(float *)calloc(NCA*NCB, sizeof(float)); /* …то же для [B] */
 pC=(float *)calloc(NRA*NCB, sizeof(float)); /* …то же для [C] */

 if(!pA || !pB || !pC) /* …если возвращаемое значение  = NULL */
 {
  printf("\n\nНехва́тка памяти для для pA или pB или pC...\n\n");
  exit(-13);
 }

 randomize(); /* инициализация датчика случайных чисел */

 for(i=0;i<NRA;i++)  /* инициализируем значения элементов матрицы [A] */
  for(k=0;k<NCA;k++)
   pA(i,k,NCA)=random(50000)/1.0e3;

 for(k=0;k<NCA;k++)  /* инициализируем значения элементов матрицы [B] */
  for(j=0;j<NCB;j++)
   pB(k,j,NCB)=random(3000)/1.0e-2;

 for(i=0; i<NRA; i++) /* начало процедуры умножения матриц */
```

```
 for(j=0; j<NCB; j++)
  {
   pC(i,j,NCB) = 0.0; /* обнуле́ние элементов матрицы [C] */
   for(k=0; k<NCA; k++)
    pC(i,j,NCB) += pA(i,k,NCA) * pB(k,j,NCB);
  }

 printf("\nРезультирующая матрица [C] (по строкам):\n\n");

 for(i=0;i<NRA;i++)
 {
  for(j=0;j<NCB;j++)
    printf("%.4e ", pC(i,j,NCB));
  printf("\n");
 }

 printf("\n\nЛюбая клавиша – окончание программы...\n\n");
 getch();

 free(pA); /* отдать память системе */
 free(pB);
 free(pC);

}  /* конец функции main() */
```

5.5 Описание программы

Формат запуска програ́ммы тако́в (NRA, NCB, NCA - числа):

```
> c_05.exe NRA NCB NCA
```

где NRA – число строк матриц [A] и [C], NCB – столбцов матриц [B] и [C], NCA – столбов матрицы [A] и строк матрицы [B]). Если в командной строке менее трех чисел, размеры матриц будут взяты по умолчанию (NRA=NCB=NCA=10).

Сопоставление элемента 2D-матрицы A[i][j] одномерному массиву pA[] производится путем замены пары индексов [i][j] на одинарный [i*j_Max+j] с помощью директивы препроцессора (ма́кроса) #define A(i, j, j_Max) pA[i*j_Max+j], где j_Max – число столбцов матрицы (это возможно, если элементы двумерной матрицы в памяти расположены *последовательно по строкам* – что соответствует правилам языка C). Использование ма́кроса позволяет уменьшить размер исходного текста и сократить число ошибок.

5.6 Вопросы для самопроверки

1. Какова стандартная последовательность работа с динамической памятью? Каким образом получить информацию о невозможности выделения памяти?
2. Привести список формальных параметров функции main. Каков смысл списка формальных параметров? Переменных среды́?
3. Каким образом выполняемое приложение может получить информацию о том, в каком месте файловой системы расположен его файл?
4. Для чего используются макросы? В каком случае их применение действительно выгодно?
5. Разработать процедуру проверки корректности умножения матриц. Например, каков должен быть корректный результат умножения матриц со всеми единичными элементами при NRA=10, NCB=12; NCA=15 ?
6. Написать ма́крос отображения двумерной матрицы на плоскую память при условии расположения элементов матрицы в памяти последовательно по столбцам.
7. Модифицировать ма́кросы программы таким образом, чтобы элементы матриц A, B и C располагались в едином линейном (также адресу́емом указателями) массиве.
8. Разработать соответствующие ма́кросы для трехмерной матрицы в языке C.

6 Лабораторная работа 6. **Использование подпрограмм в языке С. Метод итераций при решении задач**

6.1 Цель работы

Целью является создание и отладка программы на языке С, использующей возможности работы с подпрограммами и итерационного подхода к решению поставленной задачи. В работе численным итерационным методом ищется корень сложной математической функции.

6.2 Теоретическая часть

Мощность языка программирования высокого уровня заключается в разграничении действий и гибкости при вызовах частей программы (подпрограмм). Формальное определение функции было дано в работе 1 данного пособия. Имя функции в языке С – адрес кода операций, составляющих функцию. Функция может быть вызвана явно *по имени* или по *ссылке на функцию* (причем одной и той же ссылке в разные моменты могут быть присвоены адреса различных функций). Функция может вызывать сама себя (с различными значениями формальных параметров, при этом каждый раз создаются *новые копии* локальных для функции переменных), это называется *рекурсия*.

Особо важно рассмотреть метод передачи указателя на функцию другой функции, использующей первую (внешнюю) для вычислений в своем теле. При этом в списке формальных параметров (кроме обычных переменных) необходимо указать ссылку на внешнюю функцию (признаком функции в языке С является обязательное наличие круглых скобок) возможно, пока *без конкретизации имени этой функции*, и тип формальных параметров внешней функции. Общая форма заголовка такой функции будет такой:

