

Структурное программирование

ЛЕКЦИЯ №4

26 ОКТЯБРЯ 2023



Статическая память

Примеры:

- Глобальные переменные:

```
int mass[100];
```

- Строки (неизменяемые):

```
"Hello world!"
```

- Переменные явно указанные статическими:

```
static int tmp;
```



Свойства:

- Выделяется во время старта программы
- Освобождается во время завершения программы
- Инициализируется нулевыми значениями
- Фиксированный размер

Статическая память

```
void f () {  
    static int tmp = 4;  
    printf("%d\n", tmp++);  
}  
  
int main() {  
    for (int i = 0; i < 10; i++) {  
        f();  
    }  
}
```



Статическая память

```
void f () {  
    static int tmp = 4;  
    printf("%d\n", tmp++);  
}  
  
int main() {  
    for (int i = 0; i < 10; i++) {  
        f();  
    }  
}
```

```
4  
5  
6  
7  
8  
9  
10  
11  
12  
13
```



Автоматическая память

Примеры:

- Локальные переменные {...; int tmp; ...}
- Аргументы функций (...; double value, ...)

Свойства:

- Выделяется при входе в блок
- Освобождается при выходе из блока
- Изначально не инициализирована
- Фиксированный размер
- Размещается на стеке



Динамическая память

- Выделяется и освобождается когда захотим попросим.
- Изначально не инициализирована.
- Размер задается динамически.
- Хранится в куче.



Работа с динамической памятью

Для работы с динамической памятью используется библиотека `stdlib.h`

➤ `malloc`

➤ `calloc`

➤ `realloc`

➤ `free`

Выделение памяти

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

Функция выделяет блок памяти размером `size` байт и возвращает указатель на начало блока.

Если не получилось – `NULL`

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

Функция выделяет блок памяти размером `num*size` байт, зануляет его и возвращает указатель на начало.

Если не получилось – `NULL`.

Освобождение блока памяти

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

Функция освобождает блок памяти, если `ptr == NULL`, ничего не делает.

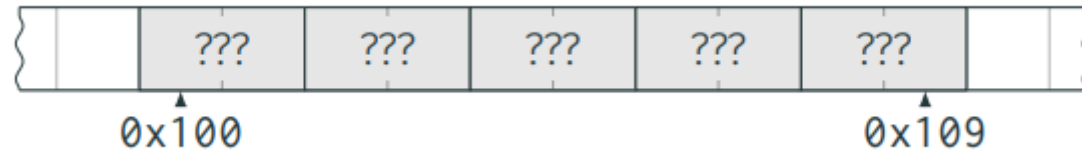
После вызова значение указателя `ptr` остается прежним, но разыменовывать его нельзя. Повторно освободить тоже нельзя!

Пример: массив динамического размера



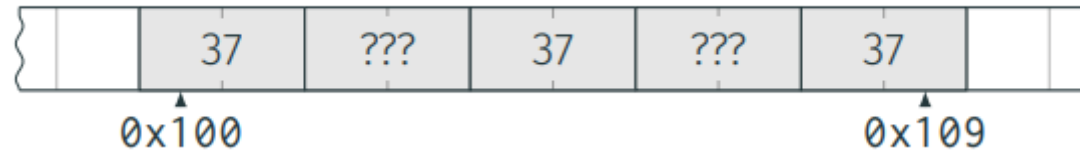
```
int n = 5;  
short* p;
```

Пример: массив динамического размера



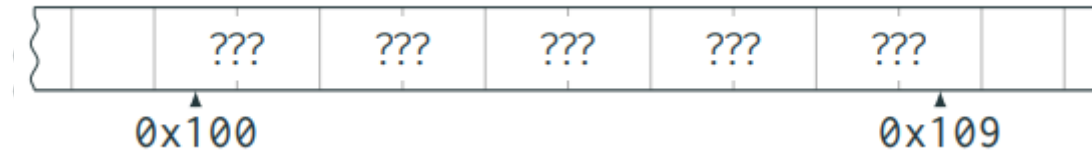
```
int n = 5;  
short* p;  
p = malloc(n * sizeof(short));  
if (p == NULL) { return 0; }
```

Пример: массив динамического размера



```
int n = 5;
short* p;
p = malloc(n * sizeof(short));
if (p == NULL) { return 0; }
p[0] = p[n / 2] = p[n - 1] = 37;
```

Пример: массив динамического размера



```
int n = 5;
short* p;
p = malloc(n * sizeof(short));
if (p == NULL) { return 0; }
p[0] = p[n / 2] = p[n - 1] = 37;
free(p);
```

Утечки памяти

Если память регулярно выделяется, но не освобождается, то рано или поздно память кончится, «утечет».

