Лекция 16. Хранение, связывание и управление памятью. Часть 3.

Ревун Артем Леонидович
ст. преп. кафедры вычислительных систем

Курс «Программирование», весенний семестр, 2024
Сибирский государственный университет телекоммуникаций и информатики (г. Новосибирск)

Курс: Администрирование ОС «АЛЬТ»

https://eios.sibsutis.ru/course/view.php?id=4007

Выполнено уже 56% курса, всего 29 мест!

Кодовое слово: Basealt-2025

Содержание

- Ключевые слова: ***auto, extern, static, register, const, volatile, restricted, _Thread_local, _Atomic***
- Функции: ***rand(), srand( ), time( ), malloc( ), calloc(), free()***
- Определение в языке С области видимости переменной и времени жизни переменной
- Проектирование более сложных программ

Выделенная память (1)

Память выделяется: статическая – во время загрузки программы, автоматическая – когда поток управления программы входит в блок.

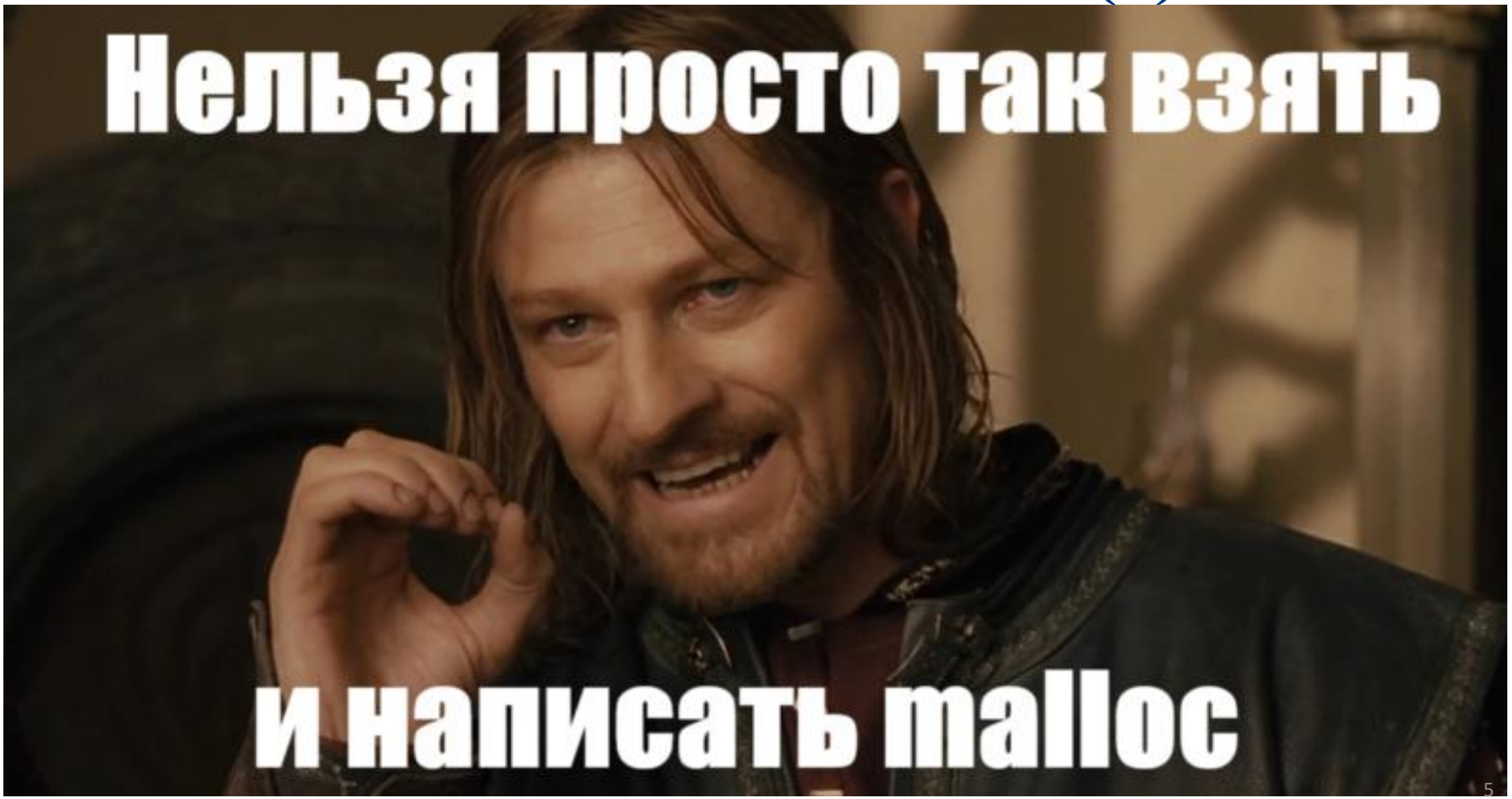
Язык Си позволяет работать с выделенной памятью.

```
void* malloc (size_t size);
```

`size_t size` – значение в байтах, выделяемого блока памяти.

Возвращаемое значение, указатель на начало блока.