```
тип_возвращаемого_значения имя_функции (переменная_1, тип_результата (*указатель_на функцию)(float, float), переменная_2..);
```

где тип_результата - тип результата, возвращаемого внешней функцией.

Например, объявлением (прототипом) функции может быть: void fun_1(int a, int(*fun) (float,float), double b); а вызов ее: fun_1(12, fun_2, 34.77); где fun_2 – возвращающая значение типа int внешняя функция от двух float-параметров.

Подобная технология полезна, напр., при численном решении уравнений. Напр., уравнение tan(x)=x не имеет аналитического (в виде конечной суперпозиции элементарных функций) решения и может быть разрешено относительно x только численно.

Одним из алгоритмов решения уравнений с одним неизвестным является ‘метод деления отрезка пополам’ (известны и применяются множество иных

алгоритмов – метод простых итераций, метод Ньютона, метод секу́щих, метод пара́бол, метод ‘золотого сече́ния’ и т.д.).

Алгоритм ‘деле́ния отрезка пополам’ (*дихотоми́я* *, *биссе́кция*) является *итерационным* (т.е. решение достигается путем последовательных действий – *итераций*; вычисления при этом осуществляются *по одному алгоритму, но с разными значениями переменных*), требует априорного знания диапазона поиска корня, условия вещественности и непрерывности на этом диапазоне функции и разного знака функции на краях диапазона. В таком случае при наличии одного корня алгоритм его гарантированного находит, при нескольких корнях может быть найден один из них (но не определено, какой) или не найден ни один.

Важно обратить внимание, что признаком окончания процесса вычислений в дихотоми́и является снижение диапазона поиска до заданной величины (но отнюдь не условие ма́лости значения самой функции).

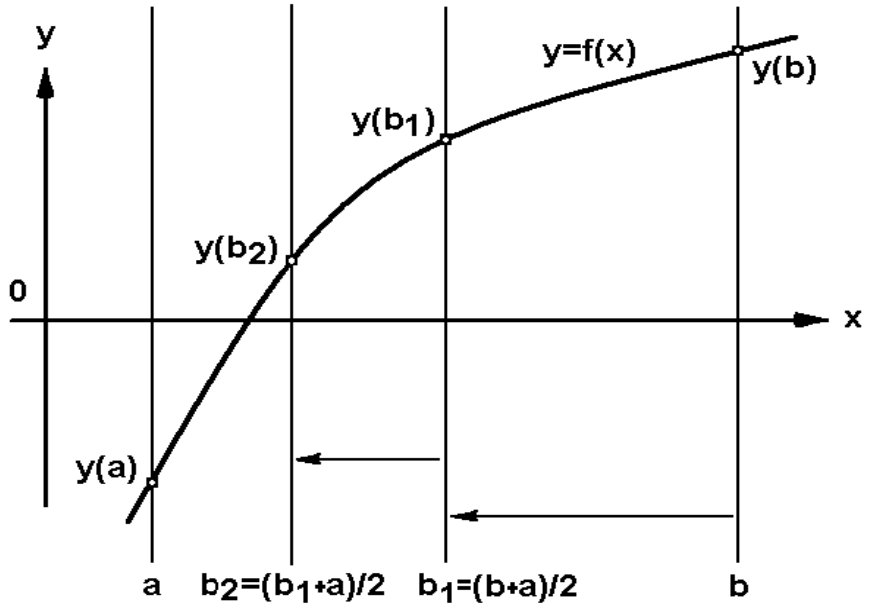

Рисунок 6.1. К описанию алгоритма ‘деления отрезка пополам’ (дихотоми́я, биссе́кция).

Последовательность выполнения дихотоми́и достаточно проста:

a) Задать границы диапазона поиска корня: x=a и x=b.

b) Вычислить y(a) и y(b).

c) Если y(a)*y(b)>0, то закончить задачу (при таких условиях решение методом дихотоми́и невозможно).

---

* Дихотомия (от греч. διχοτομία: δῐχῆ, ‘на́двое’ + τομή, ‘деле́ние’), биссе́кция (от bisection: binary, ‘два’ + section, ‘разделять’) – сино́нимы понятия ‘деление пополам’.

d) Вычислить средне арифметическое между a и b (c=(a+b)/2).

e) Вычислить y(c).

f) Если y(c)*y(b)>0, то перенести в в c (т.е. принять: b=c и y(b)=y(c)); в случае y(c)*y(a)>0 перенести a в c (т.е. принять: a=c и y(a)=y(c)).

g) Если abs(b-a)<eps (eps – заданная точность вычисления корня), принять значение корня как x0=(b+a)/2 и закончить; иначе идти к пункту d).

На рис.6.1 схематично показано последовательное приближе́ние к корню (в данном случае на протяжении двух итераций в сторону корня сдвигалась *правая граница* текущей области поиска – что показано стрелками как последовательность переноса правой границы диапазона поиска $b \rightarrow b1 \rightarrow b2$). Т.к. на каждой итерации дихотоми́и область поиска корня *су́живается ровно вдво́е*, оценкой общего числа итераций является $log_2[(b-a)/eps]$. Заметим, что для каждого уменьшения диапазона поиска корня вдво́е необходимо *всего одно вычисление функции*, корень которой ищется (плюс 3 вычисления функции перед началом итераций).

6.3 Порядок проведения работы

Студент загружает IDE фирмы Borland Int. (в случае работы в Windows загрузка происходит в DOS-сессии), подготавливает нижеприведенный исходный текст программы на языке C, компилирует ее, выявляет ошибки, запускает на выполнение. Качество выполнения программы проверяется преподавателем.