Главное правило хорошего программиста на С: закончил работать с блоком памяти – освободи его!





ЛОВИМ утечки в VS

```
#define _CRTDBG_MAP_ALLOC
#include <stdlib.h>
#include <crtDBG.h>

int main()
{
    int a = 5;
    int* p;
    for (int i = 0; i < 5; i++)
    {
        p = malloc(5);
    }
    _CrtDumpMemoryLeaks();
}
```

```
Detected memory leaks!
Dumping objects ->
C:\Users\vmbror\source\repos\Project4\Project4\Source.c(11) : {80} normal block at 0x009889B0, 5 bytes long.
Data: <    > CD CD CD CD CD
C:\Users\vmbror\source\repos\Project4\Project4\Source.c(11) : {79} normal block at 0x00984B90, 5 bytes long.
Data: <    > CD CD CD CD CD
C:\Users\vmbror\source\repos\Project4\Project4\Source.c(11) : {78} normal block at 0x00984B58, 5 bytes long.
Data: <    > CD CD CD CD CD
C:\Users\vmbror\source\repos\Project4\Project4\Source.c(11) : {77} normal block at 0x00985030, 5 bytes long.
Data: <    > CD CD CD CD CD
C:\Users\vmbror\source\repos\Project4\Project4\Source.c(11) : {76} normal block at 0x00984FF8, 5 bytes long.
Data: <    > CD CD CD CD CD
Object dump complete.
"Project4.exe" (Win32). Загружено "C:\Windows\SysWOW64\kernel.appcore.dll".
```

Список ошибок [Вывод](#) Результаты поиска символа

Фрагментация памяти



Все хорошо, все свободно

Фрагментация памяти



Все хорошо, все занято

Фрагментация памяти



Высокая фрагментация: 5 байт из 11 свободно, но выделить блок размером 2 байта не получится.

Изменение размера блока памяти

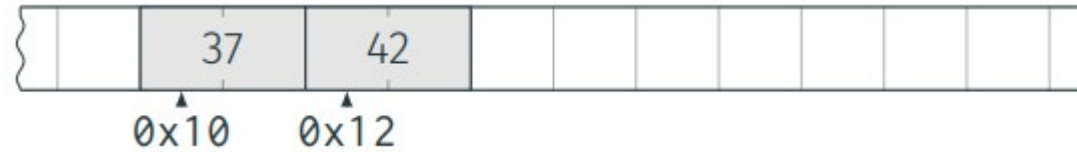
```
void* realloc(void* prt, size_t size);
```

Функция изменяет размер блока памяти до size байт. В случае успешного изменения размера возвращает указатель на начало блока, иначе NULL.

Функция может как уменьшать размер, так и увеличивать. Возможно перемещение содержимого памяти, при этом возвращается указатель на новое месторасположение.

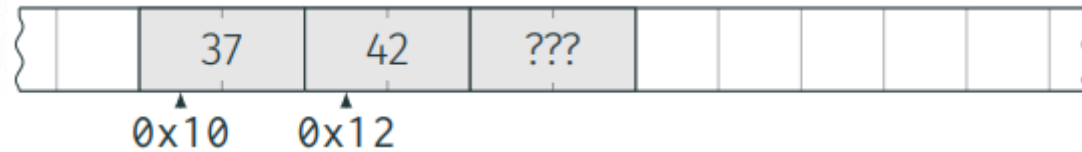
В случае неуспешного изменения размера, изначальный блок памяти не освобождается.

