МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное
образовательное учреждение высшего образования
«Юго-Западный государственный университет»
(ЮЗГУ)

Кафедра космического приборостроения и средств связи

УТВЕРЖДАЮ
Проректор по учебной работе
_____О.Г.Локтионова
«___» _____2017 г.

АНАЛИЗ ДЕМОНСТРАЦИОННОЙ ПРОГРАММЫ НА ЯЗЫКЕ СИ

Методические указания по проведению практических занятий
для студентов направления подготовки 11.03.02, 11.03.03

Курск 2017

УДК 681.3.06

Составитель В.Н. Усенков

Рецензент
Кандидат технических наук, доцент *Т.М. Белова*

Анализ демонстрационной программы на языке Си : методические указания по проведению практических занятий / Юго-Зап. гос. ун-т; сост.: В.Н.Усенков. - Курск, 2017. - 46 с.: ил. 2, прилож. 2. - Библиогр.: с. 37.

Приведены методические указания к выполнению практических работ по курсу "Системное программирование". Описывается порядок проведения практических занятий. В приложениях приводятся необходимые данные, необходимые для эффективного выполнения работ.

Методические указания соответствуют требованиям программы, утвержденной учебно-методическим объединением по специальностям автоматики и электроники (УМО АЭ).

Предназначены для студентов специальностей 11.03.02, 11.03.03 дневной и заочной форм обучения.

Текст печатается в авторской редакции

Подписано в печать . Формат 60× 84 1/16.
Усл. печ. л. 2,67. Уч.-изд. л. 2,42. Тираж 100 экз. Заказ. Бесплатно.

Юго-Западный государственный университет.
305040, г. Курск, ул. 50 лет Октября, 94.

Содержание

Введение

В практике программирования неизбежно встречается задача сопровождения программного обеспечения. Это подразумевает необходимость исследования ранее написанных программных продуктов с целью их адаптации к изменившимся условиям эксплуатации, модернизации или использования в качестве шаблона для полной переработки.

В связи с этим необходимо иметь представление о способах изучения программ, представленных исходным текстом на языке Си.

Практические работы, представленные в методических указаниях, предназначены для:

- формирования у обучающихся начальных навыков анализа программ на языке Си;
- ознакомления обучающихся с примерами написания на языке Си программ невысокой степени сложности;
- приучения обучающихся к критическому восприятию программных разработок на языке Си.

1. Демонстрационная программа на языке С. Ознакомление с программой. Изучение назначения программы

При поставке программного обеспечения к программному продукту прилагается инструкция по ее использованию, написанная в формате, согласованном с заказчиком. Нередки случаи, когда инструкция по тем или иным причинам оказывается утеряна. В этом случае пользователю приходится исследовать программу в действии и пытаться составить инструкцию самостоятельно. Что может помочь в этом случае?

Практически каждая программа в том или ином виде обычно содержит подсказку, позволяющую пользователю действовать, не обращаясь к инструкции.

В любом случае необходимо исследование программы в функционирующем виде, которое осуществляется в интерактивном диалоге посредством изучения человеко-машинного интерфейса. В зависимости от вида этого интерфейса и качества его исполнения часто удается воспользоваться большим числом предоставляемых программой возможностей.

В тех случаях, когда доступен исходный текст программы, возможно полноценное восстановление инструкции и дальнейшее сопровождение программы.

Примечания

1. Заметим, что все сказанное касается законного (со)владельца программного продукта.
2. В ряде случаев имеется возможность получить консультацию у разработчика программы.

ЗАДАНИЕ

1. Исследуйте демонстрационную программу, пытаясь определить ее назначение по тем записям, которые содержатся в ее тексте.

2. Найдите и выпишите фрагмент реализации подсказки. Проанализируйте каждый пункт и своими словами дайте интерпретацию ожидаемым действиям.

3. Изучите Приложение А и сопоставьте выводы, сделанные самостоятельно, с содержащейся там информацией.

Контрольные вопросы

1. Какова цель написания изучаемой программы?
2. Велоэргометрия относится к тематике медицины. Для чего программисту необходимо знакомство с вопросами, не связанными с программированием?
3. Изобразите структурную схему движения информации в процессе обследования
4. Какие виды информации обрабатывает рассматриваемая программа (входные сигналы)?
5. Какие возможности предоставляет FOSSIL драйвер?
6. Как реализована обработка текстовой информации данных о пациенте?
7. Как определить ограничения на количество накапливаемых данных?
8. Какие виды обработки накопленной информации выполняются программой?
9. Сохраняются ли накопленные данные после завершения работы программы?
10. Возможна ли повторная обработка накопленных данных?
11. Возможно ли редактирование входных данных после накопления?
12. В какой мере удалось выявить возможности и назначение программы при исследовании ее текста?
13. Сформулируйте несколько вопросов, которые вы могли бы задать разработчику программы.

2 Демонстрационная программа на языке С. Стиль оформления. Документирование

Стиль оформления программ и документирование играет важную роль при создании программы и неоценимы при ее сопровождении. На примере демонстрационной программы предлагается провести анализ указанных преподавателем фрагментов программы на соответствие требованиям, изложенным в материале, предоставленном ниже по тексту.

Исследованию подлежат:

- необходимость оформления и документирования;
- расстановка фигурных скобок;
- использование пробелов;
- именование функций;
- блочные комментарии;
- построчные комментарии;
- недостаток/избыток комментариев;
- описание выполняемых кодом действий.

Отметим, что, хотя важность перечисленных понятий не оспаривается никем, существует множество точек зрения на способ реализации их на практике. Любая точка зрения на этот вопрос является субъективной, поэтому для однозначной интерпретации правильности или неправильности индивидуального стиля нет оснований.

Тем не менее, начинающий программист, как показывает опыт, часто тратит слишком много времени на формирование собственного стиля при игнорировании общеизвестных рекомендаций. Вероятно, удобнее начать с любого из известных вариантов и по мере накопления опыта вырабатывать свой стиль (который, кстати, тоже видоизменяется со временем).

Необходимость оформления и документирования

Мало кто из начинающих программистов испытывает желание дополнять свои первые программы комментариями и тратить время на «украшательства» текста программы. Дело в том,

что эти программы обычно достаточно просты, чтобы помнить роль каждой их строки, а готовность ответить на любой из вопросов является истинной.

Можно долго дискутировать по этому вопросу, но вопрос, как правило, снимается сразу, если программа пишется по заказу, поскольку требования заказчика не обсуждаются. Заказчика всегда будет интересовать программный продукт, обладающий следующими свойствами:

- текст программы легко воспринимается не только автором, но и сторонними программистами, которые являются потенциальными продолжателями начатого программного проекта;
- по истечении некоторого времени исходный текст, давно забытый автором, или передаваемый другому программисту, должен быть проанализирован и понят в кратчайшие сроки.

Расстановка фигурных скобок

Необходимый, но крайне зависимый от индивидуальности программиста критерий.