```
/* исходный текст программы 1_06.c */

#include <math.h>
#include <stdlib.h>
#include <stdio.h>

#define TOR double  /* определе́ние типа вещественных чисел (float или double) */
/* TOR суть аббревиату́ра словосочетания ‘type of real’ */

TOR Fun_1(TOR); /* прототип функции Fun_1 */
TOR Fun_2(TOR); /* прототип функции Fun_2 */
TOR RootDiRoot(TOR,TOR,TOR(*)(TOR),TOR); /* прото́тип RootDicho */

main() /* главная программа */
```

```
{
 TOR xRoot;

 xRoot=RootDicho(1.0, 10.0, Fun_1, 1.0e-9);
 printf("\n\nКорень функции Fun_1  равен: %f\n", xRoot);

 xRoot=RootDicho(0.01, 1.56, Fun_2, 1.0e-9);
 printf("\nКорень функции Fun_2  равен: %f\n\n", xRoot);

 getch();

} /* конец функции main */


TOR Fun_1(TOR x)
{
 return(sin(x)-x+1.0);
} /* конец функции Fun_1 */


TOR Fun_2(TOR x)
{
 return(tan(x)-2.0*x);
} /* конец функции Fun_2 */


TOR RootDicho(TOR a, TOR b, TOR(*fun)(TOR), TOR eps)
{
 TOR c, fa,fb,fc;

 c=0.5*(a+b);  /* c - точка посредине между a и b  */

 fa=(*fun)(a);  /* вычисление значения функции в точке a */
 fb=(*fun)(b);  /* … то же в точке b */
 fc=(*fun)(c);  /* … то же в точке c */

 do { /* начало итераций по поиску корня  функции */

/* разкомментировать для отладки
 printf("\na=% 8.4e f(a)=% 8.4e\nb=% 8.4e f(b)=% 8.4e abs(a-b)=% 8.4e\nc=% 8.4e
f(c)=% 8.4e\n",  a,fa, b,fb, fabs(a-b),  c,fc);
 sleep(2);
*/

  if((fc*fb) > 0.0) /* точку b переме́стим в  c */
  {
   b=c;
   fb=fc;
  }
```

```
 if((fc*fa) > 0.0) /* точку a переме́стим в c */
 {
  a=c;
  fa=fc;
 }

 c=0.5*(a+b); /* точка  c посредине между a и b  */

 fc=(*fun)(c); /* за́ново вычислим значение функции в точке  c */

} while(fabs(a-b) > eps); /* циклы, пока за́данная точность не достигнута */

 return(c);

