



Урок 9

Файлы

Структурные типы данных



Программирование на языке Си
(базовый уровень)



Оглавление

Термины, используемые в лекции	2
Введение	2
Файлы текстовые	3
Позиционирование в файле	4
Особенности позиционирования в файле	5
Задачи	5
Множество сканирования fscanf	7
Разбор csv-файла с ошибками	8
Файлы бинарные	11
Режим работы с бинарным файлом	11
Отличие бинарных файлов от текстовых	11
Особенности бинарных файлов	12
Структуры	13
Инициализация структуры	13
Обращение к элементам структуры	14
Размер структуры	16
Задачи про студентов	16
Структуры и файлы	18
Задачи про датчик (курсовой)	19
Объединения	21
Перечисления	22
Битовые поля	23
Вычисление корней квадратного уравнения (структуры)	24
Объявление структуры	24
Глоссарий	26
Дополнительные материалы	27
Используемые источники	27

Термины, используемые в лекции

Файлы – это

Введение

На предыдущей лекции вы узнали:

- Что такое строки?

- Преобразование строки в массив байт
- Заголовочный файл `stdlib.h` и функции `atoi`, `atof`
- Массивы указателей, многомерные и VLA
- Сортировка `qsort` и чтение деклараторов

На этой лекции вы найдете ответы на такие вопросы как / узнаете:

- Как работать с файловой системой
- Файлы текстовые
- Бинарные файлы
- Множество сканирования `scanf` и `fscanf`
- Узнаем, что такое структуры, битовые поля и объединения и как с ними работать
- Подготовка к курсовому проекту

Файлы текстовые

Файл представляет собой произвольную последовательность данных. Содержимое файла может быть интерпретировано как последовательность текстовых символов или как двоичные данные (бинарные файлы).

Типы и функции для работы с файлами описаны в заголовочном файле `stdio.h`.

Чтобы работать с файлом, его необходимо открыть. Для открытия применяется функция `fopen()`, которая имеет следующий синтаксис:

```
FILE * fopen(имя_файла, режим_работы);
```

Допустимы значения второго аргумента, определяющего режим работы с файлом:

- "r" – существующий текстовый файл открывается для чтения;
- "w" – создается новый текстовый файл и открывается на запись. Если файл с таким именем существовал ранее, то его содержимое удаляется;

- "a" – текстовый файл открывается для записи в конец файла (его "старое" содержимое сохраняется) или создается;
- "r+" – существующий текстовый файл открывается для чтения и записи, текущая позиция (место в файле, по которому происходит чтение или запись) устанавливается в начало файла;
- "w+" – текстовый файл открывается или создается для чтения и записи, текущая позиция устанавливается в начало файла. Если файл с таким именем существовал ранее, то его содержимое удаляется;
- "a+" – текстовый файл открывается для чтения и записи, текущая позиция устанавливается в конец файла. Если файл не существовал, то он создается.

После завершения работы с файлом нужно закрыть. Для этого есть функция `fclose(указатель_на_файл)`:

```
#include <stdio.h>

int main(void)
{
    FILE *f;
    f = fopen("in.txt", "w"); // открытие файла in.txt на запись
    fclose(f); //Заккрытие файла. После окончания работы с файлом
    необходимо убедиться, что все записанные данные попали на диск, и
    освободить все ресурсы, связанные с ним.
    return 0;
}
```

Чтение и запись в файл — это буферизированный ввод и вывод.

Позиционирование в файле

ftell - смещение указателя на текущее положение в файле в байтах относительно начала файла. При ошибке функция возвращает -1.

```
long ftell(FILE *stream);
```

fseek - устанавливает текущую позицию в файле:

```
int fseek(FILE *stream, long offset, int origin);
```

```
//0 - ok, -1 - error
```

Позиция указателя, относительно которой будет выполняться смещение (origin)

- **SEEK_SET** – смещение в байтах отсчитывается относительно начала файла (параметр offset должен быть больше или равен 0);
- **SEEK_CUR** – смещение в байтах отсчитывается относительно текущей позиции в файле;
- **SEEK_END** – смещение в байтах отсчитывается относительно конца файла (offset должен быть меньше либо равен нулю).

```
fseek( ptrFile , 9 , SEEK_SET ); // изменить позицию на 9 байт  
относительно начала файла
```

Особенности позиционирования в файле

Если новое положение в файле находится до начала файла, возвращается ошибка.



Внимание! Если новое положение в файле находится за текущим концом файла, и файл открыт на запись или чтение/запись, файл расширяется нулями до требуемого размера.