Тем не менее, обычно рекомендуют изучить особенности четырех общеизвестных вариантов, названия которых могут при переводе передаваться по разному.

Стиль первый, предложенный Керниганом и Ричи в их широко известных книгах, посвященных популяризации языка C.

```
for(j=0; j<MAX_LEN; j++) {
        foo();
}
```

Стиль второй, который предложил Eric Allman при написании программных фрагментов системы BSD.

```
for (j=0; j<MAX_LEN; j++)
{
     foo();
}
```

Стиль третий, известный из документации некогда популярного компилятора Whitesmith C.

```
for (j=0; j<MAX_LEN; j++)
    {
    foo();
    }
```

Стиль четвертый, стиль программистов проекта GNU.

```
for (j=0; j<MAX_LEN; j++)
     {
          foo();
     }
```

Как видно, стили отличаются друг от друга:
- положением открывающей фигурной скобки;
- расположением скобок относительно порождающей их строки программы;
- величиной отступа строк, входящих в обрамленный фигурными скобками блок, относительно фигурных скобок.

Безусловно, существуют модификации стилей, удобные для индивидуального программиста.

Вероятно, правильным следует считать любой стиль, предполагая, что он выдерживается неизменным, по крайней мере, во всем изучаемом программном проекте, независимо от числа задействованных программистов.

***Замечание**: компилятор языка Си не различает стили, это чисто человеческий фактор.*

Рассмотрим пример программного текста, который нарушает однородность стиля. Попытайтесь распознать уровни вложения составных операторов сходу:

```
for(j = 0; j < MAX_LEN; j++) {
      for(k = 0; k < MAX_WIDTH; k++)
           {
            for(m = 0; m < MAX_HEIGHT; m++)
           {
                 for(n = 0; n < MAX_TIME; n++)
                      {
                       foo(j, k, m, n);
                      }
           }
           }
}
```

Использование пробелов

Ниже рассмотрен крайне неудобный для восприятия, хотя и работоспособный фрагмент программы.

```
#include    <stdlib.h>
#include      <stdio.h>
#define    P    printf
#define    I       atoi
int main(int a,char*v
[]){int r=5, i;if(a>1
) r=I(v[1]); if(r<=0)
r=5;if(r%2==0)++r;for
(i=0; i<r*r; P(i/r==(
3*r)/2-(i%r+1)||i/r==
r/2 - i%r||i/r==r/2+i
%r||i/r==i%r-r/2?"*":
" "),i++, i % r==0?P(
"\n") : 0);return 0;}
```

По существу же вопрос идет о том, как следует располагать символы пробелов, которые игнорируются компилятором, для улучшения читабельности кода:

```
так       a=b+c*d/e%f;
или так   a = b + c * d / e % f;
```

Аналогичный пример для пробелов внутри круглых скобок уже не столь однозначен:

```
так       a=fgetc(f1);
или так   a = fgetc( f1 );
```

Именование функций и переменных

Вопросы, связанные с этой темой, особенно неоднозначны для программистов, не являющихся носителями английского языка (или пользователями алфавита не латинского типа).

Из общих соображений понятно и дополнительно подтверждено опытом, что наименования любых объектов должны отражать суть этих объектов. Например, если функция обеспечивает копирование одной символьной строки в другую, ее не следует называть абстрактным именем AlfaCentavra или более прозаичным MyFirstGoodFunction.

Более естественно выглядело бы название StringCopy (для англоязычного программиста).

Играет роль также и сочетание регистров символов в имени, сравним:

```
StringCopy();
stringcopy();
strinGcopY();
sTrInGcOpy().
```

Впрочем, классическое название этой функции выглядит так:

```
strcpy();
```

что отражает склонность человека к экономии собственных ресурсов, в частности времени.

Поскольку идентификаторы языка допускают лишь латинские символы, возникают дополнительные сложности для русскоговорящих программистов. С другой стороны, имеются и положительные особенности: привязка названий функций (операторов, …) к чужому языку позволяет психологически легче структурировать программы, разделяя действия, наименования и комментарии:

```
/*запомним имя файла перед переименованием*/
```

```
StringCopy(StaroeImja, ImjaFaila);
```

Как было сказано ранее, целесообразно придерживаться единого подхода во всем изучаемом программном проекте, независимо от числа задействованных программистов.

Блочные комментарии

Блочный комментарий описывает действия или особенности нижеследующего программного текста, как правило существенного по объему и однородного по некоторым признакам (например, выполняющего некую одну сложную задачу). Введение такого вида комментариев в некотором смысле исключает необходимость использования граф-схем алгоритмов (не для всех случаев!).

Программы, снабженные блочными комментариями, удобны для сопровождения. Способы оформления таких комментариев, с одной стороны, ограничены синтаксистом языка, а с другой стороны – изобретательностью программиста.

Базовое средство комментирования в Си – это конструкция, имеющая начальный маркер (/*) и конечный маркер (*/) вида:

```
/* комментарий */
```

Приведем примеры блочных комментариев.

```
/*****************************************
 *                                       *
 *   Это блочный комментарий.            *
 *   Настоящая головная боль             *
 *   при сопровождении.                  *
 *                                       *
 *****************************************/


/*************************************
 *
 *       Это блочный комментарий с
 *       удаленной с правой стороны
 *       колонкой звездочек.
 *
 *************************************/
```

```
/*
    Это блочный комментарий с
    удаленными справа, слева, сверху и
    снизу рядами звездочек.
                                */
```

```
/*=====================================*/
/* Это описание блока верхнего уровня */
/*=====================================*/

/*-------------------------------------*/
/* Это описание блока нижнего уровня */
/*-------------------------------------*/
```

Построчные комментарии

Построчные комментарии обычно используют для уточнения действия или особенностей одной-двух строк программного текста. Как недостаток, так и избыток такого вида комментариев отрицательно сказываются на времени изучения программы (впрочем избыток их встречается крайне редко).

В ранее приведенном примере построчный комментарий предшествовал программному коду:

```
/*запомним имя файла перед переименованием*/
 StringCopy(StaroeImja, ImjaFaila);
```

но часто встречается и такой вариант:

```
StringCopy(StaroeImja, ImjaFaila); /*запомним
имя файла перед переименованием*/
```

Отметим, что в данном случае комментарий можно считать избыточным, в отличие от случая:

```
StringCopy(StaroeImja, ImjaFaila); /* УДАЛИТЬ
ЭТИ СТРО- */
```

```
RenameFile(ImjaFaila,”tttttt”)      /*КИ ПОСЛЕ
ОТЛАДКИ!   */
```

Впрочем, можно привести и более характерный пример избыточности:

```
i++;   /*   добавить 1 к i   */
```

Многие компиляторы предоставляют более удобную возможность оформления построчных комментариев, которые начинаются с маркера // и действуют до конца строки текста:

```
StringCopy(StaroeImja, ImjaFaila); //запомним имя файла перед
                                   // переименованием
```

Описание выполняемых кодом действий.