} /* конец функции RootDicho */
```

6.4 Описание программы

Программа может использовать вещественные переменные типа, определенного значением TOR (сокращение от ‘type of real’ – обычно это float или double) в зависимости от требуемой точности вычислений. При желании проследить за изменением величин в ходе вычисления корня полезно ‘разкомменти́ровать’ оператор printf в функции RootDicho и убедиться, что величина abs(a-b) снижается равно вдвое на каждой итерации.

Данная программа численным методом ищет корень уравнений sin(x)=x-1 (функция Fun_1) и tan(x)=2*x (функция Fun_2) с точностью 10^{-9}. Для первой функции диапазон поиска корня [1-10], для второй - [0,01-1,56] (в точке $x=\pi/2\approx1{,}5708$ функция *тангенс* имеет разрыв 2-го рода) Итерационные циклы естественным образом реализованы с помощью *оператора цикла с постусло́вием* do… while(…).

Следует обратить внимание на форму описания прототипов функций Funct и RootDicho в начале программы – в этом случае важен *тип переменных*, но отнюдь не их имена. В случае помещения тела функции main после Funct и RootDicho прототипы этих функций не понадобятся – само́ описание функции дает компилятору инструкции по ее использованию.

6.5 Вопросы для самопроверки

1. Каким образом в списке формальных параметров функции (среди простых переменных) указывается ссылка на функцию?
2. Что такое ‘итерационный алгоритм’?
3. По какой причине в случае одинакового знака на обоих концах диапазона поиска корня функции использование алгоритма дихотоми́и невозможно?
4. Оценить количество вызовов функции при a=0.5, b=12, eps=10^{-6}.
5. Переписа́ть цикл итераций вышеприведенной программы с использованием оператора for вместо do…while.

6.6 Вопросы для самостоятельной работы

1. ‘Прогони́те’ вышеуказанную программу при директи́ве препроцессора #define TOR float (вместо указанной #define TOR double). Какой корень будет при этом найден? Почему?

   Как следует изменить значе́ние переменной eps в целях корректного выполнения программы? Почему так (изложи́ть *собственное мнение*)?
2. Из-за каких причин при вычислении корня функции tan(x)-2*x=0 *нижний преде́л* поиска задан как 0,01? Почему верхний преде́л определен константой 1,56?

7 Лабораторная работа 7. **Управление процессами в программах на языке С**

7.1 Цель работы

Целью является создание работоспособной программы на языке С, использующей возможности управления процессами (выполнить команду MS DOS, стартова́ть процесс, завершить процесс и т.д.).

7.2 Теоретическая часть

Процессом называют последовательность выполняемых процессором инструкций (программу), запускаемую на исполнение операционной системой. Однако путем использования *функций управления процессами* можно стартовать процесс посредством пользовательской программы.

Вспомним, что MS DOC является *однозадачной операционной системой* (некоторая *имитация́ многозадачности* фактически возможна лишь путем ‘обма́на’ ОС – использования т.н. TSR – *Terminate and Stay Resident* - программ). Поэтому при запуске нового процесса (*пото́мка*, spawn) исходный (*родитель*, parent) процесс может или завершиться или перейти в ‘спящее’ состояние (до момента окончания потомка, тогда родитель ‘просыпается’). Для многозадачных ОС (Windows, Xnix и др.) процесс-родитель и потомок могут выполняться одновременно, причем возможно управление выполнением процесса-потомка со стороны родителя.

Простейше́й функций для выполнения любой MS DOS-команды является system (прото́тип в process.h или stdlib.h):

```
int system(const char *command_string);
```

где command_string – строка, содержащая допусти́мую команду MS DOS, при успешном выполнении команды возвращается 0, при неудаче (–1).

Функции семейства exec… (прототипы в process.h) загружают и выполняют программу из файла в качестве процесса-потомка, при этом программа-родитель безвозвратно теряется. Несколько прото́типов функций семейства exec приведены ниже:

```
int execl(char *path_to_file, char *arg0, char *arg1, … char *argN,  NULL);

int execv(char *path_to_file, char *argv[]);

int execve(char *path_to_file, char *argv[], char *envp[]);
```

где path_to_file – строка, содержащая полный путь к файлу дочерней программы, char *argv[] и char *envp[] аналогичны описанным ранее, в работе 5 данного пособия.

Функции типа exec… с добавлением символа p осуществляют поиск файла программы-потомка в стандартных путях DOS (переменная path в autoexec.bat), символ “l“ говорит о возможности передачи потомку аргументов в виде фиксированного (терминируемого NULL) списка, символ “v“ – то же в виде массива аргументов, в случае “e“ возможна передача процессу-потомку и переменных среды. При успешном выполнении ничего не возвращается (ибо родительский процесс более не существует), при неудаче возвращается (–1) или значение переменной errno, подробно идентифицирующее ошибку.

Наиболее гибкими возможностями обладают функции семейства spawn (прототипы в process.h), обладающие свойством запускать процесс-потомок без окончательного уничтожения родительского.

