










Раздел «Язык Си» . CoffeeFileStr :

- Чтение из файлов и выделение области памяти из "кучи".
- Строки
-  Задачи
 -  Задача 1
 -  Задача 2
 -  Задача 3
 -  Задача 4
 -  Задача 5
 -  Задача 6
 -  Задача 7
 -  Задача 8

Поиск

 Поиск

Раздел «Язык Си»

Главная
Зачем учить C?
Определения

Инструменты:

Поиск
Изменения
Index
Статистика

Разделы

Информация
Алгоритмы
Язык Си
Язык Ruby
Язык Ассемблера
EI Judge
Парадигмы
Образование
Сети
Objective C

Logon>>

Чтение из файлов и выделение области памяти из "кучи".

Большие объемы информации, которые может обрабатывать программа обычно хранятся в **файлах**.

Записи в файлах расположены **последовательно**. Значит, для того чтобы прочесть запись с номером n необходимо прочесть все предыдущие $n-1$ запись.

Для работы с файлами в языке C есть специальные функции и макросы.

Рассмотрим пример.

В файле находятся координаты точек (дробные числа): на каждую точку по два числа – координаты x и y . В первой строке файла указано количество координат – целое число.

```
4
2.5 3
0 0
0.3 0.4
2.5 1.6
```

Напишем программу, которая читает эти данные и ищет точку, которая ближе всего расположена к точке (0,0) Для некоторых задач все точки нужно хранить в оперативной памяти. Проще всего использовать массив.

Но до работы программы во-первых, неизвестно количество точек, а во-вторых, это количество может быть очень велико. Для хранения в памяти больших объемов используется расширенная память – **"куча"**.

Специальные функции определяют возможно ли выделить такой объем памяти. Если это возможно, то программа получит указатель на эту память, и с ней можно будет работать как с обычным массивом.

Но, как только память больше не нужна для работы программы, необходимо **ОСВОБОДИТЬ** ее специальными функциями.

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

// Описание структуры точки
typedef struct P{
    float x,y;
}Point;

// Функция печати всего массива точек
void printP(Point* p, int n){
    int i;
    for(i = 0; i<n; i++){
        printf("(%.2f,%.2f) ", p[i].x, p[i].y);
    }
};

int main(){
    int i;
    // Объявим указатель на точку (Point*)
    // Как только будет известно количество точек,
    // запросим место в "куче". Этот указатель получит
```

```

// адрес выделенного места
Point *massP = 0;

// Объявление файлового указателя(дескриптора файла)
// Файловый дескриптор связывается с конкретным файлом на диске,
// а затем используется для чтения записей из файла
FILE *fin;
// откроем файл "t.dat" для чтения ("r").
fin = fopen("t.dat","r");
// Файл не всегда может быть открыт (например его просто нет).
// Тогда специальная системная переменная errno будет не равна 0
// В этом случае прервем программу
if(errno){
    perror(":");
    exit(1);
}

int n; // переменная для количества точек
// Чтение из файла. Функция fscanf очень похожа на функцию scanf
// Единственное отличие, что для fscanf необходимо указать
// файловый дескриптор (fin)
fscanf(fin,"%d",&n);

// Функции для выделения памяти из "кучи"
// malloc выделяет память того размера, который мы укажем
// как параметр и возвращает адрес начала этой памяти.
// Если память не может быть выделена, то будет возвращен NULL
// malloc возвращает указатель типа *void, поэтому его необходимо
// преобразовать к нужному типу
// massP = (Point*)malloc(sizeof(Point)*n);

// Можно воспользоваться функцией calloc (память для массива)
// Она тоже выделяет память в куче, но в качестве параметра
// получает количество элементов заданного размера
// некоторые реализации при этом записывают 0
// во все байты этой выделенной памяти
massP = (Point*)calloc(n,sizeof(Point));