Комментарии должны отражать то, для чего код предназначен (т.е. намерения программиста), а не механизм выполнения. Рассмотрим пример:

```
/* сохранить текущий заголовок списка */
temp = *list;
/* перенести указатель списка на следующую
   позицию в списке */
*list = (*list) -> next;
/* копировать данные из текущей позиции списка */
memcpy (dataptr, temp -> data, sizeof *dataptr);
```

Эти комментарии документируют механизм, но не намерения программиста. Значительно лучше использовать блочный комментарий:

```
/* обработать этого клиента и сделать текущим
   следующего клиента */
temp = *list;
*list = (*list) -> next;
memcpy (dataptr, temp -> data, sizeof *dataptr);
```

Такой комментарий предпочтительней. Кроме того, он не зависит от механизма реализации замысла: когда мы решим удалить список и использовать вместо него простой массив, нам даже не придется переписывать комментарий.

В изложенном материале частично использованы рекомендации из замечательной книги [2].

ЗАДАНИЕ

1. Исследуйте текст демонстрационной программы на предмет стиля расстановки фигурных скобок.

2. Попытайтесь отнести этот стиль к одному из ранее описанных стилей. Сделайте выводы о степени соответствия. Изложите свое мнение по этому вопросу.

3. Исследуйте текст демонстрационной программы на предмет наличия комментариев каждого вида.

4. Оцените в процентах суммарный текст комментариев по отношению ко всему тексту программы. Сделайте выводы о достаточности/недостаточности комментариев.

5. Изложите свое мнение по поводу количества и качества изученных комментариев.

6. Исследуйте текст демонстрационной программы на предмет использования пробелов для удобочитаемости программного текста. Изложите свое мнение по этому вопросу.

Контрольные вопросы

1. Какие виды комментариев используются в заданном фрагменте демонстрационной программы?
2. Соответствует ли стиль комментирования требованиям, изложенным в книге?
3. Предложите возможные улучшения текста программы в этом направлении.
4. Какой из стилей расстановки фигурных скобок используются в заданном фрагменте демонстрационной программы?
5. Соответствует ли стиль расстановки скобок требованиям, изложенным в книге?
6. Предложите возможные улучшения текста программы в этом направлении.
7. Соответствует ли стиль именования переменных и функций требованиям, изложенным в книге?
8. Предложите возможные улучшения текста программы в этом направлении.
9. Изучите материал из рекомендованной книги (). Согласны ли вы с утверждениями, изложенными в книге? Представьте свое мнение по этим вопросам.

3. Демонстрационная программа на языке С. Структура программы. Глобальные и локальные объекты

Основной структурной единицей программного текста является функция. Обязательным является наличие функции с именем **main**. Именно с этой функции начнется выполнение программы после ее запуска. Часто функция main находится позиционно на последнем месте программного текста, что не изменяет порядка запуска программы. Такое расположение следует из желания расположить все функции в порядке, удобном для обработки компилятором: вначале функции, которые будут вызываться последующими функциями, а затем – вызывающие функции.

Каждая функция имеет тип, имя, передаваемые в функцию параметры и тело функции в фигурных скобках:

```
тип имя(параметры)
тип параметров
{
 Тело функции
}
```

Например, функция, выполняющее сложение двух целочисленных операндов, может выглядеть следующим образом:

```
int CalcSum(oper1, oper2)
int oper1, oper2;
{
 int sum;
 sum=oper1+oper2;
 return sum;
}
```

Приведенная запись представляет так называемый «старый» стиль оформления функции. В «новом» стиле описания типов операндов осуществляется не отдельной строкой, а при объявлении операндов:

```
int CalcSum(int oper1, int oper2)
{
 int sum;
```

```
 sum=oper1+oper2;
 return sum;
}
```

Тип функции определяет тип результата, который возвращается функцией посредством оператора return и может быть присвоен переменной соответствующего типа:

```
int result, x;
result=CalcSum(x, 8);
```

Переменные, объявленные в теле функции актуальны только на время выполнения функции; такие переменные называют *локальными*. Доступ к локальным переменным извне закрыт, поэтому возможно дублирование имен таких переменных, расположенных в различных функциях.

Для того, чтобы иметь возможность доступа к общим для нескольких функций переменным, их следует определить на уровне функций. Такие переменные носят название *глобальные*. Обычно глобальные переменные располагают вверху программного текста, перед первой функцией. Делается это для удобства поиска таких переменных в тексте.

В нижеследующем примере объявлены три переменные с именем ***sum***. Первая из них по тексту является глобальной, две последующие – локальными.

```
#include "math.h"

int sum; /*глобальная переменная*/

int CalcSum(int oper1, int oper2)
{
 int sum; /*первая локальная переменная*/
 sum=oper1+oper2;
 return sum;
}

float Aver(float *mass1, int N)
{
 int i;
 float sum; /*вторая локальная переменная*/
 for(i=0;i<N;i++)
```

```
 sum+=*mass1++;
 return sum/(float)N;
}
```

В начале программного текста обычно располагают директивы препроцессора, например ***include*** для указания библиотек, в которых сосредоточены используемые в программе функции.

ЗАДАНИЕ

1. Изучить демонстрационную программу.
2. Выделить и описать все структурные единицы программного текста.
3. Идентифицировать и выбрать в качестве примера одну из функций, работающих только с локальными переменным.
4. Идентифицировать и выбрать в качестве примера функцию, осуществляющую доступ к глобальным переменным.
5. Идентифицировать и выбрать в качестве примера функции, одна из которых вызывается другой функцией.

Контрольные вопросы

1. Что такое структура программного текста?
2. Из каких компонентов состоит программный текст?
3. Каким образом (по каким признакам) можно идентифицировать в тексте программы отдельную функцию?
4. Что такое локальные переменные?
5. Что такое глобальные переменные?
6. Могут ли существовать идентификаторы объектов с идентичными названиями?
7. Приведите примеры локальных переменных в демонстрационной программе.
8. Приведите примеры глобальных переменных в демонстрационной программе.
9. Какие библиотеки используются в демонстрационной программе?

10. Какие директивы препроцессора используются в демонстрационной программе? Опишите их назначение.
11. Отыщите структурные и прочие несоответствия в нижеследующем программном тексте (или в тексте, выданном преподавателем для этого упражнения).

```
#include "math.h"

/*==================================*/
/* Это программа для вычисления
/* среднего значения элементов массива
/*==================================*/
#define N 20

float mass[20];
float sum, srednee;

/*заполним массив значениями для проверки*/
FillMass(N);
/*вычислим сумму элементов массива*/
sum=CalcSum(mass,N);
/*Выведем на экран значение среднего*/
srednee=sum/N;
printf("Среднее=%d",sum);
/*завершение программы              */
goto VSE;

int CalcSum(int oper1, int oper2)
{
 int sum; /*первая локальная переменная*/
 sum=oper1+oper2;
 return sum;
}

void FillMass(float *mass1, int N)
{
 int i;
 float sum;

 for(i=0;i<N;i++)
 mass1[i]=i++;
 return 0;
}
VSE:
```

4. Демонстрационная программа на языке С. Иерархия пользовательских функций

Анализ простых программ, соответствующих небольшим текстам программы, обычно осуществляется без особых затруднений. Но по мере роста количества функций и при усложнении алгоритмов функционирования разобраться в программе становится намного сложнее.