```
int spawnl(int mode, char * path_to_file, char *argv0, char *argv1, … NULL);

int spawnv(int mode, char * path_to_file, char *argv[], char *env);

int spawnle(int mode, char * path_to_file, char *argv0, char *argv1, … NULL,
            char **envp);

int spawn(int mode, char * path_to_file, char *argv0, char *argv1, … NULL);
```

Ключевым здесь является значение ключа mode, определяющего поведение родительского процесса после старта процесса-потомка. При mode=P_WAIT процесс-потомок начинает выполняться, а процесс-родитель ‘засыпает’ до момента окончание потомка; при mode=P_OVERLAY процесс-потомок замещает родителя (как при вызове функций семейства exec); вариант mode=P_NOWAIT предполагает совместное (*конкурентное*) выполнение обоих процессов (чего в однозадачной ОС невозможно, посему mode=P_NOWAIT в среде MS DOS недопустимо). Символы “p“, “l“, “v“, “e“ квалифицируют функции семейства spawn так же, как и exec.

7.3 Порядок проведения работы

Студент загружает IDE фирмы Borland Int. (в случае работы в Windows загрузка происходит в DOS-сессии), подготавливает нижеприведённый исходный текст программы на языке C, компилирует ее, выявляет ошибки, запускает на выполнение. Качество выполнения программы проверяется преподавателем.

```
/* исходный текст программы 1_07.c */

#include <conio.h> /* прототип для функции clrscr */
#include <process.h> /* прототипы для system, spawn */
#include <stdio.h>

main()
{
 char file[80], arg1[20],arg2[20],arg3[20];
 int mode=0; /* режим вызова spawn */

 clrscr(); /* очистить экран */

label_1: // в ответ на вопрос нажата клавиша “Y”

 printf("\nКомандная строка для использования в SYSTEM :\n");
 scanf("%s", file);

 if(-1 == system(file))
  printf("\n\nНевозможно выполнить SYSTEM(%s) \n\n", file);

 printf("\nЖелаете снова использовать SYSTEM? (Y/*)\n");
 if(getch()=='Y')
  goto label_1;

label_2: // в ответ на вопрос нажата клавиша “P”

 printf("\nДанные для вызова SPAWNL (разделитель - пробел)\n");
 printf("mode (0 - P_WAIT, 2 - P_OVERLAY), file, arg1, arg2, arg3) :\n");
 scanf("%d %s %s %s %s", &mode, file, arg1, arg2, arg3);

 if(!(mode==0 || mode==2))
  goto label_2;

 if(-1 == spawnl(mode, file, “”, arg1, arg2, arg3, NULL))
  printf("\n\nНевозможно выполнить SPAWNL(%d,%s, ,%s,%s,%sNULL)\n\n",
              mode, file, arg1, arg2, arg3);

 printf("\nЖелаете снова использовать SPAWNL / SYSTEM ? (P/Y/*)\n");

 if(getch()=='P')
  goto label_2;
 else
 if(getch()=='Y')
  goto label_1;

