

Лекция №5

Файлы

Работа с файлами в языке C

Случайные числа

Генераторы случайных чисел

Использование в языке C

Списки данных

Указатели на структуры

Списки данных

Файлы

Файл (англ. file) — именованная область данных на носителе информации

Слово **file** впервые применено к компьютерному хранилищу в 1950 году

« ...результаты бесчисленных вычислений можно держать «в картотеке» (on file) и получать снова. Эта «картотека» теперь существует в запоминающей трубке, разработанной в лабораториях RCA. Она электрически сохраняет цифры, отправленные в вычислительную машину, и держит их в хранилище, заодно запоминая новые — ускоряя интеллектуальные решения в лабиринтах математики. »

В 1952 году слово **file** отнесли к **колоде перфокарт**. [2] Поначалу словом file называли **само устройство памяти**, а не его содержимое. Например, диски IBM 350, назывались **disk files**.

Системы наподобие Compatible Time-Sharing System ввели концепцию **файловой системы**, когда на одном запоминающем устройстве существует несколько виртуальных «устройств памяти», что и дало слову «файл» современное значение.

Имена файлов в CTTS состояли из двух частей, «основного имени» и «дополнительного имени» (последнее существует и поныне как расширение имени файла).

Файлы. Расширение.

Файл (англ. file) — именованная область данных на носителе информации

Расширение имени файла (часто расширение файла или расширение) как самостоятельный **атрибут** файла существует в файловых системах FAT16, FAT32, NTFS, используемых операционными системами MS-DOS, DR-DOS, PC DOS, MS Windows и используется для определения типа файла. Оно позволяет системе определить, **каким приложением следует открывать данный файл**.

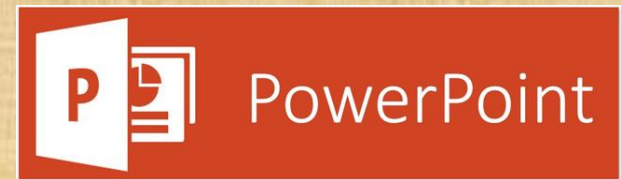
(По умолчанию в операционной системе Windows расширение скрыто от пользователя.)

В остальных файловых системах расширение — условность, часть имени, отделённая самой правой точкой в имени.

*.doc
*.docx



*.ppt
*.pptx



*.sln



Файлы в ОС Unix

Интуитивное определение файла звучит примерно так. Файл -- именованная область на жестком диске. С точки зрения ОС UNIX это совсем не так.

В ОС UNIX файл - очень удобная **абстракция**. С точки зрения UNIX файлом называется "**что-нибудь**", из чего можно **считывать** информацию или во что можно **записывать** информацию:

- Файлы в обычном смысле: файлы, которые хранятся на жестком диске (можно считывать из них и записывать в них информацию);
- Экран монитора: файл, в который можно выводить информацию (отобразится на экране монитора);
- Клавиатура: файл, из которого можно считывать информацию;
- Принтер: файл, в который можно выводить информацию (печать текста);
- Модем: файл, из которого можно считывать информацию и в который можно записывать информацию (обмен информации по сети);

Файл - это всё, что предназначено для ввода или вывода информации.

С этой точки зрения файлы бывают разными: принтер может только выводить информацию, а клавиатура -- только вводить. У такого рода файлов есть много особенностей. У файла на жестком диске есть понятие конца файла. Мы можем его считывать до тех пор, пока он не кончится. Тогда как у клавиатуры нет конца.

Работа с файлами в Си.

Самый распространенный способ работы с файлами в Си связан со структурой **FILE**. Эта структура определена в заголовочном файле стандартной библиотеки **<stdio.h>**. Размер этой структуры и ее поля зависят от ОС и от версии компилятора. Поэтому никто не пользуется структурой FILE. Обычно пользуются указателем на эту структуру: **FILE***. Например:

```
FILE *f = fopen("file1.txt", "r");
```

fopen - функция из стандартной библиотеки. Первый параметр - имя файла (в текущем каталоге). Второй параметр задает режим открытия файла; в данном случае "r" означает, что файл будет открыт только для чтения.