Внимание! Функции позиционирования могут быть неприменимы к стандартным потокам, потому что стандартные потоки могут быть связаны с устройствами, которые не допускают произвольное позиционирование (например, терминалы).

Задачи

1. Дан текстовый файл in.txt, содержащий целые числа. Посчитать сумму чисел.

```
FILE *f;
int sum = 0, n;
f = fopen("in.txt", "r");
while (fscanf (f, "%d", &n) == 1)
    sum += n;
fclose (f);
printf ("%d\n", sum);
```

2. Ввести имя файла и напечатать его размер. Ввести имя файла и напечатать его размер. Функция ftell возвращает значение указателя текущего положения потока.

```
FILE *f;
static char filename[100]={0};
size_t size;
printf("Input file name: ");
scanf("%s",filename);
f = fopen (filename, "r");
if (f != NULL) {
    fseek (f, 0, SEEK_END);
    size = ftell (f);
    fclose (f);
    printf ("File size of '%s' - %lu bytes.\n",filename,
size);
} else {
    printf ("Can't open file %s\n", filename);
}
```

3. Дан текстовый файл in.txt. Необходимо посчитать количество цифр в файле и дописать это число в конец данного файла.

```
FILE *f;
int sum = 0, n;
signed char c; // обязательно signed! иначе заиклится
f = fopen("in.txt", "r+"); // режим чтение и дозапись
while ( (c = fgetc(f)) != EOF ) {
    if(c >= '0' && c <= '9') {
        sum += c - '0';
    }
}
fprintf (f, " %d", sum);
fclose (f);
```

Множество сканирования fscanf

В файле input.txt дана строка. Вывести ее в файл output.txt три раза через запятую и показать количество символов в ней

```
#include <stdio.h>

const int line_width = 256;

int main(void)
{
    char * input_fn = "input.txt";
    char * output_fn = "output.txt";
    char line[line_width];
    char c;
    FILE *fp;
    //
    if((fp = fopen(input_fn, "r")) == NULL)
    {
        perror("Error occured while opening input file!");
        return 1;
    }
    //
    int count = 0;
    while((c = getc(fp)) != EOF) && (c != '\n')
        line[count++] = c;
    line[count] = '\0';
    //
    fclose(fp);
    if((fp = fopen(output_fn, "w")) == NULL)
    {
        perror("Error occured while opening output file!");
        return 1;
    }
    //
    for (int i = 0; i < 3; i++)
    {
        if (i)
            fprintf(fp, ", ");
        fprintf(fp, "%s", line);
    }
    fprintf(fp, " %d", count);
    //
    fclose(fp);
    //
    return 0;
}
```

В файле input.txt дана строка из 1000 символов. Показать номера символов, совпадающих с последним символом строки. Результат записать в файл output.txt

```
#include <stdio.h>
#include <string.h>
#define MAXELEMENTS 1000
void input(char *string)
{
    FILE *in;
    in = fopen("input.txt", "r");
    fscanf(in, "%[^\n]", string);
    fclose(in);
}
void output(char *str)
{
    FILE *out;
    out = fopen("output.txt", "w");
    int len = strlen(str)-1;
    for(int i = 0; i < len; i++)
        if(str[i] == str[len])
            fprintf(out, "%d ", i);
    fclose(out);
}
int main(int argc, char **argv)
{
    char stringFile[MAXELEMENTS];
    input(stringFile);
    output(stringFile);
    return 0;
}
```

Разбор csv-файла с ошибками

Текстовый файла csv, состоит из строк следующего формата:

YEAR;MONTH;DAY;HOUR;MINUTE;TEMPERATURE

dddd;mm;dd;hh;mm;temperature

dddd - год 4 цифры

mm - месяц 2 цифры

dd - день 2 цифры

hh - часы 2 цифры

mm - минуты 2 цифры

temperature - целое число от -99 до 99

Рассмотрим более упрощенный вариант файла. Файл с ошибками temperature_small1.csv в формате YEAR;MONTH;DAY;

2021;01;16	20;06;16
2021;01;16	2021;07;16
2021;01;16	2021;08;16
2021;01;16	2021;09;16
2021;02;xx	2021;10;16
2021;02;17	2021;11;16
2021;aa;16	2021;12;16
2021;04;16	
2021;05;16	