При анализе сложных программ часто полезно исследовать иерархию пользовательских функций – определить взаимозависимость функций друг от друга и упорядочить их по уровням вложенности вызовов. Очевидно, на верхнем уровне иерархии всегда будет находиться функция main(), поскольку выполнение любой программы, написанной на языке Си, начинается именно с этой функции. Функции, которые вызываются из main(), образуют следующий уровень иерархии. Далее возможно повторить поиск вызовов и построить дальнейшие зависимости. Результат работы удобно представить в наглядной форме, представленной на нижеследующем рисунке.

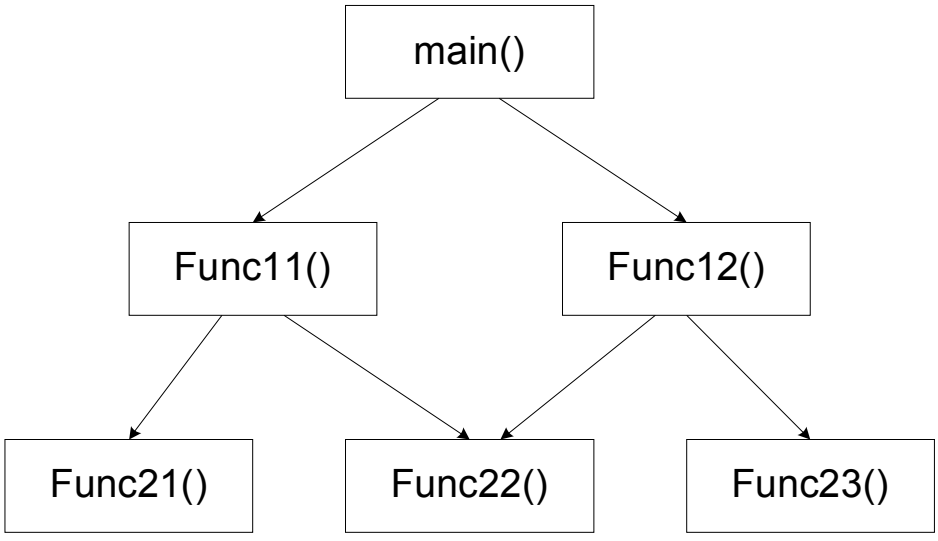


Рисунок 4.1. Пример изображения иерархических зависимостей пользовательских функций.

На рисунке показана функция верхнего уровня main(), функции второго уровня Func11() и Func12() и функции третьего

уровня Func21(), Func12(), Func12(). Стрелками показаны направления вызова – от вызывающей функции к вызываемой.

Возможны варианты, когда одна из функций нижних уровней вызывается несколькими функциями более высоких уровне, что на рисунке присуще функции Func22().

Иногда удобно предварительно составить таблицу взаимосвязей функции по примеру нижеследующей таблицы.

Таблица 4.1 – Пример табличного представления взаимосвязей функций.

Функция	Список вызываемых функций	Список вызывающих функций
main()	Func11(), Func12()	-
Func11()	Func21(), Func22()	main()
Func12()	Func22(), Func23()	main()
Func21()	-	Func11()
Func22()	-	Func11(), Func12()
Func23()	-	Func12()

Сложные программы могут иметь соответственно сложные иерархические зависимости, но тем важнее при их анализе иметь под руками аналог рисунка 4.1.

В представленных примерах приведены простейшие явные случаи взаимодействия функций. Язык Си допускает более сложные варианты вызова, используя механизм указателей. При этом вызываемая функция может выбираться из некоторого списка во время выполнения программы, что усложняет построение иерархических зависимостей но часто упрощает программу.

При использовании прерываний обработчик прерываний может вызывать другие функции также по критериям, вычисляемым во время выполнения программы.

Особый случай наблюдается и при реализации механизма *сопрограмм* [6].

Приведенный ранее рисунок иногда полезно дополнить переменными, указав их взаимосвязь с функциями, как показано на рисунке ниже.

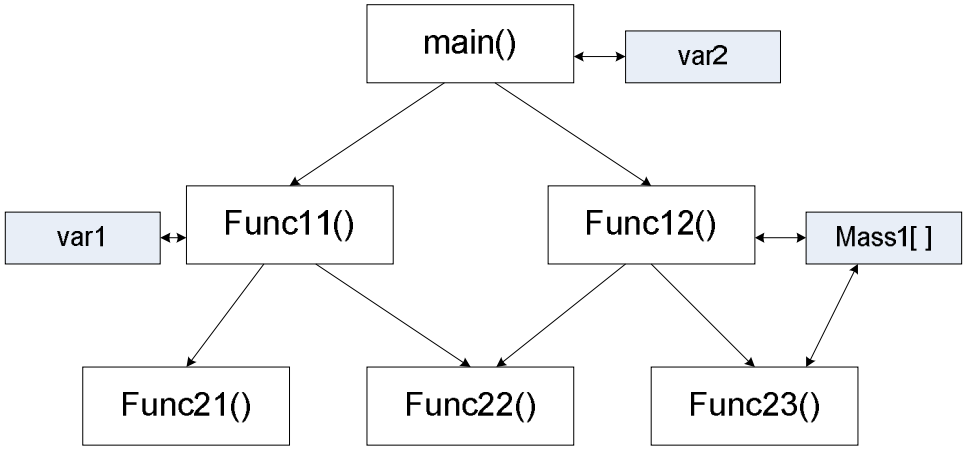


Рисунок 4.2. Иерархические зависимости между функциями с учетом переменных.

Отметим, что формальное выполнение этой рекомендации скорее всего приведет к чрезмерно усложненному и запутанному рисунку; выбор учитываемых переменных и их количество следует тщательно продумать исходя из важности этих переменных для исследуемого аспекта программы.

Во многих случаях при составлении иерархии целесообразно объединять несколько функций в программный блок, что упростит рисунок и улучшит понимание программы. Будем полагать, что программный блок содержит только внутренние взаимосвязи и выполняет некую единую функцию. В этом случае мы получим не иерархию пользовательских функций, а иерархию программных модулей.

Более подробно рассматриваемая тематика освещается в дисциплине «Технология программирования», раздел «Архитектура программного обеспечения».

ЗАДАНИЕ

1. Проанализировать демонстрационную программу (или фрагмент этой/другой программы, предоставленный преподавателем).
2. Составить таблицу зависимостей.
3. Изобразить иерархию по образу рис. 4.1.

4. Усложнить рисунок, включив в него указанные преподавателем переменные.