Эта функция возвращает ненулевой указатель, если открытие прошло успешно; и возвращает NULL, если произошла ошибка. Ошибка может возникать в следующих ситуациях:

- не существует файла;
- у программы недостаточно прав доступа для работы с файлом.

```
if (f == NULL) {  
    // файл не удалось открыть  
}  
else {  
    // Работа с файлом  
}
```

Работа с файлами в Си.

Что надо сделать после того, как мы поработали с файлом?

Формальный ответ: "Закрыть файл. - `fclose(f)`;

Ввиду механического устройства жесткого диска, данные в файл попадают не сразу.

Сначала данные записываются в так называемый буфер (область оперативной памяти), и когда он переполнится, то данные из буфера будут записаны в файл. Такая схема придумана для ускорения работы с файлами.

Буфер - это поле структуры `FILE`: указатель на массив `char`'ов.

Если мы напишем `fprintf(...)`, то запись произведется в буфер. И только тогда, когда буфер будет заполнен, он будет сразу весь записан на жесткий диск.

По этой причине, если мы не закроем файл функцией `fclose(f)`, то последние данные из буфера не запишутся в файл. Отсутствие этой команды может привести к потере данных в файле, который был открыт для записи (дозаписи).

Зачем закрывать файлы, открытые только на чтение?

Если не закрывать файлы (которые открыты даже для чтения), то это может привести к ограничению доступа к файлу для других программ.

Ограничения зависят от ОС

в Windows если файл открыт на чтение, то из другой программы его нельзя удалить
В любой ОС есть ограничение на количество одновременно открытых файлов. И это еще одна причина для закрытия файлов.

Работа с файлами в Си.

С точки зрения UNIX клавиатура и экран - это файлы.

Стандартные константы (заранее открытые файлы):

- FILE *stdin - стандартный файл (поток) ввода scanf(...) ~ fscanf(stdin, ...)
- FILE *stdout - стандартный файл (поток) вывода printf(...) ~ fprintf(stdout, ...)
- FILE *stderr

Метод отладки программы: подмена потоков:

```
//FILE *f = fopen(...);  
FILE *f = stdin;
```

При этом код программы будет содержать такие функции: fscanf(f, ...) или fprintf(f, ...).

stderr - это стандартный файл (поток) ошибок. По умолчанию выводит данные на экран.

- **небуферизованный файл (поток)**. Поэтому в этот файл (поток) все байты уходят без "задержки", которая могла бы возникнуть при буферизированном подходе.

```
fprintf(stderr, ...);
```

Текстовые и бинарные файлы.

fopen(f, "file1.txt", "w");

fopen("file1.txt", "wt") -- откроет файл как **текстовый** файл;

fopen("file1.txt", "wb") -- откроет файл как **бинарный** файл.

Символы переноса строк запишутся по разному

fprintf("Hello\n");

- | | | |
|------------------|----------|---------------|
| • "wb" в Windows | - 6 байт | - Hello\10 |
| • "wt" в Windows | - 7 байт | - Hello\10\13 |
| • "wt" в UNIX | - 6 байт | - Hello\10 |
| • "wb" в UNIX | - 6 байт | - Hello\10 |

При чтении файла:

- Параметр «rb» - символ \10 будет восприниматься как перевод строки
- Параметр «rt» - символ \10\13 будет восприниматься как перевод строки

Пример работы с файлами

```
void main()
{
    int a;
    FILE* f_src = fopen("src.txt", "rt");
    FILE* f_dst = fopen("dst.txt", "wt");

    if (f_src == NULL || f_dst == NULL)
    {
        printf ("Error!");
        return;
    }
    while (!feof(f_src))
    {
        fscanf(f_src, "%d", &a);
        a = a*a;
        fprintf(f_dst, "%d\n", a);
    }
    printf("Finish!\n");
    fclose(f_src);
    fclose(f_dst);
}
```

Пример работы с файлами

```
void main()
{
    int a;
    FILE* f_src = fopen("src.txt", "rt");
    FILE* f_dst = fopen("dst.txt", "wt");

    if (f_src == NULL || f_dst == NULL)
    {
        printf ("Error!");
        return;
    }

    while (!feof(f_src))
    {
        fscanf(f_src, "%d", &a);
        a = a*a;
        fprintf(f_dst, "%d\n", a);
    }

    printf("Finish!\n");
    fclose(f_src);
    fclose(f_dst);
}
```