} /* конец функции main */
```

7.4 Описание программы

В представленной программе используется функция clrscr (прототи́п в conio.h) для очистки текстового экрана перед началом работы, что упрощает диалог и вывод информации.

В качестве параметра функции system выступает текстовая строка file, значение которой вводится пользователем (при использовании пробела внутри этой строки во внимание будет принята только *часть строки до пробела*; т.о. в этом случае нет возможности передать вызываемой программе параметры с использованием механизма командной строки). При работе с программой можно пробовать ввод известных системных команд DIR, HELP, VER, TREE etc; также можно запускать на исполнение любую COM/EXE/BAT-программу.

Более гибким инструментом является вызов spawnl. Его мощь заключается в возможности передачи программе-потомку параметров через командную строку. В вышеприведенной программе в вызове spawnl(mode, file, “”, arg1, arg2, arg3, NULL) задействованы фактически четыре параметра командной строки – arg0, arg1, arg2 и arg3; но пользователем вводятся только последние три - arg1, arg2 и arg3. Эти четыре параметры соответствуют argv[0] - argv[3] в программе-потомке, однако argv[0] всегда содержит полный путь к файлу исполняемой программы и присвоенное ему в вызывающей программе значение пропадает (именно из-за этого в качестве argv0 при вызове spawn задается пустая строка в виде “”).

Лежащий на поверхности метод проверки возможностей вызова spawn состоит в задании параметров 0 c:\tc_2\work\1_05.exe 50 60 70. При этом c:\tc_2\work\1_05.exe – полный путь к откомпилированной ранее (см. работу 5 данного пособия) программе 1_05.c; при корректном вызове исполняемый файл 1_05.exe будет выполнен так же, как если бы он был стартова́н из командной строки MS-DOS с соответствующими параметрами: c:\tc_2\work>1_05.exe 50 60 70.

Нетрудно априори предположить результаты вы́зовов 0 c:\windows\notepad my_text.txt 0 0 и 0 winrar 1_06.exe 0 0 ; терминирующие нули необходимы для формального удовлетворения ввода согласно scanf("%d %s %s %s %s", &mode, file, arg1, arg2, arg3). Обратите внимание, что программа 1_07.exe до окончания программы-потомка становится неуправляемой (‘ви-си́т’ - что полностью соответствует описанию режима P_WAIT; при P_OVERLAY такого, естественно, не будет, ибо процесс-потомок тут же после своего старта замещает родителя).

7.5 Вопросы для самопроверки

1. Как значения P_WAIT, P_NOWAIT и P_OVERLAY параметра mode влияют на функционирование spawn? Почему для однозадачных ОС значение mode=P_NOWAIT неприемлемо?
2. Какому значению mode при вызове функций семейства spawn соответствует вызов system?
3. Какому значению mode при вызове функций семейства spawn соответствует вызов execl(char *path_to_file, char *arg0, char *arg1, … char *argN, NULL)?
4. По какой причине в данной программе третий формальный параметр функции spawnl описан как пустая строка (“”)?

8 Лабораторная работа 8. **Использование гра́фики в программах на языке С**

8.1 Цель работы

Целью работы является создание работоспособной программы на языке С, использующей графические режимы вывода информации. В качестве конкретного примера рассматривается задача графического выделения корня алгебраического уравнения.

8.2 Теоретическая часть

Обычный для MS DOS режим экрана в 25 строк по 80 символов в строке называется *те́кстовым* (80×25 знакоме́ст). *Графические режимы* характеризуются бо́льшим разрешением экрана (от 320×200 независимо выводимых точек – *пи́кселов*) и в IDE фирмы Borland Int. поддерживаются с помощью расшире́ния набора функций (прототипы в хи́дере graphics.h), которых в классическом С нет.

Любая графическая С-программа организуется следующим образом:

- Установка графического видеорежима.
- Создание и манипулирование графическими объектами.
- Восстановление первоначальной конфигурации видео (и, если необходимо, выход из программы).

Инициализация (установка) графического режима (*видеорежима*) реализуется функцией

```
void initgraph(&driver, &gmode, “путь_к_графическому_драйверу”);
```

Здесь driver – тип драйвера: напр., CGA (до 320×200 пиксел), EGA (до 640×350), VGA (до 640×480); значение driver=DETECT инициирует автодетектирование возможностей графического драйвера. Значение mode конкретизирует возможности данного драйвера (напр., VGAHI соответствует 640×480 при 16 цветах и буферу размером в 1 экран). Параметр “путь_к_графическому_драйверу” – полный путь к файлу драйвера (напр., egavga.bgi, часто указывается только путь в каталог, содержащий файлы драйверов).

Проверить возможности конкретного графдрайвера можно с помощью вызова detectgraph, протестировать результат вызова initgraph - функцией graphresult, возвращающий код ошибки инициализации графрежима (grOk – ини-

циализация успешна); текст ошибки возвращается функцией grapherrormsg как grapherrormsg(код_ошибки).

Имеется возможность скомпонова́ть код графдрайвера с кодом приложения, для этого сначала драйвер компилируется в объектный файл (расшире́ние obj), который затем указывается в файле проекта для совместной линковки.

Графическая подсистема включает множество функций как для уточнения параметров (разре́шающая способность экрана, допустимое число цветов etc), так и для отриско́вки графических примитивов. Графическими примитивами являются точка (отриско́вывается функцией putpixel), отрезки прямых – line, lineto, linerel, прямоугольник – bar, многоугольник – drawpoly, дуга – arc, окружность – circle, эллипс – ellipse и т.д.; с целью отриско́вки правильных окружностей следует использовать коэффициент отношения пиксельных размеров по осям абсцисс и ординат, определяемый фу́нкцией getaspectratio.

Заметим, что во всех случаях используется дека́ртова *правая нижняя* система координат, причем началу координат соответствует пиксел x=y=0.

8.3 Порядок проведения работы

Студент загружает IDE фирмы Borland Int. (в случае работы в Windows загрузка происходит в DOS-сессии), подготавливает нижеприведённый исходный текст программы на языке C (функционирование программы заключается в графическом отображении заданной функции), компилирует ее, выявляет ошибки, запускает на выполнение. Качество выполнения программы проверяется преподавателем.