Контрольные вопросы

1. Ограничено ли в языке Си количество возвращаемых функцией значений?
2. Ограничено ли в языке Си количество параметров функции?
3. Можно ли при анализе произвольной программы на языке Си учесть все возможные взаимосвязи?
4. Как в языке Си выглядит механизм вызова функций с использованием указателей?
5. Может ли пользовательская функция вызывать более одной другой функции?
6. Может ли пользовательская функция быть вызванной из нескольких других функций?
7. Можно ли осуществить взаимодействие между функциями, обеспечив взаимный обмен данными?
8. Какую роль играют глобальные переменные при информационном взаимодействии функций?
9. Охарактеризуйте термин «программный блок» в аспекте анализа программы на языке Си.
10. Что такое сопрограммы и как их можно описать в аспекте иерархии функций (модулей)?

5. Демонстрационная программа на языке С. Использование библиотеки стандартных функций

Популярность языка Си во многом определяется наличием богатого набора функций, разработанных коллективами программистов и охватывающих практически все значимые направления применения языка Си.

Эти готовые к использованию функции поставляются обычно вместе со средой разработки программного обеспечения и оформлены в виде так называемых *библиотек*.

Типичная библиотека содержит набор функций, объединенных общей тематикой. В рамках среды разработки фирмы Borland C (Turbo C) поставляются две библиотеки:

- графическая;
- системная.

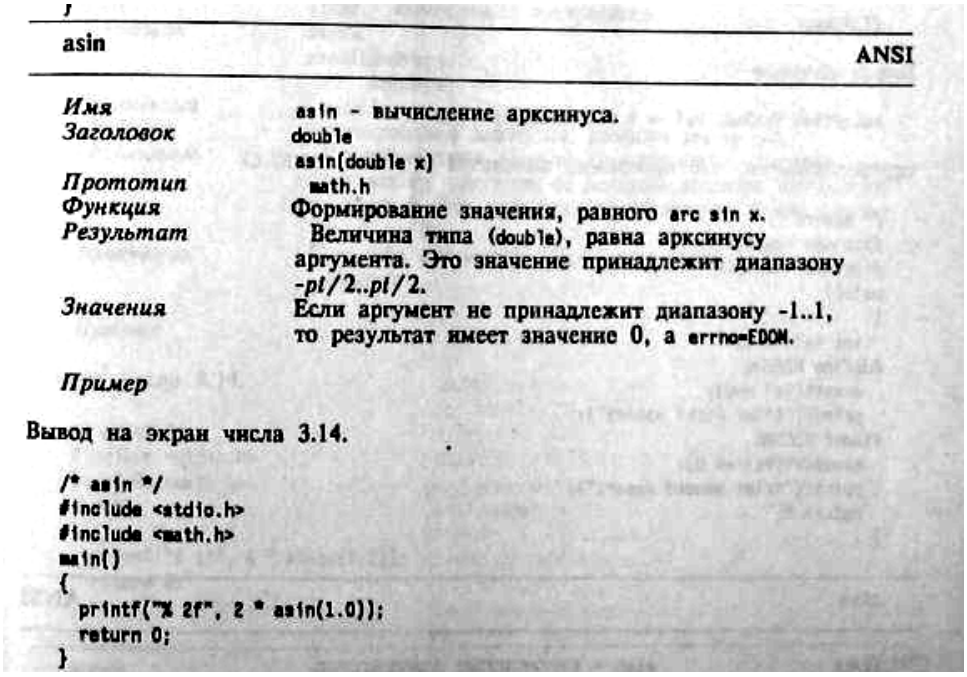
Системная (библиотека стандартных функций) содержит два вида функций: совместимые со стандартом ANSI и Turbo. Последние ориентированы на программную среду, оговоренную фирмой Borland (операционная система DOS на IBM PC совместимых компьютерах) и используют возможности этой среды. Стандарт ANSI можно считать платформно-независимым, т.е. при использовании совместимых с ним функций конечный продукт может быть использован под управлением других ОС и на другой аппаратной платформе.

Ввиду разнообразия функций по их предназначению, предпринимают меры разбиения их на группы с одинаково направленной функциональностью. Таким образом, функции различают по их принадлежности к *предварительно объявленным типам* stdio.h, stdlib.h, time.h, stdarg.h, setjmp.h и другим типам (math.h,..). Файлы с расширением *.h называют *прототипами* и указывают принадлежность к ним каждой библиотечной функции.

При использовании библиотечных функций необходимо предварительно указать компилятору список прототипов всех включенных в программу функций. Это достигается применением следующей конструкции языка:

```
#include <stdio.h>
#include "math.h"
...
```

В справочной литературе для каждой функции приводится ее описание:



asin ANSI

Имя asin - вычисление арксинуса.
Заголовок
double
asin(double x)
Прототип math.h
Функция Формирование значения, равного arc sin x.
Результат Величина типа (double), равна арксинусу аргумента. Это значение принадлежит диапазону *-pi/2..pi/2*.
Значения Если аргумент не принадлежит диапазону -1..1, то результат имеет значение 0, а errno=EDOM.

Пример

Вывод на экран числа 3.14.

```
/* asin */
#include <stdio.h>
#include <math.h>
main()
{
  printf("% 2f", 2 * asin(1.0));
  return 0;
}
```

Данный пример взят из книги [1].

ЗАДАНИЕ

1. Исследовать демонстрационную программу (или ее фрагмент по указанию преподавателя) на предмет наличия библиотечных функций, относящихся к системным.

2. Выписать названия всех найденных функций и разбить их по прототипам:

№	Функция	Прототип	Краткое описание

3. Для нескольких функций (по указанию преподавателя) найти и записать подробное описание.

Контрольные вопросы

1. Что такое библиотечная функция?
2. Какие функции включены в состав системной библиотеки?
3. Опишите свойства ANSI функций. Приведите пример такой функции.
4. Опишите свойства Turbo функций. Приведите пример такой функции.
5. Что такое прототип функции? Приведите описание одного из прототипов.
6. Для чего предназначена директива препроцессора #include?
7. Как увеличится код программы при подключении одного из разделов библиотек, например по директиве #include <math.h>?
8. Все ли функции, определенные стандартом ANSI, включены в состав библиотек Borland C (Turbo C)?

6. Демонстрационная программа на языке С. Использование библиотеки графических функций

В настоящее время (на момент написания методических указаний это 2017 год) практически все программное обеспечение использует графический режим при формировании пользовательского интерфейса. Это происходит во многом благодаря совершенствованию аппаратной части вычислительной техники. В прежних разработках, к которым относится и демонстрационная программа, использование графического интерфейса осуществлялось обычно в случаях, когда текстовый режим не был способен адекватно удовлетворить потребности пользователя. Обычной практикой было совмещение двух режимов: текстового для основной работы и графического для построения графиков или отображения рисунков. В демонстрационной программе реализован именно такой подход.

Для внедрения в программу графики использовалась графическая библиотека из поставки Borland C. В свою очередь, эта библиотека состоит из двух частей: библиотеки *текстовой графики* (прототип conio.h) и библиотеки *растровой графики* (прототип graphics.h).