- **Некорректная работа при символьных данных**
- **Отсутствие проверок корректности**
- **Лишний «перевод строки» ведет к некорректной работе**

Работа с файлами через строки

В языке C есть семейство функций ~ *atoi* (a - ASCII, i - integer)

N = atoi(string);

Надо заметить, что функция *atoi* **безопасная**, но **не очень удобная**:

- Безопасная в том смысле, что не сломается: `atoi("25a") == 25`
- "Неудобства" заключаются в том, то если мы передаем в качестве параметра строку, в которой есть не только числа, нужно быть очень внимательным и знать, как работает эта функция. Функция *atoi* никак не проинформирует нас, если преобразование прошло неудачно.
- `atoi("abc") == 0`

```
int N = atoi(string);  
long L = atol(string);  
long long LL = atoll(string);  
float F = atof(string);
```

```
void main()  
{  
  
    char temp[20];  
    int a;  
    scanf("%s", temp);  
    a = atoi(temp);  
    printf("%d", a);  
}
```


Работа с файлами через строки

```
void main()
{
    int a;
    char str[20];
    FILE* f_src = fopen("src.txt", "rt");
    FILE* f_dst = fopen("dst.txt", "wt");

    if (f_src == NULL || f_dst == NULL)
    {
        printf ("Error!");
        return;
    }
    while (!feof(f_src))
    {
        fscanf(f_src, "%s", str);
        a = atoi(str);
        a = a*a;
        fprintf(f_dst, "%d\n", a);
    }
    printf("Finish!\n");
    fclose(f_src);
    fclose(f_dst);
}
```

Полезные опции: fseek и ftell

int fseek(FILE *f, long offset, int flag); // Для файлов, которые открыты на чтение

FILE *f - файл, в котором передвигаемся;

long offset - количество байтов для отступа,
отступ производится в соответствии с 3-м параметром;

int flag - позиция, от которой будет совершен отступ;

в стандартной библиотеке C для этого параметра определены 3 константы:

- SEEK_SET -- начало файла;
- SEEK_CUR -- текущая позиция;
- SEEK_END -- конец файла;

int fseek() - возвращает ноль, если операция прошла успешно, иначе возвращается ненулевое значение.

long int ftell(FILE *f); // определение текущего положения в файле
// Для файлов, которые открыты на чтение

Работа с файлами в Си

```
FILE *f = fopen("file1.txt", "r");
```

```
fscanf(f, "%s", str);
```

```
fprintf(f, "%d\n", a);
```

```
int N = atoi(string);
```

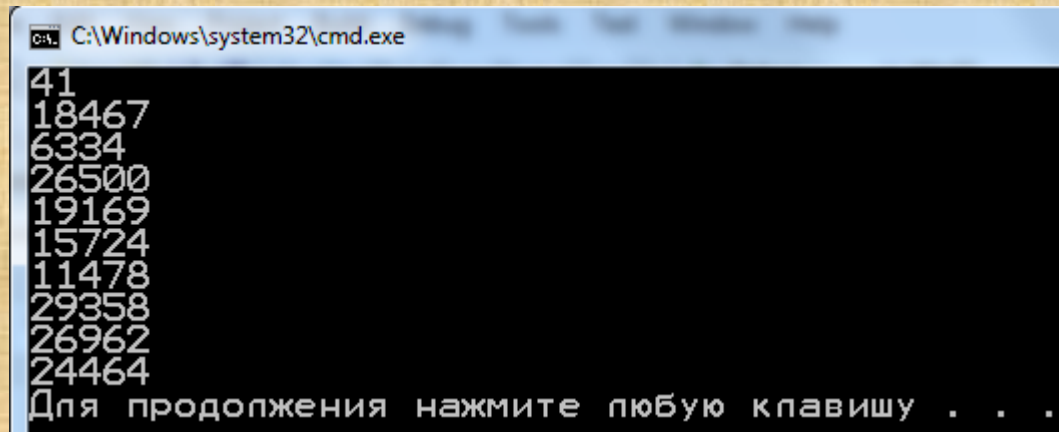
???