Разбор файла с ошибками

<pre>#include <stdio.h> #include <conio.h> #define N 3 int main(int argc, char **argv) { FILE *open; char name[] = "temperature_small1.csv"; open = fopen(name, "r"); if (open==NULL) return 1; int Y,M,D; int r;</pre>	<pre>while((r = fscanf(open, "%d;%d;%d",&Y,&M,&D))>0){ if(r<N) { char s[20],c; r = fscanf(open, "%[^\\n]%c", s,&c); printf("ERROR %d=%s%x\\n",r,s,c); } else printf("%d = %d;%d;%d\\n", r,Y,M,D); } return 0; }</pre>
---	--

Разбор файла с ошибками пропускаем “плохие” строки с помощью fgetc

<pre> #include <stdio.h> #include <conio.h> #define N 3 int main(int argc, char **argv) { FILE *open; char name[] = "temperature_small1.csv"; open = fopen(name, "r"); if(open==NULL) return 1; int Y,M,D; int r; char ch; while((r = fscanf(open, "%d;%d;%d",&Y,&M,&D))>0) { </pre>	<pre> if(r<N) { do { putchar(ch); ch=fgetc(open); }while(EOF!=ch && '\n'!=ch); // пропускаем оставшиеся символы до конца файла или строки (на случай, если строка была чересчур длинной } else printf("%d = %d;%d;%d\n", r,Y,M,D); } return 0; } </pre>
---	--

Разбор файла с ошибками без scanf

<pre> int counter = 0; int arr[3] = {0}; int line=0; while((ch=fgetc(open)) !=EOF) { if(ch==';') { counter++; if(counter>N) counter = 0; } else if(ch=='\n') { if(counter==2) printf("%d = %d;%d;%d\n", counter, arr[0],arr[1],arr[2]); </pre>	<pre> else printf("ERROR %d = %d;%d;%d\n", counter,arr[0],arr[1],arr[2]); counter = 0; arr[0]=arr[1]=arr[2]=0; } else if(ch>='0' && ch<='9') arr[counter] = arr[counter]*10 + ch - '0'; else if(ch != 0xD) counter = 0; } </pre>
---	---

Файлы бинарные

Режим работы с бинарным файлом

```
FILE * fopen(имя_файла, режим_работы) ;
```

Для открытия файла в бинарном режиме к значению второго аргумента приписывается буква b, например, "rb" означает открытие существующего бинарного файла на чтение, а "a+b" (можно "ab+") – открытие бинарного файла для чтения и записи с позиционированием в конец файла.

- "wb": бинарный файл открывается для записи
- "rb": бинарный файл открывается для чтения
- "ab": бинарный файл открывается для дозаписи
- "w+b": бинарный файл создается для записи/чтения
- "r+b": бинарный файл открывается для чтения/записи
- "a+b": бинарный файл открывается или создается (при его отсутствии) для чтения/дозаписи

Отличие бинарных файлов от текстовых

Отличие текстового режима от бинарного в том, что в текстовом режиме некоторые последовательности символов могут восприниматься особым образом. Набор специальных последовательностей и их интерпретация зависят от операционной системы. На операционной системе Windows перевод строки в текстовых файлах записывается как последовательность из двух символов: символ с кодом 13 и символ с кодом 10. В случае открытия файла как текстового, данная последовательность будет отображаться как один символ '\n', и именно в таком виде будет прочитана функциями стандартной библиотеки. В случае же открытия

файла как бинарного, она будет состоять из двух байт со значениями 13 и 10.

```
1 2021;01;16;01;01;-47
2 2021;01;16;01;03;-44
3 2021;01;16;01;04;-43
4 2021;01;16;01;05;-xx
```

Файл	Правка	Вид	Кодировка	Справка
00000000:	32 30 32 31 3B 30 31 3B	31 36 3B 30 31 3B 30 31		2021;01;16;01;01
00000010:	3B 2D 34 37 0D 0A 32 30	32 31 3B 30 31 3B 31 36		;-47..2021;01;16
00000020:	3B 30 31 3B 30 33 3B 2D	34 34 0D 0A 32 30 32 31		;01;03;-44..2021
00000030:	3B 30 31 3B 31 36 3B 30	31 3B 30 34 3B 2D 34 33		;01;16;01;04;-43
00000040:	0D 0A 32 30 32 31 3B 30	31 3B 31 36 3B 30 31 3B		..2021;01;16;01;
00000050:	30 35 3B 2D 78 78 0D 0A	32 30 32 31 3B 30 32 3B		05;-xx..2021;02;