При выборе и инициализации графического режима, по умолчанию весь экран представляет собой единое графическое окно. Координаты левого верхнего угла (0,0), а координаты противоположного угла могут быть определены вызовом соответствующих функций (getmaxx(), getmaxy()). Позже можно изменить графическое окно вызовом функции setviewport().

Для вывода на графическое окно текста можно использовать режим битового или режим штрихового текста. В первом случае качество шрифта ограничено матрицей 8x8 пикселей, во втором случае реализован масштабируемый векторный текст с возможностью выбора стиля шрифта (простой, индексный, тройной, готический). Однако для векторных шрифтов базовая поддержка ограничивается лишь основной частью таблицы ASCII.

Аппаратные возможности графики ограничены режимом VGA стандарта для IBM PC.

Подробную информацию о графическом режиме можно найти в [1].

ЗАДАНИЕ

1. Исследовать демонстрационную программу (или ее фрагмент по указанию преподавателя) на предмет наличия библиотечных функций, относящихся к графическим.

2. Выписать названия всех найденных функций и разбить их по прототипам:

№	Функция	Прототип	Краткое описание

3. Для нескольких функций (по указанию преподавателя) найти и записать подробное описание.

Контрольные вопросы

1. Каковы параметры графических режимов, использованных в демонстрационной программе?
2. Какие функции обеспечивают настройку графического режима?
3. Возможен ли вывод текстовой информации в графическом режиме?
4. Какие функции позволяют выводить текстовую информацию?
5. Какие функции использованы для вывода текстов в демонстрационной программе?
6. Какой вид шрифтов использован в демонстрационной программе?

7. Какие функции позволяют формировать изображение в графическом режиме?
8. Какие функции использованы для формирования изображения в демонстрационной программе?
9. Какого вида графическая информация отображается в демонстрационной программе?
10. Какие цветовые ограничения существуют при выводе графической информации в демонстрационной программе?
11. В какой степени графические возможности сопоставимы с используемыми в настоящее время?

7. Демонстрационная программа на языке С. Использование массивов и структур

Массивы в языке Си задаются в виде идентификатора требуемого типа, за которым в квадратных скобках следует размерность массива. Например, в нижеследующем выражении описан массив целых чисел с количеством элементов, равным 3.

```
int vec[3];
```

Элементы массивов нумеруются начиная с 0. Таким образом, массив vec[3] содержит элементы vec[0], vec[1], vec[2]. Каждый элемент данного массива имеет целочисленное значение.

Двумерный массив, описанный ниже, содержит 6 элементов целого типа:

```
int arr[3][2];
```

Массив arr можно представить в виде таблицы с тремя строками и двумя столбцами, или как массив двухэлементных векторов в количестве 3. Элементы этого массива обозначаются как arr[0,0]…arr[2,1]

Массивы в языке Си тесно связаны с указателями. Следующие два выражения для массива vec из примера эквивалентны:

vec[1]
*(vec+1)

Для двумерного массива mm[][] соответственно имеем:
(*(*(mm+i)) +j) эквивалентно mm[i][j]

Начальный элемент массива mm[0][0] может быть представлен как: (*(*(mm+0)) +0), что эквивалентно последующим записям после пошаговых упрощений:

((mm+0))
**(mm)
**mm

Имя массива с опущенными скобками в языке Си считается указателем на первый элемент массива. Например, для массива int vec[3] из примера запись **vec** является указателем на элемент vec[0].

Для определения адреса в памяти, в котором расположен элемент массива (или переменная) используется операция снятия адреса, обозначаемая символом «**&**».

Для определенных следующим выражением переменной и массива
int x, my[4];
имеем:
&x – адрес расположения переменной x
&my[0] или my - адрес начала массива my

Сказанное справедливо для массивов любых типов, но существуют особенности использования массивов символов, если они предназначены для хранения символьных строк. Функции, работающие с символьными строками, предполагают наличие специального ограничителя в конце символьной строки. В качестве такого ограничителя принят символ с шестнадцатеричным кодом 0x00. По этой причине максимальное число символов в символьной строке на единицу меньше размерности массива, а фактическое количество определяется положением кода 0x00 и может быть вычислено путем вызова специальной функции.

Структуры являются более сложной конструкцией языка Си. Элемент *структуры* может содержать набор данных произвольных типов. Поля структуры размещаются в памяти друг за другом в той последовательности, в которой перечислены в описании. Для объявления структуры язык Си предусматривает конструкцию вида:

```
struct
{
```

```
 поле1;
 поле2;
 …
}имя_структуры;
```

Приведем пример структуры:

```
struct
{
 char nazv[30];
 double stoim;
 int artikul;
} tovar;
```

Очевидно, назначение этой структуры – набор описаний параметров некоего товара, который включает в себя название товара, его стоимость и артикул.

К каждому элементу структуры можно получить доступ двумя способами:

```
 tovar->artikul = 123;
 tovar.artikul = 123;
```

Структурным элементом можно манипулировать как единым целым, что позволяет упростить организацию сложных данных.

Более подробную информацию о массивах и структурах можно найти в [1].

ЗАДАНИЕ

1. Исследовать демонстрационную программу (или ее фрагмент по указанию преподавателя) на предмет наличия массивов.

2. Выписать названия найденных массивов, определить их параметры и функциональное назначение.

№	Массив	Тип, размерность	Функциональное назначение

3. Исследовать демонстрационную программу (или ее фрагмент по указанию преподавателя) на предмет наличия структур.

4. Для нескольких структур (по указанию преподавателя) составить подробное описание.

Контрольные вопросы

1. Как организованы массивы в языке Си?
2. Какова связь между массивами и указателями?
3. В каких случаях элементы массива могут быть доступны из любой пользовательской функции?
4. Опишите способы доступа к элементу массива.
5. С какой целью в языке Си используют структуры?
6. Для каких целей используются структуры в демонстрационной программе?
7. Как вычислить размер одного элемента структуры?
8. Можно ли задать и использовать массивы структурных элементов?
9. Опишите способы доступа к элементу структуры.
10. Выразите свое мнение о необходимости использования структур в демонстрационной программе.

8. Демонстрационная программа на языке С. Направления улучшения и модернизации

В редких случаях программа полностью удовлетворяет потребителя. Обычные пожелания заказчика включают в себя:
- устранение обнаруженных несоответствий техническому заданию;
- устранение обнаруженных программных ошибок;
- расширение функциональности программы;
- адаптацию к изменившимся условиям эксплуатации;
- изменения пользовательского интерфейса.