«Случайные» числа в Си

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

```
void main()
{
    int i, r;

    for (i = 0; i < 10; i++)
    {
        r = rand();
        printf("%d\n", r);
    }
}
```



```
C:\Windows\system32\cmd.exe
41
18467
6334
26500
19169
15724
11478
29358
26962
24464
Для продолжения нажмите любую клавишу . . .
```

```
#include <stdlib.h>
int rand( void );
```

Функция RAND() генерирует положительное целое число от 0 до RAND_MAX

Случайные числа



Использование надёжных источников энтропии, таких, как тепловой шум, дробовой шум, фотоэлектрический эффект, квантовые явления и т. д.

Псевдослучайные числа

«особенные алгоритмы»

Никакой детерминированный алгоритм не может генерировать полностью случайные числа, он может только аппроксимировать некоторые их свойства.

«Всякий, кто питает слабость к арифметическим методам получения случайных чисел, грешен вне всяких сомнений» - Джон фон Нейман

Если в качестве источника энтропии использовать текущее время, то для получения целого числа от 0 до N достаточно вычислить остаток от деления текущего времени в миллисекундах на число N+1. Недостатком этого ГСЧ является то, что в течение одной миллисекунды он выдает одно и то же число.

Инициализация генератора «случайных» чисел в Си

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

```
void srand( unsigned int seed );
```

Функция `srand` выполняет инициализацию генератора случайных чисел `rand`. Генератор псевдо-случайных чисел инициализируется с помощью аргумента `seed`.

```
#include <time.h>
```

```
Тип данных time_t
```

Этот тип данных используется для представления целого числа — количества секунд, прошедших после полуночи 00:00 , 1 января 1970 года в формате GMT. Это обусловлено историческими причинами, связанными со становлением платформы UNIX.

```
#include <time.h>
```

```
time_t time( time_t * timeptr );
```

```
time_t time( );
```

Функция возвращает текущее календарное значение времени в секундах. Если аргумент не является нулевым указателем, ей передается значение времени [типа time_t](#).

«Случайные» числа в Си

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

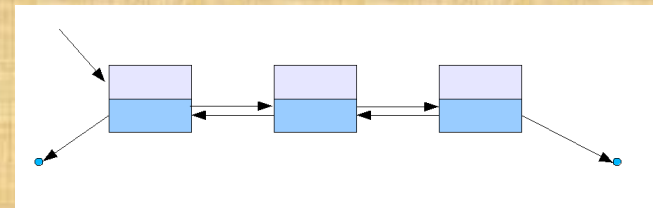
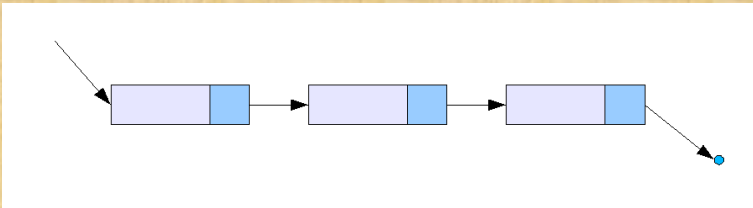
```
void main()
{
    int i, r;

    srand(time(NULL));
    for (i = 0; i < 10; i++)
    {
        r = rand();
        printf("%d\n", r);
    }
}
```

???

Связные списки данных

- **динамическая структура данных** в информатике
- состоит из **узлов**
- каждый узел содержит
 - собственно данные
 - одну или две **ссылки** («связки») на следующий и/или предыдущий узел списка

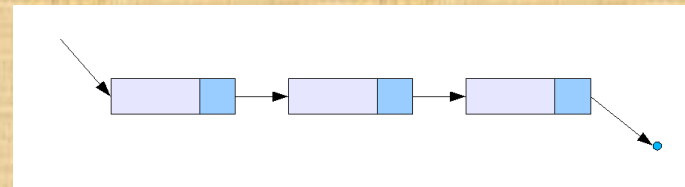


Принципиальные преимущества перед массивом:

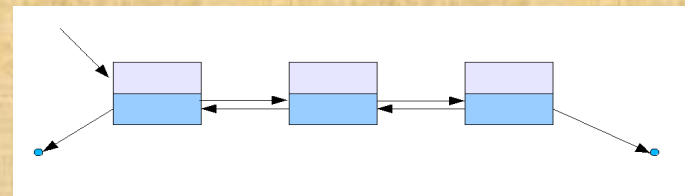
1. структурная гибкость: порядок элементов связного списка может не совпадать с порядком расположения элементов данных в памяти компьютера
2. порядок обхода списка всегда явно задаётся его внутренними связями.