Особенности бинарных файлов

При работе с бинарными файлами не возможно использование текстовых функций, таких как `fprintf/fscanf` или `fgets/fputs`, потому что они воспринимают данные записанные в файле как текст - последовательность строковых данных. Бинарные же файлы содержат двоичные данные ровно в том виде как они хранятся в памяти компьютера. На 32-ух битной архитектуре с целым типом 4 байта, данные хранятся в формате little-endian: нулевой байт числа - самый последний.

```
unsigned int i = 0x31323334;
//little-endian - 0x34 0x33 0x32 0x31
//                                &i          +1          +2          +3
```

Функции чтения и записи бинарных файлов

Для чтения из файла данных в двоичном виде используются функции `fread`.

```
size_t fread (const void *ptr, size_t size, size_t nmemb, FILE
*fp);
// return value = read_bytes / size; Задавайте размер считываемых
данных всегда равным 1, иначе не будет ясно, что он был записан.
```

Функция `fread` считывает данные из бинарного файла. Параметр `ptr` — это адрес начала буфера, в который будут записаны считанные данные. `size` — это размер одного элемента данных, а `nmemb` — это количество элементов данных, которые необходимо прочитать. `fp` — дескриптор потока, из которого ведётся чтение

Для записи в файл данных в двоичном виде используются функции `fwrite`.

```
size_t fwrite(const void *ptr, size_t size, size_t nmemb, FILE
*fp);
// Возвращаемое значение получается делением нацело действительно
записанного количества байт на размер одного элемента данных.
```

Пример:

```
unsigned int i = 0x31323334, u=0;
FILE *f = fopen ("out.bin", "wb");
fwrite (&i, sizeof (int), 1, f); // данные запишутся в
формате little-endian
fclose(f);

f = fopen ("out.bin", "rb");
fread (&u, 1, 1, f);
fclose(f);
printf("u = %x\n",u); // u = 34
```

Структуры

Данный тип позволяет составлять из нескольких стандартных типов какой-то другой тип, и работать с ним как с единым целым. Структура состоит из полей стандартного типа, или же другой структуры.

```
struct имя_структуры
{
    компоненты_структуры
};
```

Над структурами определены все те же операции, что и над полями из которых они состоят. Рассмотрим пример объявления структурного типа

```
struct student {
    uint8_t name[50]; // массив из 50 символов
    uint32_t group; // целое число
    uint8_t country[30]; // массив из 30 символов
} student1, student2; // переменные с типом struct student

struct student course[200]; // массив из 200 структур
struct student *pst; // указатель на struct student
```

Инициализация структуры

Существует два способа инициализации структуры:

По позиции — значения указываются в операторных скобках передаются, в том порядке, в котором они следуют в структуре

```
struct student student1 = {"Ivan", 1, "Russia"};
```

По имени — значения передаются по именам элементов структуры и не зависят от порядка

```
struct student student1 = {.name="Ivan", group=1, country="Russia"};
```

Можно одновременно совмещать определение типа структуры, ее переменных и сразу же инициализировать структуру

```
#include <stdint.h>
struct student {
    uint8_t name[50]; // массив из 50 символов
    uint32_t group; // целое число
    uint8_t country[30]; // массив из 30 символов
} student1 = {"Ivan", 1, "Russia"}; // переменные с типом struct
student
```

Удобно объявить тип с помощью оператора typedef тогда не нужно каждый раз писать ключевое слово struct

```
#include <stdint.h>
typedef struct student {
    uint8_t name[50]; // массив из 50 символов
    uint32_t group; // целое число
    uint8_t country[30]; // массив из 30 символов
} student;
student student1 = {"Ivan", 1, "Russia"}; // переменные с типом
student
```

Обращение к элементам структуры

Можно обращаться к элементам переменной типа “структура”, т.е. получать их значения или, наоборот, присваивать им новые значения.

Для обращения к полю структуры используют оператор “.” (точка)

```
student1.group = 101; // в переменную student1 поле group  
положили 101
```

При обращении к структуре через указатель можно также использовать оператор разыменования - "*" или оператор стрелка - "->". У операторов точка и стрелка самый высокий приоритет. Следующие два примера эквивалентны

```
student1.group = 101;  
pst = &student1;  
printf("%d\n", pst->group);
```

```
student1.group = 101;  
pst = &student1;  
printf("%d\n", (*pst).group);
```

Разрешается присваивать структуры одного типа, при этом все поля копируются.

```
student1.group = 101;  
strcpy(student1.name, "Ivan");  
strcpy(student1.country, "Russia");  
student2 = student1;  
printf("%s\n", student2.name);
```

?



```
student1.group = 101;  
strcpy(student1.name, "Ivan");  
strcpy(student1.country, "Russia");  
student2 = student1;  
printf("%s\n", student2.name);
```

Ivan



Внимание! Операции сравнения над структурами не определены.