Выделенная память (2)



Выделенная память (3)

Каждый вызов функции `malloc()` должен сопровождаться вызовом процедуры `free()`.

```
void free (void* ptr);
```

В качестве единственного аргумента принимает адрес, возвращенный ранее функцией `malloc()`.

Вызов процедуры `free()` освобождает память, которая была выделена. Таким образом продолжительность хранения выделяемой памяти, зависит от комбинации `malloc()…free()`.

Выделенная память (4)

Функция `calloc()`, предлагает ещё один способ выделения памяти.

```
void* calloc (size_t num, size_t size);
```

где *num* – кол-во элементов, а *size* размер элемента в байтах. Особенность `calloc()` заключается в том, что блок инициализирует все биты нулями.

```
int * pData, i;  scanf ("%d",&i);
pData = (int*) calloc (i, sizeof(int));
if (pData==NULL) exit (EXIT_FAILURE);
```

Самостоятельно изучить

Раздел: Функция генерации случайных чисел и статическая переменная 502 страница, глава 12

Изучили? Проверяем!

Генераторы псевдослучайных чисел (1)

```
static unsigned long int next = 1;
int randl(void){
    next = next * 1103515245 + 12345;
    return (unsigned int)(next / 65536) %
                                32768;
}
void srandl(unsigned int seed){
    next = seed;
}
```

Генераторы псевдослучайных чисел (2)

```
static unsigned long int next = 1;
next = next * 1103515245 + 12345;
return (unsigned int)(next / 65536) % 32768;
//при next = 1
1 16838 19304 1591 10185 10992 13777 11572
//при next = 2
2 1018 28604 2731 2950 185 12576 4068
//при next = 3
3 17857 32088 13622 14275 3943 32386 2105
```

Генераторы псевдослучайных чисел (2)

```
#include <time.h> /*прототип ANSI time()*/
static unsigned long int next = 1;
void srandl(unsigned int seed){
    next = seed;
}
srandl((unsigned int) time(0));
time_t time (time_t* timer);
timer – указатель на хранящееся время
time – возвращает текущее календарное время
```

Генераторы псевдослучайных чисел (3)

Time of Unix - возвращаемое значение обычно представляет собой количество секунд, прошедших с 00:00 часов 1 января 1970 года по Гринвичу (т.е. текущую временную метку с начала эпохи Unix). Такие временные значения, часто называются timestamp.

Сейчас timestamp x > 1739947500 (13:45)

Размер long ing 4294967295

Размер long long 18446744073709551615

Mon Jul 22 2554 06:34:33 GMT+0700 (Новосибирск, стандартное время)

Квалификаторы типов в Си (1)

Квалификаторы в Си, позволяют установить:

- Тип данных (`int, float, unsigned, etc…`)
- Класс хранения (`auto, static, register, etc…`)
- Постоянство хранения
- Изменчивость данных

Квалификаторы типов в Си (2)

Рассмотрим работу квалификаторов в Си:

const

volatile

restrict

_Atomic

typedef

Квалификатор const (1)

Квалификатор `const` делает переменную неизменяемой (только для чтения). После того, как она инициализирована.

```
const int days[12] = {31, 28, 31, 30,
        31, 30, 31, 31, 30, 31, 30, 31};
```

Квалификатор const (2)

При использовании квалификатора `const` с указателями следует разделять:

- Объявляете ли вы указатель как `const`
- Влияете ли вы на возможность изменения данных на которые указатель ссылается

Квалификатор const (3)

Указатель *pf ссылается на объект данных типа float, который не может быть изменён. Изменить сам указатель можно, таким образом он может ссылаться и на другое значение типа const.

```
float cfa[5] = {1., 2., 3., 4., 5.};
const float *pf = cfa;
printf("%f, %f, %f", *pf, *(pf+1), *(pf+2));
1.000000, 2.000000, 3.000000
```

Квалификатор const (4)

```
float cfa[5] = {1., 2., 3., 4., 5.};
const float *pf = cfa;
printf("%f, %f, %f", *pf, *(pf+1), *(pf+2));
*pf = 7.;
error: assignment of read-only location '*pf'
```

Квалификатор const (4)

```
float cfa[5] = {1., 2., 3., 4., 5.};
const float *pf = cfa;
pf = cfa+1;
printf("%f, %f, %f", *pf, *(pf+1), *(pf+2));
2.000000, 3.000000, 4.000000
//тоже самое
float const *pf = cfa;
```

Квалификатор const (5)

Указатель *pt ссылается на const, что говорит о отсутствии возможности модифицировать его значение.