Виды связанных списков

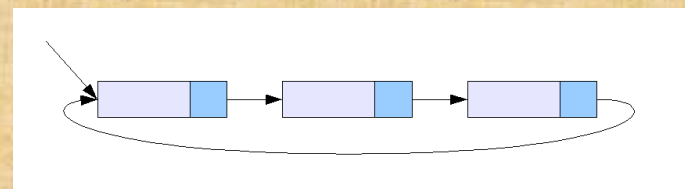
Линейный однонаправленный список



Двунаправленный связный список



Кольцевой связный список



Достоинства:

1. Эффективное (за константное время) добавление и удаление элементов
2. Размер ограничен только объёмом памяти компьютера и разрядностью указателей
3. динамическое добавление и удаление элементов

Недостатки:

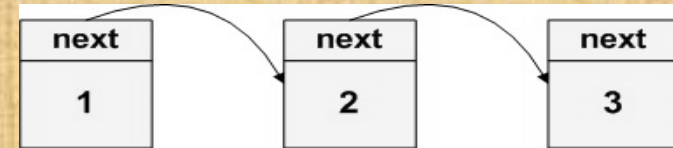
1. Сложность прямого доступа к элементу-определение физического адреса по его индексу
2. Ряд операций со списками медленнее, чем с массивами (произвольный доступ)
3. Соседние элементы списка могут быть распределены в памяти нелокально, что снизит эффективность кэширования данных в процессоре
4. Накладные расходы на перебор элементов снижают эффективность распараллеливания

Работа со списками в Си

Список — это упорядоченная последовательность связанных данных, связанных между собой.

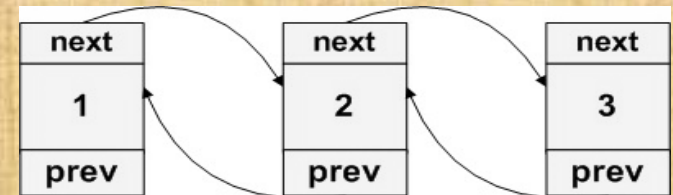
Односвязный список характеризуется наличием одной связи у соседних элементов. Другими словами, каждый элемент знает об одном соседе

- передвигаться можно только последовательно в одном направлении: от начала к концу

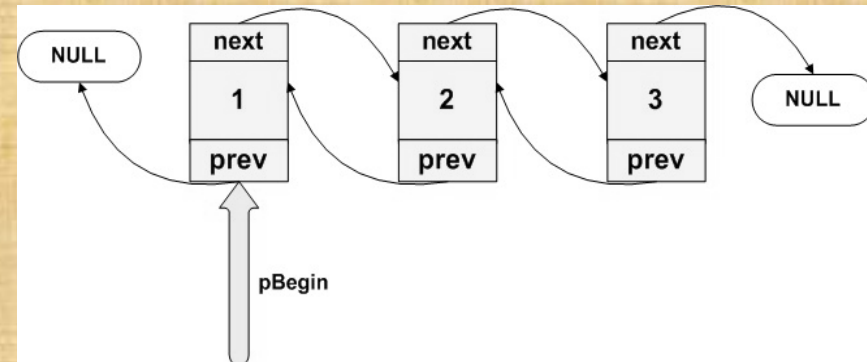


Двусвязный список отличается наличием двух связей у каждого элемента. При этом организуется двунаправленность списка.