Иногда встречаются случаи, когда изменения в программу вносятся разработчиком по своей инициативе:
- устранение ошибок;
- изменение (не всегда в лучшую сторону) пользовательского интерфейса;
- желание привлечь новых заказчиков;
- …

При передаче сопровождения другому программисту могут возникнуть пожелания:
- расширения описательных комментариев в программе;
- разработка или корректировка документации на программу;
- расширение номенклатуры документации;
-…

ЗАДАНИЕ

Изложить свое мнение по поводу необходимости внесения изменений в демонстрационную программу по направлениям:
- комментарии, их количество и качество;
- идентификаторы объектов, их понятность и логичность;
- типы данных, их номенклатура и структура;
- состав и номенклатура функций;
- пользовательский интерфейс;

- алгоритмы - ускорение вычислений, оптимизация кода;
- сопряжение с аппаратурой сбора данных – смена датчика на более «интеллектуальный»;
- прочие направления.

Контрольные вопросы

1. Заказчика не устраивает архаичность пользовательского интерфейса. Предложите необходимые изменения.
2. Заказчик приобрел операционную систему Windows нового поколения. Предложите необходимые изменения.
3. Заказчик желает расширить набор функций статистической обработки данных. Предложите необходимые изменения.
4. Заказчик сменил аппаратуру съема данным с пациента. В новом контроллере используется микро-ЭВМ и вывод информации осуществляется по интерфейсу RS-232 в цифровом виде. Предложите необходимые изменения.

Библиографический список

1. Белецкий Я. Энциклопедия языка Си: Пер. с польск.- М.: Мир, 1992.- 687 с., ил.
2. Искусство программирования на С. Фундаментальные алгоритмы, структуры данных и примеры приложений. Энциклопедия программиста: Пер. с англ./ Хэзфилд Ричард, Кирби Лоуренс и др. К.: Издательство «ДиаСофт», 2001. – 736 с.
3. Фролов А., Фролов Г. Библиотека системного программиста. Том 4. Программирование модемов.- М.: Диалог-МИФИ, 1993, 236 стр
4. Велоэргометрия: http://kard.doctortext.ru/page_3_6.php
5. Сердечный ритм: https://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D1%80%D0%B4%D0%B5%D1%87%D0%BD%D1%8B%D0%B9_%D1%86%D0%B8%D0%BA%D0%BB
6. Сопрограммы в языке программирования С. http://club.shelek.ru/viewart.php?id=338

Приложение А - Описание демонстрационной программы на языке Си.

Демонстрационная программа представляет собой программный продукт, написанный на языке Си для транслятора Borland C 3.0.

Дата разработки – апрель 1996 г.

Операционная система, в среде которой должна функционировать программа – DOS.

Назначение программы – сбор, накопление и предварительная обработка информации, собираемой в процессе обследования пациентов в кабинете функциональной диагностики. Данные собираются с датчиков, установленных на пациенте, проходящем сеанс велоэргометрии.

Смысл велоэргометрии заключается в проведении электрокардиографического исследования во время физической нагрузки на специальном велосипеде – велоэргометре. Велоэргометрия применяется для выявления начальных стадий и скрытых форм ишемической болезни сердца, а также для определения индивидуальной толерантности к физической нагрузке (см. Приложение Б).

При проведении массовых обследований необходимо вести документирование входных данных о пациенте и результатов его обследований. С этой целью в программе предусмотрена фиксация информации, касающаяся базовых сведений о пациенте:

- ФИО;
- возраст;
- профессия;
- предварительный диагноз.

Кроме этого, регистрируется дата проведения сеанса велоэргометрии.

Для удобства обработки информации в массовых масштабах в программе введен специальный вид данных, реализованный с помощью структуры:

```
struct pacient
 {
/* char o_time[20];*/
 char o_date[30];
 char name[40];
 char age[4];
 char profession[30];
 char diag[50];
 unsigned int rythm[200];
 }  pac={"Октябрь 1994 ",
      "Неизвестный В.Ф.",
      "46",
      "Водитель автокрана N123-45.",
      "Брадикардия..."
      };
```

Характер собираемых и обрабатываемых биомедицинских данных – сердечный ритм. Для его сбора используется специально разработанное устройство на базе микро-ЭВМ, воспринимающее сигналы с датчика и передающее их в персональный компьютер. Общие сведения о сердечном ритме описаны в Приложении Б.

Наиболее полную информацию о состоянии сердца можно получить на основе изучения электрокардиограммы, которая также регистрируется в сеансе велоэргометрии, но в нашем случае измеряются и запоминаются только длительности между сердечными сокращениями.

Цель написания программы с научно-медицинской точки зрения – накопление информации для выработки методики предварительного формирования диагноза, осуществляемого в оперативном режиме. Это позволит проводить массовое обследование в условиях, отличающихся от условий оснащенности кабинета функциональной диагностики (например, в полевых условиях).

Информационная обработка, заложенная в программу, позволяет работать со следующими математическими понятиями:

- наименьшее значение;
- наибольшее значение;
- среднее арифметическое;

- энтропия;
- относительная энтропия;
- асимметрия;
- эксцесс;
- мода;
- дисперсия;
- частота встречаемости.

Данная информация предоставляется оператору в удобном виде после проведения сеанса и предназначена для дальнейшего анализа с целью формирования критериев методики выявления, например, ишемии сердца.

Для удобства обработки такой информации в массовых масштабах в программе введен специальный вид данных, реализованный с помощью структуры:

```
struct stat
 {
  int   mini;              /* наименьшее значение */
  int   maxi;              /* наибольшее значение */
  float aver;              /* среднее арифмет. */
  float h;                 /* энтропия */
  float hr;                /* отн. энтропия */
  float asimm;             /* асимметрия */
  float exc;               /* эксцесс */
  int   moda;              /* мода */
  float sigma;            /* дисперсия */
  unsigned int   mm[256];  /* частота встречаемости */
 } st={1,1,1,1,1,1,1,80,1,
       0,0,0,0,0,0,0,0,0,0,
       0,0,0,0,0,0,0,0,0,0,
       0,0,0,0,0,0,0,0,0,0,
       0,0,0,0,0,0,0,0,0,0,
       0,0,0,0,0,0,0,0,0,0,
       0,0,0,0,0,0,0,0,0,0,
       0,0,0,0,0,0,0,0,0,0,5,
       10,20,30,40,50,60,70,80,90,80,
       70,60,50,40,30,20,10,0,0,0};
```

Данные передаются по стандартному последовательному каналу (интерфейс RS-232). В рамках DOS при этом удобно использовать так называемые FOSSIL драйверы. В нашем случае

FOSSIL драйвер используется на уровне следующих программных компонентов:

```
/*******************************************/ //
/* ПРИМИТИВЫ FOSSIL ДРАЙВЕРА      */          //
/**********************************/          //
// Инициализация драйвера                     //
int beg_FOSSIL(int comN,int *par)             //
{                                             //
…
}                                             //
                                  //
// Деинициализация драйвера                   //
void end_FOSSIL(int comN)                     //
{                                             //
…
}

// Прием символа с ожиданием
int get_FOSSIL(int comN)
{
…
}

// Передача символа с ожиданием
int put_FOSSIL(int comN,unsigned char sym)
{
…
}

// Установка параметров COM-порта
int set_FOSSIL(int comN,unsigned char par)
{
…
}                                                //
                                 //
// Управление линией DTR                         //
void dtr_FOSSIL(int comN,unsigned char dtr)      //
{                                                //
   asm mov ah,06h                                //
   asm mov dx,comN                               //
   asm mov al,dtr                                //
   asm int 14h                                   //
}                                                //
                                 //
// Сброс буфера приемника                        //
void clr_in_FOSSIL(int comN)                     //
{                                                //
```

```
        asm mov ah,0ah                                  //
        asm mov dx,comN                                 //
        asm int 14h                                     //
     }                                                  //
     /**
```