Всегда будет ссылаться только на уже созданный объект

```
float cfa[5] = {1., 2., 3., 4., 5.};
float * const pt = cfa;
printf("%f, %f, %f", *pt, *(pt+1), *(pt+2));
1.000000, 2.000000, 3.000000
```

Квалификатор const (6)

```
float cfa[5] = {1., 2., 3., 4., 5.};
float  * const pt = cfa;
pt=cfa+2;
printf("%f, %f, %f", *pt, *(pt+1), *(pt+2));
error: assignment of read-only variable 'pt'
```

Квалификатор const (7)

Указатель *ptr ссылается на const и объект данных типа float, что говорит о отсутствии возможности модифицировать как его значение, так и адрес на который он указывает.

```
float cfa[5] = {1., 2., 3., 4., 5.};
const float * const ptr = cfa;
```

Квалификатор const (8)

```
//Защита значений в функциях
void display(const int array[], int limit);
//Защита глобальных переменных  //constant.h
const double PI = 3.14159;
const char * MONTHS[12] = {"Январь", "Февраль", "Март",
"Апрель ", "Май", "Июнь", "Июль", "Август", "Сентябрь",
                           "Октябрь", "Ноябрь", "Декабрь"};
//main.c
extern const double PI;
extern const * MONTHS [];
```

Квалификатор volatile (1)

Квалификатор `volatile` сообщает компилятору, что переменная может иметь значение, которое изменяется действиями, внешними по отношению к программе.

```
volatile unsigned long long *loc_time;
```

Квалификатор volatile (2)

Указатель `loc_time` е передаётся в другую программу, например в программу которая в реальном времени изменяет время в секундах записывая значения по адресу `loc_time` из вне.

```
volatile unsigned long long *loc_time;
```

Квалификатор volatile (3)

В чём тогда смысл? До появления `volatile` компилятор не мог однозначно принимать решения при выполнение оптимизации кода. `Volatile` явно указывает компилятору на то, что значение будет изменяться в течении времени вне программы, таким образом такие параметры не будут затронуты при оптимизации.

Квалификатор volatile (4)

Оптимизируя такой код компилятор может сам поместить значение x в регистровую память процессора, такая операция называется кэшированием.

```
int main(void){
    volatile int volatint = 10;
    int val1 = x;
    /* код, в котором x не используется */
    int val2 = x;
}
```

Квалификатор volatile (5)

Для автоматической оптимизации кода при компиляции следует указать флаги:

-O0. По умолчанию, все оптимизации отключены, код генерируется быстро, что полезно для отладки и разработки.

-O1. Включает базовые оптимизации, такие как устранение общих подвыражений и упрощение потока управления. Этот уровень подходит для большинства разработки и тестирования.

Квалификатор volatile (6)

-O2. Включает более агрессивные оптимизации, такие как встраивание функций, оптимизация циклов и другие. Предполагает улучшение производительности во время выполнения, но может увеличить размер кода.

-O3. Включает ещё более агрессивные оптимизации, включая векторизацию, встраивание функций и раскрутку циклов. Может значительно улучшить производительность, но может привести к увеличению исполняемых файлов.

Квалификатор volatile (6)

-Os. Оптимизирует размер кода, а не скорость выполнения. Пытается уменьшить размер результирующего исполняемого файла, даже если это означает жертву некоторой производительностью во время выполнения.

-Og. Оптимизирует для отладки.

Квалификатор restrict (1)

По аналогии с `volatile` указывает компилятору на возможность оптимизировать определенные разновидности кода. Может быть применён только к указателям, которые представляют единственное первичное средство доступа к объекту данных.

```
int arr[10];
int * restrict restarr = (int *)
                 malloc(10 * sizeof(int));
int * par = arr;
```

Квалификатор restrict (2)

```
for (int n = 0; n < 10; n++){
    par[n] += 5;
    restarr[n] += 5;
    arr[n] *= 2;
    par[n] += 3;
    restarr[n] + = 3;
}
```

Квалификатор restrict (3)

```
for (int n = 0; n < 10; n++){
    par[n] += 5;
    arr[n] *= 2;
    par[n] += 3;
    restarr[n] += 8; //оптимизация 5+3
}
```

Квалификатор restrict (4)

```
for (int n = 0; n < 10; n++){
    par[n] += 5;
    arr[n] *= 2;
    par[n] += 3;//3+5 не будет работать
    restarr[n] += 8; //оптимизация 5+3
}
```

Квалификатор restrict (5)

Если квалификатор `restrict` не указан, компилятор рассчитывает на самый плохой случай. А именно на случай что данные могут быть изменены между двумя применениями указателя. Применение `restrict` укажет компилятору на свободу при поиске оптимизации.

Квалификатор restrict (6)

```
void * memcpy(void * restrict s1,
              const void * restrict s2, size_t n);
```

Копирует *n* байт из указателя *s2* в *s1*; Типы указателей не имеют значение, функция “просто” копирует байты.

```
void * memmove (void * s1, const void * s2,
                                      size_t n);
```

Копирует *n* байт из указателя *s2* в *s1*; Типы указателей не имеют значение, функция “просто” копирует байты.

В чём же отличие?)

Квалификатор _Atomic (1)

```
int hogs; // обычное объявление
hogs = 12; // обычное присваивание

_Atomic int hogs; // hogs - атомарная переменная
atomic_store(&hogs, 12); // макрос из stdatomic.h
```

Продолжение следует…
Спасибо за внимание

Ревун Артем Леонидович
ст. преп. кафедры вычислительных систем

Курс «Программирование», весенний семестр, 2024
Сибирский государственный университет телекоммуникаций и информатики (г. Новосибирск)