Размер структуры

Размер структуры (sizeof) всегда не меньше суммы размеров ее полей. Размер структуры зависит от порядка следования описания ее полей.

<pre>struct str1 { uint32_t u; uint8_t c1; int32_t i; uint8_t c2; } s1; struct str2 { uint32_t u; int32_t i; uint8_t c1; uint8_t c2; } s2; printf("Sizeof s1 = %lu\n", sizeof(s1)); printf("Sizeof s2 = %lu\n", sizeof(s2));</pre>	<pre>Sizeof s1 = 16 Sizeof s2 = 12</pre>
--	--

Задачи про студентов

Описать структуру для представления информации о человеке: фамилия (не более 30 символов), имя (не более 30 символов), возраст. Описать функцию, которая для заданного массива из описанных структур определяет:

- Возраст самого старшего человека
- Количество людей с заданным именем (имя также является параметром функции)
- Количество людей, у которых есть однофамильцы


```

#include <stdio.h>
#include <stdint.h>
#include <string.h>
#define STR_SIZE 30
#define STUDEN_NUMBER 200
struct student {
    char surname[STR_SIZE];
    char name[STR_SIZE];
    uint8_t age;
};
//возраст самого старшего человека;
int Eldest(struct student*
course,int number)
{
int max = course->age;
for(int i=1;i<number;i++)
    if(max < (course+i)->age)
        max = (course+i)->age;
return max;
}

```

```

//количество людей с заданным
именем (имя также является
параметром функции);
int SameNameNumber(struct student*
course,int number,char* name)
{
int counter = 0;
for(int i=0;i<number;i++)

if(!strcmp(course[i].name,name))
    counter++;
return counter;
}

```

```

//количество людей, у которых есть однофамильцы;
int Namesakes(struct student* course,int number)
{
int counter=0;
for(int i=0;i<number;i++)
    for(int j=i+1;j<number;j++)
        if(!strcmp(course[i].surname,
course[j].surname))
        {
            counter++;
            break;
        }
return counter;
}
void AddStudent(struct student* course, int number,
char* surname, char* name, int age)
{
    course[number].age = age;
    strcpy(course[number].name,name);
    strcpy(course[number].surname,surname);
}
void print(struct student* course,int number)
{
    for(int i=0;i<number;i++)
        printf("%s\t%s\t%d\n",
course[i].surname,
course[i].name,
course[i].age
);
}

```

```

}
int AddCourse(struct student* course)
{
    int c=0;
    AddStudent(course,c++, "Ivanov", "Ivan", 18);
    AddStudent(course,c++, "Petrov", "Ivan", 19);
    AddStudent(course,c++, "Petrov", "Ivan", 19);
    AddStudent(course,c++, "Petrov", "Ivan", 19);
    AddStudent(course,c++, "Petrov", "Ivan", 19);
    AddStudent(course,c++, "Ivanov", "Vasily", 44);
    return c;
}

```

```

int main(void)
{
    struct student course1[STUDEN_NUMBER]; // массив из 200 структур
    struct student course2[STUDEN_NUMBER]; // массив из 200 структур
    int number1=AddCourse(course1);
    int number2=AddCourse(course2);
    print(course1,number1);
    printf("Eldest student = %d\n",Eldest(course1,number1));
    char* name = {"Ivan"};
    printf("Name %s number =
%d\n",name,SameNameNumber(course1,number1,name));
    printf("Same surname number = %d\n",Namesakes(course1,number1));

    return 0;
}

```

Структуры и файлы

Удобно загружать и выгружать структуры в бинарные файлы

```

int SaveFileCourse(int
number,struct student*
course,char* filename)
{
    FILE *f = fopen (filename,
"wb");
    int r = 0;
    if(f)
        r = fwrite(course,
sizeof(struct student), number,
f);
    fclose(f);
    return r;
}

```

```

int LoadFileStruct(int
number,struct student*
course,char* filename)
{
    FILE *f = fopen (filename,
"rb");
    int r = 0;
    if(f)
        r = fread(course,
sizeof(struct student),
number, f);
    fclose(f);
    return r;
}