- зная любой элемент списка, можно получить информацию как о следующем элементе списка, так и о предыдущем



Для управления списком необходимо знать только первый его элемент — начало списка. Последний элемент определяется по признаку того, что следующего за ним не существует, т. е. указатель равен NULL



Работа со односвязным списком в Си

```
struct Cell
```

```
{
```

```
    int Val;
```

```
    struct Cell* pnext;
```

```
};
```

Каждый элемент списка содержит два поля:

- собственно целочисленное число
- указатель на следующий элемент списка:

```
void main()
```

```
{
```

```
    struct Cell* pBegin = NULL;
```

```
    pBegin = CreateList();
```

```
    PrintList(pBegin);
```

```
    pBegin = ClearList(pBegin);
```

```
}
```

Работа со односвязным списком в Си

```
struct Cell* CreateList()
{
    struct Cell* pBegin;
    struct Cell* pNew1;          struct Cell* pNew2;          struct Cell* pNew3;

    pNew1 = malloc(sizeof(struct Cell));
    scanf("%d", &(pNew1->Val));

    pNew2 = malloc(sizeof(struct Cell));
    scanf("%d", &(pNew2->Val));

    pNew3 = malloc(sizeof(struct Cell));
    scanf("%d", &(pNew3->Val));

    pBegin = pNew1;
    pNew1->pnext = pNew2;
    pNew2->pnext = pNew3;
    pNew3->pnext = NULL;

    return pBegin;
}
```

Работа со односвязным списком в Си

```
void PrintList(struct Cell* pBegin)  
{  
    struct Cell* pCur = NULL;  
    if (pBegin == NULL)  
    {  
        printf("No elements\n");  
        return;  
    }  
    pCur = pBegin;  
    do  
    {  
        printf("%d\n", pCur->Val);  
    }  
    while(pCur != NULL);  
}
```


Работа со односвязным списком в Си

```
struct Cell* ClearList(struct Cell* pBegin)  
{  
    struct Cell* pCur = NULL;  
    if (pBegin == NULL)  
        return NULL;  
  
    do  
    {  
        pCur = pBegin;  
        pBegin = pBegin->pnext;  
        free(pCur);  
  
    }  
    while(pBegin != NULL);  
  
    return NULL;  
  
}
```

Деревья

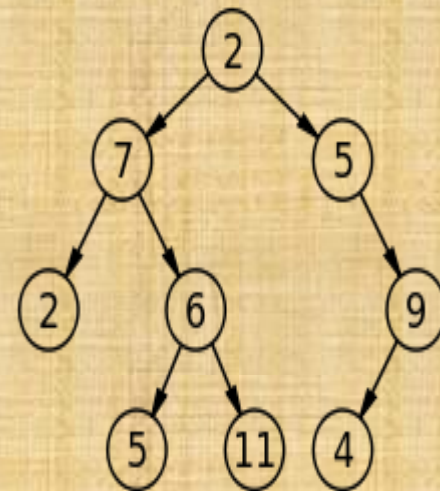
Дерево — это связный ациклический граф.

- **Связность** - наличие путей между любой парой вершин
- **Ациклическость**
 - отсутствие циклов
 - между парами вершин имеется только по одному пути

Лес — упорядоченное множество упорядоченных деревьев.

Ориентированное (направленное) дерево — ациклический орграф (ориентированный граф, не содержащий циклов), в котором только одна вершина имеет нулевую степень захода (в неё не ведут дуги), а все остальные вершины имеют степень захода 1 (в них ведёт ровно по одной дуге).

Вершина с нулевой степенью захода называется **корнем дерева**, вершины с нулевой степенью исхода (из которых не исходит ни одна дуга) называются концевыми вершинами или **листьями**



Стек

Стек (англ. stack) — абстрактный тип данных, представляющий собой список элементов, организованных по принципу **LIFO** (англ. last in — first out, «последним пришёл — первым вышел»).

Принцип работы стека – «стопка тарелок»: чтобы взять вторую сверху, нужно снять верхнюю.

Понятие стека ввел в 1946 Алан **Тьюринг**

В 1957 году немцы Клаус Самельсон и Фридрих Л. Бауэр запатентовали идею Тьюринга.

Зачастую стек реализуется в виде **однонаправленного списка**.

При организации стека в виде однонаправленного списка значением переменной стека является указатель на его вершину — адрес вершины.

Если стек пуст, то значение указателя равно NULL.