```
/* исходный текст программы 1_08.c */

#include <graphics.h> /* прото́типы графических функций */
#include <math.h>
#include <stdio.h>
#include <dos.h>

float Func(float); /* прото́тип функции Func */

main()
{
 int gdriver=DETECT, gmode, err_code,
     colors_max,x_max,y_max, x0,y0,
     i,N=50;
 float k_x,k_y,
     x_left=-1.4,x_right=1.4, /* диапазон по оси абсцисс */
     dx, x1,y1,x2,y2;
 char str[80];
```

```
initgraph(&gdriver, &gmode, "egavga.bgi"); /* инициализация графрежима */

if((err_code=graphresult()) != grOk) /* инициализации графики некорректна ? */
 {
  printf("\nОшибка инициализации графики: %d\n", grapherrormsg(err_code));
  getch();
  exit();
 }

 colors_max=getmaxcolor(); /* максимальное число цветов */
 x_max=getmaxx(); /* число пи́кселов по оси абсцисс и ординат */
 y_max=getmaxy();

 setfillstyle(0,BLACK); /* зали́вка – сплошной черный цвет */
 floodfill(0,0,0); /* начали зали́вку с точки x=y=0 */

 setcolor(WHITE); /* установить белый цвет как текущий */
 rectangle(0,0,x_max,y_max); /* бордюр по всему экрану */

 setcolor(LIGHTGREEN); /* установить ярко-зеленый цвет как текущий */
 sprintf(str,"Screen: colors_max=%d x_max=%d pix y_max=%d pix",
                      colors_max,x_max,y_max);
 outtextxy(10,10, str); /* вывод параметров графики */

 sprintf(str,"Graph: x=[%.3e : %.3e] y=[%.3e : %.3e]",
                     x_left,x_right,Func(x_left),Func(x_right));
 outtextxy(10,25,str);

 x0=x_max/2; /* коорд. центра экрана – точки x=y=0 для отрисовки графика */
 y0=y_max/2;

 setcolor(LIGHTGRAY); /* установить ярко-серый цвет как текущий */
 line(20,y0, x_max-20,y0); /* ось абсцисс */
 line(x0,40, x0,y_max-20); /* ось ординат */

 y1=Func(x_left); /* значение функции в левой и правой точках */
 y2=Func(x_right);

 k_x=(0.9*x_max)/(x_right-x_left); /* коэффициенты для осей абсцисс и ординат */
 k_y=(0.9*y_max)/8.0;

 dx=(x_right-x_left)/N; /* шаг вычисления функции по оси абсцисс */

 setcolor(DARKGRAY); /* установить темно-серый цвет как текущий */
 rectangle(x0+k_x*x_left,y0-k_y*y1, x0+k_x*x_right,y0-k_y*y2);

 setcolor(YELLOW); /* установить желтый цвет как текущий */
 setlinestyle(DOTTED_LINE, 0, THICK_WIDTH); /* стиль линии – жирный пунктир */

 for(i=0; i<N; i++) /* цикл по шагам вдоль оси абсцисс */
```

```
  {
   x1=x_left+dx*i; /* левая граница отрезка */
   y1=Func(x1);

   x2=x1+dx; /* правая граница отрезка */
   y2=Func(x2);

   setlinestyle(DOTTED_LINE, 0, THICK_WIDTH);
   line(x0+k_x*x1,y0-k_y*y1, x0+k_x*x2,y0-k_y*y2); /* рисовать отрезок линии */

   if(y1*y2 <= 0.0) /* это точка пересечения с осью абсцисс */
    circle(x0+k_x*0.5*(x1+x2), y0, 4); /* рисовать окружность */

   delay(100); /* ждать 0,1 сек */
  }

 setcolor(LIGHTRED);
 outtextxy(x_max-250, y_max-15, "Любая клавиша - продолжить...");

 getch();