// Далее можно использовать эту память как обычный массив
for(i = 0; i < n; i++){
    // прочитаем в нее n элементов из файла
    // как только один элемент прочитан,
    // файловый указатель перемещается на следующий
    // и так до конца файла
    fscanf(fin,"%f%f",&(massP[i].x),&(massP[i].y));
}
// напечатаем наши точки на экране
// консоль и экран - тоже файлы
// ввод - stdin
// вывод stdout
printP(massP, n);
// Память больше не нужна,
// ОСВОБОЖДАЕМ ее
free(massP);
// файл нужно закрыть
fclose(fin);
return 0;
}

```

Однако не всегда известно количество записей в файле, и не всегда есть необходимость сохранять все их в память. Рассмотрим пример поиска минимально удаленной точки от (0,0)

```

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

typedef struct P{
    float x,y;
}Point;

void printP(Point p){
    int i;
    printf("(%.2f,%.2f) ", p.x, p.y);
};

// вычисление расстояния от точки(0,0) до точки a
float distFromZero(Point a){
    return sqrt(a.x*a.x+a.y*a.y);
}

```

```
};

int main(){
    int i;
    float min=0;
    Point mp;
    // описание дескриптора файла
    FILE *fin;
    // открытие файла
    fin = fopen("t.dat","r");
    // проверка открылся файл или нет
    if(errno){
        perror(":");
        exit(1);
    }
    int n = 0;
    // будем читать записи до конца файла
    // если достигнут конец файла, fscanf вернет значение
    // равное константе EOF
    while(fscanf(fin,"%f%f",&(massP.x),&(massP.y))!=EOF){
        // в этом случае сохранять BCE данные в массив нет необходимости
        n++;
        if (n < 2){
            min = distFromZero(massP);

        }
        else{
            float dst = distFromZero(massP);
            if(min >= dst)
                min = dst;
        }
    }
    printf("min = %0.2f\n",min);
    // Файл обязательно закрываем
    fclose(fin);
}
```

Строки

Строки в языке C – это, в сущности, массив символов. Для удобной работы с ними существует множество различных функций, описанных в заголовочном файле **string.h**.

Для корректной работы с этими функциями массив символов должен быть "правильной" строкой. Последний символ в массиве должен быть `\0`

Строки можно сравнивать, копировать, соединять и т.д. При этом необходимо всегда помнить, что строка с результатом копирования, соединения и др. всегда должна иметь достаточное количество памяти. Иначе результат работы программы будет непредсказуемый.

Рассмотрим сначала простой пример чтения двух строк и сравнения их.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <ctype.h>

int main(){
    int i;
    // два массива символов достаточной длины для
    // чтения
    char str[100], str2[100];
    int len; // размер строки
    // Чтать будем из файла
    FILE *fin;
    // открывем файл
    fin = fopen("str0.dat","r");
    // проверяем как открылся файл
    if(errno){
        perror(":");
        exit(1);
    }

    // читаем строку str (формат %s)
    // fscanf читает до разделителя: пробел, перенос строки, табуляция
    // fscanf при чтении сам добавляет \0 в конец массива
    // поэтому необходимо выделять память на один символ больше
    fscanf(fin,"%s",str);
```

```
// читаем строку str2 (формат %s)
fscanf(fin,"%s",str2);
// получим длину строки str
len = strlen(str);
// для сравнения используется функция strcmp
// Строки НЕЛЬЗЯ сравнивать оператором ==
// Так как строка - это массив символов, то
// оператором == будут сравниваться УКАЗАТЕЛИ на массивы
if(strcmp(str, str2) == 0)
    printf("одинаковые\n");
else
    if(strcmp(str,str2)>0)
        printf("первая больше \n");
    else
        printf("вторая больше \n");
// файл нужно закрыть
fclose(fin);
return 0;
}
```

Получив строку можно обращаться с ней как с обычным массивом. Рассмотрим пример, в котором печатаются только заглавные буквы из строки:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <ctype.h>

int main(){
    int i;
    char str[100];
    int len;
    FILE *fin;
    fin = fopen("str.dat","r");
    if(errno){
        perror(":");
        exit(1);
    }
    // прочитали строку из файла (сразу всю)
    fscanf(fin,"%s",str);
    // файл больше не нужен, закроем
    fclose(fin);
    // получили длину строки - количество символов в ней
    len = strlen(str);