Возможности, предоставляемые программой, могут быть поняты из нижеследующего программного фрагмента, представляющего собой реализацию подсказки, вызываемой оператором при необходимости.

```
     /**********************/
     /*   H E L P          */
     /**********************/
     void  help(void)
     {
     printf("\n СИСТЕМА СБОРА И ОБРАБОТКИ ЭКГ Ver.%s
==COM%d==\n",Ver,Port+1);
     printf("U   - установка номера последовательного канала\n");
     printf("B   - сбор данных с АЦП\n");
     printf("L   - просмотр данных и выделение R-R интервалов \n");
     //printf("R   - долговременный сбор данных        РЕЖ 9\n");
     //printf("A   - амплитуда пульса                  РЕЖ 0\n");
     //printf("T   - гистограмма и функция в реальном времени  РЕЖ
9\n");
     printf("D   - проба \n");
     printf("2   - автоматическое выделение R-R (запись в
out.me2)\n");
     //printf("P   - считывание данных из пульсометра\n");
     //printf("L   - чтение файла \n");
     //printf("S   - запись файла\n");
     //printf("C   - расчет статистических данных\n");
     //printf("E   - редактор\n");
     //printf("G   - графики \n");
     printf("Q   - выход\n");
     }
```

Некоторые виды информации выводятся на экран в графическом виде, для чего используется стандартная графическая библиотека, поставляемая вместе с компилятором языка Си.

Все накопленные данные могут быть сохранены в виде файлов. Любой из созданных файлов может быть впоследствии считан для проведения предварительной математической обработки и графического просмотра результатов обработки.

Приложение Б - FOSSIL-ДРАЙВЕРЫ

Основные понятия

Название FOSSIL является набором первых символов из названий нескольких коммуникационных программ - "Fido/Opus/SEAdog Standard Layer". Эти программы используют FOSSIL драйверы для работы с асинхронным последовательным адаптером.

FOSSIL драйверы используются для расширения функций BIOS, обслуживающих асинхронный последовательный адаптер и модем. Кроме того, FOSSIL драйверы поддерживают несколько функций для работы с клавиатурой, видеоадаптером и системным таймером.

Использование FOSSIL драйверов позволяет увеличить скорость обмена через последовательный адаптер до 38400 бод (функции BIOS допускают максимальную скорость только 9600 бод).

FOSSIL драйвер самостоятельно обработывает прерывания от COM-портов. Он содержит два внутренних буфера, организованных в виде очереди.

В первый буфер - буфер передатчика - записываются данные, передаваемые компьютером внешниму устройству (модему). Драйвер самостоятельно определяет, когда асинхронный адаптер способен передать внешнему устройству очередной символ (т.е. когда свободен регистр данных COM-порта) и записывает его в регистр данных COM-порта. При этом переданный символ удаляется из буфера и происходит передача следующего символа.

Во второй буфер - буфер приемника - драйвер записывает данные, поступающие в компьютер через COM-порт. Затем содержимое этого буфера может быть считано программой при помощи специальной функции драйвера.

Примером такого FOSSIL драйвера может являться драйвер Gwinn's Communications Controller, X00.SYS Version V1.30. Вы можете получить любые FOSSIL драйверы и документацию на них практически на каждой станции BBS.

Существуют специальные FOSSIL драйверы, обеспечивающие программную эмуляцию аппаратных протоколов коррекции

ошибок - от MNP2 до MNP5. Дополнительные функции, поддерживаемые этими драйверами, мы рассмотрим позже.
Взаимодействие программы с FOSSIL драйвером
Интерфейс программ с FOSSIL драйвером обеспечивается через прерывание INT 14h. При этом FOSSIL драйвер подменяет встроенный обработчик прерывания INT 14h. FOSSIL драйвер программирует асинхронный адаптер непосредственно через обращение к его регистрам.

Функции, поддерживаемые FOSSIL драйвером

Установка скорости передачи данных
Передача символа с ожиданием
Прием символа с ожиданием
Определение состояния драйвера
Инициализация FOSSIL драйвера (COM-порта)
Деинициализация драйвера
Управление линией DTR
Определение параметров системного таймера
Передача данных
Сброс буфера передатчика
Сброс буфера приемника
Передача символа без ожидания
Чтение символа из буфера без удаления
Чтение символа из буфера клавиатуры
Чтение символа из буфера клавиатуры с ожиданием
Управление потоком

При связи двух устройств, работающих с различными скоростями, используют механизм управления потоком. Он подразумевает что приемное устройство, не справляющееся с обработкой поступающих ему данных, подает передающему устройству определенный сигнал. При поступлении в передающее устройство данного сигнала оно приостанавливает передачу и ожидает, пока приемное устройство не обработает принятые данные и не подаст сигнал, разрешающий возобновить передачу данных.
Вы можете выбрать тот или иной метод управления потоком, установив соответствующий бит регистра AL:

D0 - Использование для управления передачей символов XON(Ctrl-C)/XOFF(Ctrl-K). При установке данного бита FOSSIL драйвер будет приостанавливать дальнейшую передачу данных удаленному модему при получении символа XOFF. Для возобновления передачи необходимо передать драйверу символ XON.
D1 - Использование для управления потоком сигналов CTS/RTS. При установке данного бита FOSSIL драйвер будет приостанавливать дальнейшую передачу данных удаленному модему, если сигнал CTS переходит в неактивное состояние. Для возобновления передачи необходимо перевести линию CTS в активное состояние. FOSSIL драйвер будет также переключать линию RTS в неактивное состояние, когда буфер приемника будет заполнен на определенную величину.
D3 - Использование символов XON/XOFF для управления приемом данных. При установке данного бита FOSSIL драйвер будет передавать удаленному модему символ XOFF, когда буфер приемника драйвера заполнится на определенную величину. Когда программа считает символы из буфера приемника, удаленному модему будет передан символ XON, сигнализирующий, что передачу можно продолжить.
Дополнительная функция для управления потоком
Установить положение курсора
Определение текущего положения курсора
Вывод символа на экран
Отслеживание сигнала DCD
Вывод символа на экран (BIOS)
Установка и удаление функций, вызываемых по таймеру
Перезагрузка системы
Чтение блока данных из буфера драйвера в буфер программы
Запись блока данных из буфера программы в буфер драйвера
Передача сигнала BREAK
Получение информации о драйвере
Установка внешней функции обработчика
Отключение внешней функции обработчика

Материал изложен по книге [3].