```

Загрузим `struct student course1[STUDEN_NUMBER];` в файл `"course1.bin"`, затем загрузим из файла `struct student course2[STUDEN_NUMBER];`

```
int main(void)
{
    struct student course1[STUDEN_NUMBER]; // массив из 200 структур
    struct student course2[STUDEN_NUMBER]; // массив из 200 структур
    int number1=AddCourse(course1);
    print(course1,number1);
    int number2=number1;

    printf("=%d\n",SaveFileCourse(number1,course1,"course1.bin"));

    printf("=%d\n",LoadFileStruct(number2,course2,"course1.bin"));
    print(course2,number2);
    return 0;
}
```

Представление данных в файле

Файл	Правка	Вид	Кодировка	Справка
00000000:	49 76 61 6E 6F 76 00 00	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	Ivanov.....
00000010:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	00 00 00 00 00 00 49 76Iv
00000020:	61 6E 00 00 00 00 00 00	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	an.....
00000030:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	12 50 65 74Pet
00000040:	72 6F 76 00 00 00 00 00	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	rov.....
00000050:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	49 76 61 6E 00Ivan.
00000060:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
00000070:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	13 50 65 74 72 6F 76Petrov
00000080:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
00000090:	00 00 00 00 00 00 00 00	49 76 61 6E 00 00 00 00	00 00 00 00 00 00 00 00Ivan....
000000A0:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
000000B0:	00 00 00 00 00 00 00 00	13 50 65 74 72 6F 76 00 00 00	00 00 00 00 00 00 00 00Petrov...
000000C0:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
000000D0:	00 00 00 00 00 00 49 76 61 6E 00 00 00 00 00 00	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00Ivan.....
000000E0:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
000000F0:	00 00 00 13 50 65 74 72 6F 76 00 00 00 00 00 00	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00Petrov.....
00000100:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
00000110:	00 00 49 76 61 6E 00 00	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	..Ivan.....
00000120:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
00000130:	13 49 76 61 6E 6F 76 00	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	.Ivanov.....
00000140:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 56U
00000150:	61 73 69 6C 79 00 00 00	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	asily.....
00000160:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	2C,

Задачи про датчик (курсовой)

Описать структурный тип для представления сбора информации с датчика температуры, необходимые поля: дата (день, месяц, год) и температура. Используя

этот тип, описать функцию, принимающую на вход массив таких данных и упорядочивающую его по возрастанию, по дате

```
#include <stdio.h>
#include <stdint.h>
#include <string.h>
#define SIZE 30
struct sensor {
    uint8_t day;
    uint8_t month;
    uint16_t year;
    int8_t t;
};
void cgangeIJ(struct sensor* info,int
i,int j)
{
    struct sensor temp;
    temp=info[i];
    info[i]=info[j];
    info[j]=temp;
}
//упорядочивающую его по неубыванию
температуры
void SortByT(struct sensor* info,int
n)
{
    for(int i=0; i<n; ++i)
        for(int j=i; j<n; ++j)
            if(info[i].t>=info[j].t)
                cgangeIJ(info,i,j);
}
```

```
unsigned int DateToInt(struct
sensor* info)
{
    return info->year << 16 |
info->month << 8 |
        info->day;
}
//упорядочивающую его по дате
void SortByDate(struct sensor*
info,int n)
{
    for(int i=0; i<n; ++i)
        for(int j=i; j<n; ++j)

if(DateToInt(info+i)>=
        DateToInt(info+j))

cgangeIJ(info,i,j);
}
void AddRecord(struct sensor*
info,int number,
uint16_t year,uint8_t
month,uint8_t day,int8_t t)
{
    info[number].year = year;
    info[number].month = month;
    info[number].day = day;
    info[number].t = t;
}
```

<pre> int AddInfo(struct sensor* info) { int counter=0; AddRecord(info,counter++,2021,9,16,9); AddRecord(info,counter++,2022,9,2,-9); AddRecord(info,counter++,2021,1,7,8); AddRecord(info,counter++,2021,9,5,1); return counter; } void print(struct sensor* info,int number) { printf("=====\n"); for(int i=0;i<number;i++) printf("%04d-%02d-%02d t=%3d\n", info[i].year, info[i].month, info[i].day, info[i].t); } </pre>	<pre> int main(void) { struct sensor info[SIZE]; int number=AddInfo(info); print(info,number); printf("\nSort by t\n"); SortByT(info,number); print(info,number); printf("\nSort by date\n"); SortByDate(info,number); print(info,number); return 0; } </pre>
---	--

