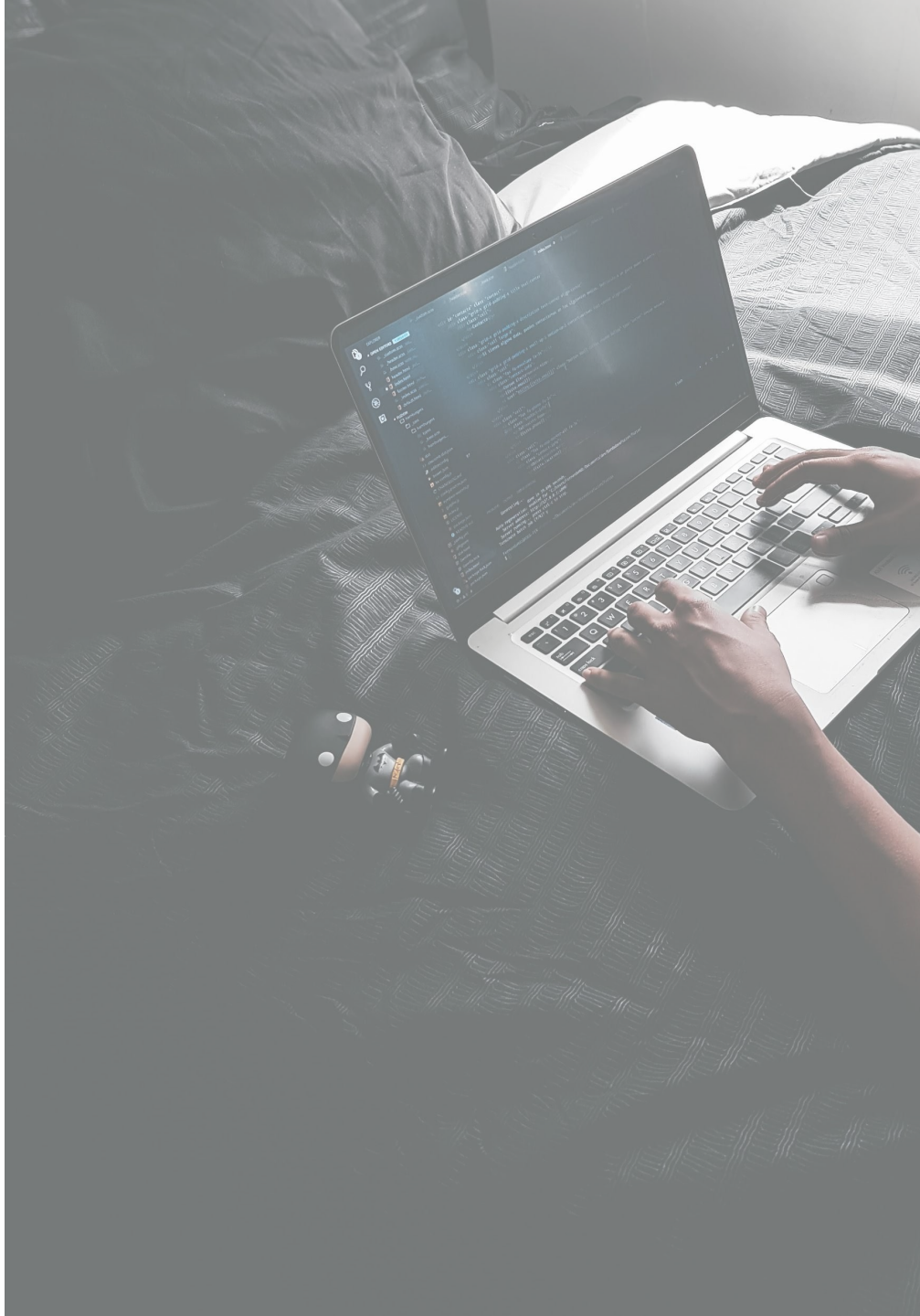
GeekBrains

Урок 10. Структурные типы данных. Файлы



Урок 10

На этом уроке

- Как работать с файловой системой
- 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)

```
#include <stdio.h>
int main(void)
{
    FILE *f;
    f = fopen("in.txt","w"); // открытие файла in.txt на запись
    fclose(f);  //Закрытие файла. После окончания работы с файлом
необходимо убедиться, что все записанные данные попали на диск, и
освободить все ресурсы, связанные с ним.

    return 0;
}
```

Режим работы с текстовым 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

Режим работы с текстовым файлом

- "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 создаётся

Режим работы с бинарным файлом

Для открытия файла в бинарном режиме к значению второго аргумента приписывается буква b

Например:

- "rb" означает открытие существующего бинарного файла на чтение
- "a+b" (можно "ab+") – открытие бинарного файла для чтения и записи с позиционированием в конец файла

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 нуля)