} /* конец функции main */


float Func(float x) /* функция для отрисовки */
{
 return (tan(x)-2*x);
} /* конец функции Func */
```

8.4 Описание программы

Математическое выражение отрисо́вываемого графика определено в теле функции Func, левая и правая граница по оси абсцисс заданы переменными x_left и x_right, число участков при линейной аппроксимации – N. Максимальное число цветов для выбранного графрежима - colors_max, число пикселов по горизонтали и вертикали экрана - x_max и y_max.

Текущий цвет (которым будут отрисо́вываться все графпримитивы) задается setcolor, вызов outtextxy(x,y,str) выводит текст из строки str в точку графэкрана с координатами x,y (строка Screen определяет параметры дисплейного экрана, Graph - прямоугольника вывода графика). Оси координат отрисо́вываются цветом LIGHTGRAY, сам график – YELLOW (эти константы определены в хи́дере graphics.h); для графика выбран стиль линии – DOTTED_LINE (пунктир) и толщина THICK_WIDTH (три пи́ксела).

Важным является вычисление коэффициентов перевода значений чисел в координаты экрана k_x и k_y (в единицах ‘пи́кселы/число’), на них впоследствии будут умножены значения как параметра, так и значения самой отрисо́вываемой функции (коэффициент 0,9 указывает на использование 90%

диапазона по ширине/высоте экрана). Следует обратить внимание на преобразования ‘числа/пи́кселы’ x0+k_x*x1, x0+k_x*x2, но y0-k_y*y1, y0-k_y*y2 (особенно на знаки после x0 и y0).

Собственно график отрисо́вывается с помощью функции line в виде кусочно-линейной последовательности от точки (x1,y1) до (x2,y2) на каждом шаге по оси абсцисс; с целью оживления процесса отрисо́вки реализована задержка на 0,1 сек после каждой прямой - delay(100). Заме́тим, что вызов lineto(x,y) отрисо́вывает прямую от текущего пи́кселя до пи́кселя (x,y); этим удобно пользоваться при последовательной отрисо́вке множества линий.

Каждое пересечение функции с осью абсцисс определяется по условию разности знаков отрисо́вываемой функции в начале и конце каждого прямолинейного отрезка аналогично применяемому в процедуре дихотомии (использован условный оператор if(y1*y2 <= 0.0) - см. описание работы 6 в данном пособии).

8.5 Вопросы для самопроверки

1. Перечислить несколько графических режимов (с указанием основных параметров).
2. В какой системе координат адресуются пи́кселы в графическом режиме? Что это означает?
3. Если априори известно, что данный графрежим позволяет хранить в памяти более одного экрана и реализованы функции записи изображений на каждый экран и визуализации конкретного экрана на дисплее, то каким образом можно реализовать анимацию (предложить алгоритм)?
4. Коэффициент масштабирования по оси ординат определяется в программе как k_y=(0.9*y_max)/8.0. Как переопределить его с целью автоматического учета значений отрисовываемой функции в начале и конце диапазона параметра?
5. Переписать тело цикла for(i=0; i<N; i++) таким образом, чтобы вызов функции Func внутри цикла происходил еди́ножды (исходя из принципа экономии вычислений).

Список использованной литературы

1. Керниган Б., Ритчи Д. Язык программирования С. –М.: Вильямс, 2009. -304 с.
2. Кочан С. Программирование на языке С. –М.: Вильямс, 2008. -496 с.
3. Павловская Т.А. С/С++. Программирование на языке высокого уровня. -СПб.: Питер, 2009. -464 с.

Учебное издание.

Баканов Валерий Михайлович,
Орлов Валентин Александрович

ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ С

УЧЕБНО-МЕТОДИЧЕСКОЕ ПОСОБИЕ
по выполнению лабораторных работ

Подписано в печать ___._____.2010 г.
Формат 60 × 84 1/16.
Объем 3,7 п.л. Тираж 100 экз. Заказ _____

Отпечатано в типографии Московского государственного
университета приборостроения и информатики.
107996, Москва, ул. Стромынка, 20.