Объединения

Описание объединений в Си аналогично описанию структур, с той лишь разницей что используем зарезервированное слово **union**. В объединении все поля располагаются начиная с одного и того же адреса. Изменение одного поля влечет за собой изменение всех остальных полей. В каждый конкретный момент времени имеет смысл только одно поле, какое именно решать вам.

```

union u {
    int i;
    char ch[4];
    float f;
}

```

Удобно использовать объединения для доступа к отдельным частям значений

```

union intbytes {
    uint32_t number;
    uint8_t bytes[4];
}

```

```

} d;
    d.number = 0x12345678;
    printf ("Number  %x",d.number);
    printf(" in memory is: %x %x %x %x\n", d.bytes[0],
d.bytes[1], d.bytes[2], d.bytes[3]);
// Number 12345678 in memory is: 78 56 34 12

```

Перечисления

Перечисления — это тип данных, которые содержат набор констант, и каждой константе сопоставлено определенное числовое значение.

Перечисление определяется с помощью ключевого слова enum:

```

enum название_перечисление { константа1, константа2, ...
константаN };

```

Каждой константе присваивается числовое значение по умолчанию начиная с 0:

- первой константе - число 0,
- второй константе - число 1,
- третьей - 2 и так далее.

Также можно явным образом присвоить константам числовые значения.

Пример перечислений

<pre> #include <stdio.h> enum operation // арифметическая операция { ADD = 1, // сложение SUB = 2, // вычитание MUL = 4 // умножение }; int calculate(int x, int y, enum operation op) { switch(op) { case ADD: return x + y; case SUB: return x - y; case MUL: return x * y; default: return 0; } } </pre>	<pre> int main(void) { enum operation op = MUL; int result = calculate(5, 6, op); printf("Result: %d\n", result); // Result: 30 result = calculate(7, 8, ADD); printf("Result: %d\n", result); // Result: 15 result = calculate(4, 3, SUB); printf("Result: %d\n", result); // Result: 1 return 0; } </pre>
--	--

<pre> } } </pre>	<pre> } </pre>
------------------	----------------

Битовые поля

- Битовые поля обеспечивают удобный доступ к отдельным битам данных
- Битовое поле не может существовать само по себе. Оно может быть только элементом структуры или объединения
- Позволяют формировать объекты с длиной, не кратной байту

```

struct имя_структуры
{
    тип1 имя_поля1 : ширина_поля1;
    тип2 имя_поля2 : ширина_поля2;
    //.....
    типi имя_поляi : ширина_поляi;
}

```

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

```
struct point
```

```
{
    unsigned int x:5;    // 0-31
    unsigned int y:3;    // 0-7
};
```

7	6	5	4	3	2	1	0
1	0	1	0	0	0	1	0
point.y				point.x			

```
int main(void)
```

```
{
    struct point center = {0, 5};
    center.x = 2;
    printf("x=%d    y=%d \n", center.x, center.y);    // x=2    y=5
    return 0;
}
```

Вычисление корней квадратного уравнения (структуры)

Объявление структуры

Перепишем программу с применением структур

```
#include <stdio.h>
#include <conio.h>
#include <locale.h>
#include <math.h>
#include <stdlib.h>

typedef struct Equation
{
enum {COEF_N=3, ROOT_N=2} SIZE;
enum {NO_ROOTS=-1, COMPLEX_ROOTS=0, C=2, B=1, A=0} ROOTS;
float coef[COEF_N];
float roots[ROOT_N];
int rootsNumber;
float sqrD;
float B;
} Equation;
```

Ввод данных

```
float InputFloat(const char* message)
{
float number;
printf("%s", message);
scanf("%f", &number);
return number;
}

void Input(Equation* equation)
{
const char* str[]={"Введите a:\n", "Введите b:\n", "Введите c:\n", NULL};
printf("Вычисление корней квадратного уравнения \\\n"
"a*x*x+b*x+c=0\\n");
for(int i=0; i<COEF_N; i++)
equation->coef[i] = InputFloat(str[i]); //1//18//32
}
```

Функция вывода

```
void Print(Equation *equation)
{
switch(equation->rootsNumber)
```



```

{
    case 2:
        printf("Корни квадратного уравнения \n");
        printf("X1 = %f \n",equation->roots[0]);
        printf("X2 = %f \n",equation->roots[1]);
        break;
    case 1: printf("Корень линейного уравнения
%f\n",equation->roots[0]); break;
    case COMPLEX_ROOTS:
        printf("Корни квадратного уравнения комплексные \n");
        break;
    case NO_ROOTS: printf("Корней НЕТ!\n"); break;
    default:      printf("Ошибка количества
корней%d!\n",equation->rootsNumber);
}
}