    // перебираем все символы в поисках заглавных букв
    for(i=0; i< len; i++){
        // функция isupper() выясняет строчная буква или заглавная
        if(isupper(str[i])){
            printf("%c ",str[i]);
        }
    }

    return 0;
}
```

Рассмотрим еще пример. В файле записана длинная строка. Ее длина неизвестна, но она может оказаться больше буфера для ввода. В этой строке есть "мусор" в виде символов "#". Эти символы необходимо удалить. Исправленную строку запишем в другой файл.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <ctype.h>

// функция удаления #
void changed(char* st){
    // получим длину строки
    int len = strlen(st);
    int i=1;
    // создадим два указателя на char
    char *p; // отвечает за начало группы #
    char *pnext; // отвечает за конец группы #
    // указатель p получает адрес начала строки
    p = st;
```

```

// переберем всю строку в поисках #
while(i<len ){

    pnext = p;
    //если встретился # перемещаем указатель pnext
    // пока не встретится любой другой символ
    while(pnext[0] == '#'){
        pnext++;
        i++;
    }
    // если # был, то ВМЕСТО него ставим "конец строки"
    // для функции strcat
    // например: a123#ff -> a124'\0'ff
    // strcat будет "думать", что строка - a123, а следующая, на которую
    // указывает pnext: ff
    // Тогда он сможет соединить из в одну: a123ff
    if(pnext != p){
        // поставили "конец" вместо #
        p[0] = '\0';
        // соединяем
        strcat(st,pnext);
    }
    i++;
    p++;
}
// если в конце строки стоят #, обрежем ее
if( p[0] == '#'){
    p[0]='\0';
}

};

int main(){
    int i;
    char str[10]; // маленький буфер для чтения
    int len; // длина прочитанной строки
    int count = 0;
    FILE *fin; // файл для чтения
    FILE *fout; // файл для записи
    // открываем файл для чтения
    fin = fopen("str1.dat","r");
    if(errno){
        perror(":(");
        exit(1);
    }
    // открываем файл для чтения
    fout = fopen("str_out.dat","w");
    if(errno){
        perror(":(");
        exit(1);
    }

    // функция fgets позволяет указать максимальный размер
    // прочитываемой строки
    // будем читать по-частям.
    // fgets читает либо указанное количество символов, либо до переноса строки, если символов меньше
    // если не может прочитать , возвращает NULL
    while(fgets(str,9,fin)!=NULL){
        // получаем размер прочитанной строки
        len = strlen(str);
        // fgets не ставит конец строки в конец массива.
        // ставим сами ПОСЛЕ прочитанного текста
        // В этом случае сохранится "перенос строки"
        if(len < 9)
            str[len]='\0';
        else
            str[9]='\0';
        // вызов функции замены для прочитанной части строки
        changed(str);
        // печать в файл исправленной части
        fputs(str,fout);
    }
    // закрыли оба файла
    fclose(fin);
    fclose(fout);
    return 0;
}

```

Пример работы с "картинкой".

В файле содержится изображение. Белая клетка – это символ ".", а черная – "*". Известно, что ширина изображения не превышает 100 символов. На картинке нарисованы черные прямоугольники. Прямоугольники не касаются друг друга и не пересекаются.

Требуется подсчитать количество прямоугольников.

При решении заметим, что нам не известны размеры картинок, а только максимальная ширина. Поэтому мы не сможем выделить необходимое количество памяти для хранения всей картинки.

Рассмотрим решение, в котором достаточно каждый раз хранить только две строки из файла.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <math.h>
#include <ctype.h>

int main(){
    int i;
    char buf[100]; // буфер для чтения
    char *w,*w1; // будет выделена память для двух строк
    int len; // ширина картинки
    int count = 0; // количество прямоугольников
    int before=0; // была ли впереди *
    // файл для чтения
    FILE *fin;
    fin = fopen("rec.dat","r");
    if(errno){
        perror(":(");
        exit(1);
    }
    // прочитаем первую строку и узнаем ее размер
    fscanf(fin,"%s",buf);
    len = strlen(buf);
    // выделим память из "кучи" для двух строк
    w = (char*)calloc(len,sizeof(char));
    w1 = (char*)calloc(len,sizeof(char));
    // заполним точками строку w
    memset(w,'.',sizeof(char)*len);