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

Особенности позиционирования в файле

- Если новое положение в файле находится за текущим концом файла, и файл открыт на запись или чтение/запись, файл расширяется нулями до требуемого размера
- Если новое положение в файле находится до начала файла, возвращается ошибка
- Функции позиционирования могут быть неприменимы к стандартным потокам, потому что стандартные потоки могут быть связаны с устройствами, которые не допускают произвольное позиционирование (например, терминалы)

Задача

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);
```

Задача

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);
}
```

Задача

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

```
FILE *f;
int sum = 0, n;
signed char c;// обязательно signed! иначе зациклится
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);
```

Задача

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

```c
#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;
}
```

Пример

GeekBrains

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

```c
#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);
```

```c
   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;
}
```

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

Особенности бинарных файлов

- При работе с бинарными файлами использование текстовых функций (таких как fprintf/fscanf или fgets/fputs) невозможно, потому что они воспринимают данные^ записанные в файле, как текст — последовательность строковых данных
- Бинарные же файлы содержат двоичные данные ровно в том виде, в котором 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

Файлы бинарные

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
```

Файлы бинарные

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.
```

Файлы бинарные

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

- Позволяют составлять из нескольких стандартных типов какой-то другой тип и работать с ним как с единым целым
- Состоит из полей стандартного типа или же другой структуры
- Определены все те же операции, что и надо полям, из которых они состоят

```
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
```

Структуры

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

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

- При обращении через указатель можно использовать оператор разыменования — “*” или оператор стрелка — “->”
- У операторов точка и стрелка самый высокий приоритет
- Разрешается присваивать структуры одного типа, при этом все поля копируются
- Операции сравнения над структурами не определены
- Размер структуры (sizeof) всегда не меньше суммы размеров её полей. Размер структуры зависит от порядка следования описания её полей

Пример

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

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

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

```
?
```

Пример

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

```
Ivan
```

Пример

```
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));
```

```
Sizeof s1 = ?
Sizeof s2 = ?
```

Пример

```c
#include <stdint.h>
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));
```

```
Sizeof s1 = 16
Sizeof s2 = 12
```

Задача

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:

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

Решение задачи про студентов (начало)

```c
#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;
}
```

```c
//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;
}
```

Решение задачи про студентов (продолжение)

```
//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;
}
```

Решение задачи про студентов (окончание)

```
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;
}
```

Пример

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

Решение задачи про датчик (начало)

```
#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;
}
```

Решение задачи про датчик (окончание)

```
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
        );
}
```

```
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;
}
```

Объединения

- Описание аналогично описанию структур, но используется зарезервированное слово union
- Все поля располагаются начиная с одного и того же адреса
- Изменение одного поля влечёт за собой изменение всех остальных полей. В каждый конкретный момент времени имеет смысл только одно поле, какое именно — решать вам
- Удобно использовать для доступа к отдельным частям значений

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

Пример

```
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
```

Пример

```
union intbytes {
    uint32_t number;
    float real;
    uint8_t bytes[4];
} d;


  d.real = 3.14;
    printf ("Real  %f\n",d.real);
    printf ("Number  %x\n",d.number);
    printf(" in memory is: %x %x %x %x\n", d.bytes[0], d.bytes[1],
d.bytes[2], d.bytes[3]);
```

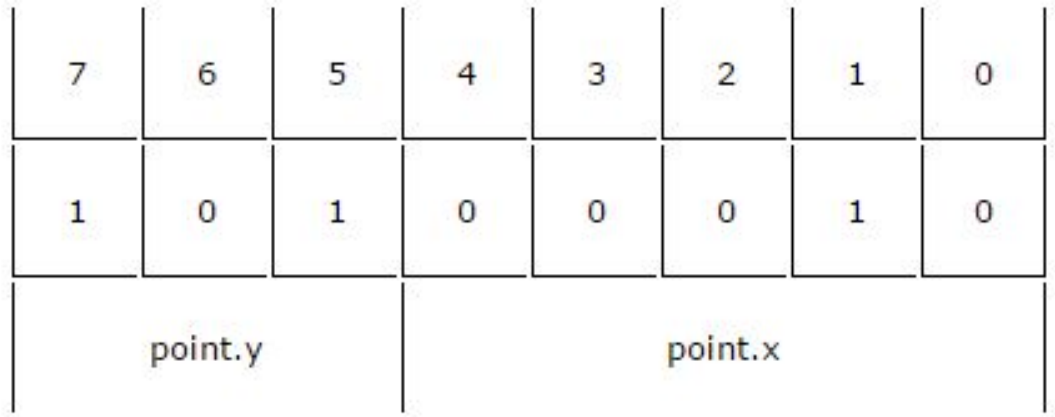
Битовые поля

- 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;
}
```

Пример

```c
#include <stdio.h>
struct point
{
    unsigned int x:5;    // 0-31
    unsigned int y:3;    // 0-7
};
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;
}
```



Практическое задание

Задачи на строки и файлы G1-G22

Для новичков достаточно 9-10 задач на выбор!
* - сделать все!

ВАШИ ВОПРОСЫ