```

Вычисление корней квадратного уравнения

<pre> float sqr(float x) { return x*x; }; void CalcRealRoots(Equation *e) { float d = sqrtf(e->sqrD); //2//16 e->roots[0] = (-e->B + d)/e->coef[A]; e->roots[1] = (-e->B - d)/e->coef[A]; } void CalcRoots(Equation *e) { e->rootsNumber=0; e->B = e->coef[B]/2; if(e->coef[A]!=0) { e->sqrD = sqr(e->B) - e->coef[A]*e->coef[C]; if(e->sqrD<0) e->rootsNumber=COMPLEX_ROOTS; } } </pre>	<pre> else { e->rootsNumber=2; CalcRealRoots(e); } else { if(e->coef[B]!=0) { e->roots[0] = -e->coef[C]/e->coef[B]; e->rootsNumber=1; } else e->rootsNumber=NO_ROOTS; } } </pre>
--	---

```

void CalcRootsTest(Equation *equation)
{
    equation->roots[0]    = 1;
    equation->roots[1]    = 2;
    equation->rootsNumber = 2;
}

void SquareEquation(void)
{
    Equation* equation = malloc(sizeof(Equation)); //new в C++
    Input(equation);
}

```

```

    //CalcRootsTest(equation);
    CalcRoots(equation);
    Print(equation);
    free(equation);
}

```

Вычисление корней квадратного уравнения (точка входа)

```

int main(int argc, char **argv)
{
    char Choice;
    setlocale(LC_ALL, "Rus");
    while(1)
    {
        printf("1. Вычисление
корней квадратного
уравнения\n");
        printf("0. Выход\n");
        printf("Для выход
нажмите Q\n");
        Choice = getch();
    }
}

```

```

switch(Choice)
{
    case '1':
        SquareEquationNew();
        break;
    case '0':
    case 'q':
    case 'Q':
        return 0;
    break;
    default:
        printf("Непонятный выбор
%x\n",Choice);
        break;
}
}
return 0;
}

```

Глоссарий

Компилятор - Получает из каждой единицы трансляции код на машинном языке (Ассемблер) и генерирует объектный модуль с машинным кодом.

Препроцессор - компонент, производящий набор текстовых подстановок над файлом для получения его окончательного вида и передачи компилятору. Подстановка текстов заголовочных файлов (директива #include), условная трансляция, макроподстановки


Линковщик - обеспечивает слияние нескольких объектных файлов в один исполняемый.

GNU — операционная система типа Unix, но отличная от Unix тем, что является свободной и не содержит его кода.

Linux - семейство Unix-подобных операционных систем на базе ядра Linux, включающих тот или иной набор утилит и программ проекта GNU, и, возможно, другие компоненты.

Unix - семейство переносимых, многозадачных и многопользовательских операционных систем, которые основаны на идеях оригинального проекта AT&T. Разработанного в 1970-х годах в исследовательском центре Bell Labs Кеном Томпсоном, Деннисом Ритчи и другими.

Дополнительные материалы

1. Домашнее задание задачи B1 - B21
2. Оформление программного кода
<http://www.stolyarov.info/books/pdf/codestyle2.pdf>
3. Стандарт разработки программного обеспечения MISRA на языке C
http://easyelectronics.ru/files/Book/misra_c_rus.pdf
4. [fflush – функция языка Си. "Все о Hi-Tech"](#)
5.  Программирование на языке C. Урок 18. Установка и настройка GCC на V...
6. [C/C++ на Linux в Visual Studio Code для начинающих](#)
7. [О квадратных уравнениях в правильном порядке / Хабр](#)
8. [Онлайн калькулятор. Решение квадратных уравнений](#)
- 9.

Используемые источники

1. cprogramming.com - учебники по [C](#) и [C++](#)
2. free-programming-books - ресурс содержащий множество книг и статей по программированию, в том числе по [C](#) и [C++](#) (там же можно найти ссылку на распространяемую бесплатно автором html-версию книги Eckel B. «Thinking in C++»)
3. tutorialspoint.com - ещё один ресурс с множеством руководств по изучению различных языков и технологий, в том числе содержит учебники по [C](#)

4. Юричев Д. [«Заметки о языке программирования Си/Си++»](#) - «для тех, кто хочет освежить свои знания по Си/Си++»
5. [Онлайн версия «Си для встраиваемых систем»](#)