    // будем читать в цикле пока не закончится файл
    do{
        // скопируем весь прочитанный буфер в строку w1
        memcpy(w1,buf,len*sizeof(char));

        for(i = 0; i < len; i++){
            // если встретился "верхний левый угол" прямоугольника
            // считаем его
            if(w1[i] == '*'){
                if(before == 0 && w[i] == '.' )
                {
                    count++;
                    before = 1;
                }
            }else{
                before = 0;
            }
        }
        // скопируем в w строку w1. Она стала "верхней"
        memcpy(w,w1,sizeof(char)*len);
        // напечатаем ее для наглядности
        printf("%s\n",w);
        // читаем из файла пока он не кончился
    }while(fscanf(fin,"%s",buf)!=EOF);
    // печатаем сколько прямоугольников
    printf("count=%d\n",count);
    fclose(fin);
}
```

Задачи

Задача 1

Дана строка с текстом, записанная в файл. Текст состоит из слов. Слова разделяются пробелами, точками, запятыми. Может быть несколько пробелов, точек и запятых вместе. Точки, пробелы и запятые словами не являются.

Написать программу, которая подсчитывает количество слов в тексте.

Задача 2

Дана список имен файлов. Некоторые файлы имеют "расширение" – в конце имени файла ставится точка, а за ней не более 3 букв. Например `.txt`. Каждое имя файла записано на отдельной строке.

Написать программу, которая выводит все имена файлов с заданным расширением.

Задача 3

Полное имя человека на русском языке выглядит так: **Имя Отчество Фамилия**. Например, *Андрей Павлович Макаров* или *Татьяна Павловна Варна*.

Дан список таких имен. Каждое имя записано на отдельной строке.

Написать программу, которая определяет пол каждого человека из списка и преобразует все имена к виду: **Фамилия И.О. пол (м/ж)** Все новые записи выводить в файл `name.out`.

Указание. Файл со списком имен сохранить в кодировке win1251. В программе использовать ТАКУЮ ЖЕ кодировку и пользоваться функцией поиска подстроки `strstr()`

Задача 4

В файле содержится изображение. Белая клетка – это символ ".", а черная – "*". Известно, что ширина изображения не превышает 100 символов. На картинке нарисованы черные прямоугольники. Прямоугольники не касаются друг друга и не пересекаются.

Требуется найти прямоугольник с самой большой площадью и вывести координаты его левого верхнего и правого нижнего угла.

Для хранения информации о картинке использовать не более 202 байт памяти (2 строки) и буфер (100 символов).

Задача 5

Целое число – это непрерывная последовательность цифр. Положительное число может иметь знак + перед ним или не иметь никакого знака. Перед отрицательным числом пишется знак -. В файл записана строка с цифрами и другими символами.

1. Написать программу, которая печатает первое число из строки.
2. Написать программу, которая подсчитывает сумму всех чисел в строке

Задача 6

Дробное число – это непрерывная последовательность цифр, в которой может встретиться одна точка. Дробное число всегда начинается хотя бы с одной цифры. Положительное число может иметь знак + перед ним или не иметь никакого знака. Перед отрицательным числом пишется знак -. В файл записана строка с цифрами и другими символами.

1. Написать программу, которая печатает первое число из строки.
2. Написать программу, которая подсчитывает сумму всех чисел в строке

Задача 7

Разные камни составляют цепочку. Цепочка из бусинок может считаться бусами, только если:

1. цепочка состоит из двух симметричных половинок (например: ABCBA, ABBA)
2. цепочка состоит повторяющихся периодических последовательностей камней (бусинок)

Написать программу, которая определяет является ли данная цепочка бусами.

Задача 8

В строке A(не более 255 символов) может встретиться подстрока B (размер B не более размера A).

Написать программу, которая заменяет все вхождения подстроки B на подстроку C (размер не более 255 символов).

Замена начинается слева.

Например, строка A: FFWWFFSSSSF, строка B: SS, строка C: ZZZ Результирующая строка:FFWWFFZZZZZF (`strstr`, `strcat`,`strcpy`);

-- TatyanaOvsyannikova2011 – 15 Nov 2016

(с) Материалы раздела "Язык Си" публикуются под лицензией [GNU Free Documentation License](#).