Пример работы realloc



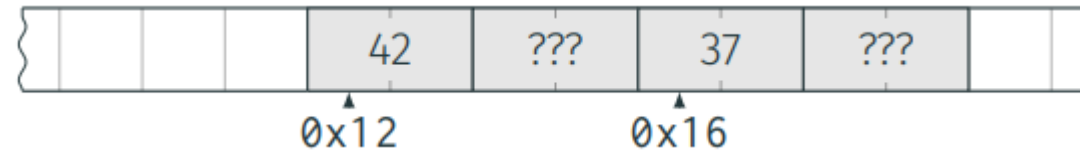
```
short* x = malloc(sizeof(short));  
short* y = malloc(sizeof(short));  
if (x == NULL || y == NULL) { return 0; }  
*x = 37;  
*y = 42;
```

Пример работы realloc



```
short* x = malloc(sizeof(short));  
short* y = malloc(sizeof(short));  
if (x == NULL || y == NULL) { return 0; }  
*x = 37;  
*y = 42;  
short* y2 = realloc(y, 2 * sizeof(short));
```

Пример работы realloc



```
short* x = malloc(sizeof(short));
short* y = malloc(sizeof(short));
if (x == NULL || y == NULL) { return 0; }
*x = 37;
*y = 42;
short* y2 = realloc(y, 2 * sizeof(short));
short* x2 = realloc(x, 2 * sizeof(short));
if (x2 == NULL || y2 == NULL) { return 0; }
```

Изменяем размер правильно

```
char* x;  
x = malloc(2000);  
if (x == NULL) { return 0; }  
use(x);  
  
x = realloc(x, 4000);  
if(x==NULL){/* а больше нет нашего x */ }
```

Изменяем размер правильно

```
char* x;  
x = malloc(2000);  
if (x == NULL) { return 0; }  
use(x);
```

```
char* x2 = realloc(x, 4000);  
if (x2 == NULL) { free(x); return 0; }  
x = x2;  
use(x);
```



Используем указатели

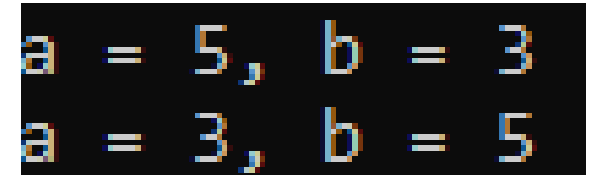
Как поменять местами два любых куска памяти? (2 любых переменных?
Одного типа)

```
a = 5, b = 3  
a = 3, b = 5
```

Используем указатели

```
void swap(void* a, void* b, size_t size) {
    char* tmp;
    tmp = (char*)malloc(size);
    memcpy(tmp, a, size);
    memcpy(a, b, size);
    memcpy(b, tmp, size);
    free(tmp);
}

int main()
{
    int a = 5, b = 3;
    printf("a = %d, b = %d\n", a, b);
    swap(&a, &b, sizeof(int));
    printf("a = %d, b = %d\n", a, b);
}
```

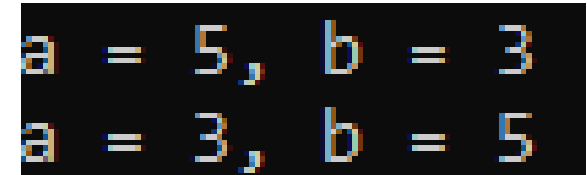


```
a = 5, b = 3
a = 3, b = 5
```

Используем указатели

```
void swap(void* a, void* b, size_t size) {
    char* tmp;
    tmp = (char*)malloc(size);
    memcpy(tmp, a, size);
    memcpy(a, b, size);
    memcpy(b, tmp, size);
    free(tmp);
}

int main()
{
    int a = 5, b = 3;
    printf("a = %d, b = %d\n", a, b);
    swap(&a, &b, sizeof(a));
    printf("a = %d, b = %d\n", a, b);
}
```



```
a = 5, b = 3
a = 3, b = 5
```

Используем указатели

```
void swap(void* a, void* b, size_t size) {
    char tmp;
    size_t i;
    for (i = 0; i < size; i++) {
        tmp = *((char*)b + i);
        *((char*)b + i) = *((char*)a + i);
        *((char*)a + i) = tmp;
    }
}

int main()
{
    int a = 5, b = 3;
    printf("a = %d, b = %d\n", a, b);
    swap(&a, &b, sizeof(a));
    printf("a = %d, b = %d\n", a, b);
}
```

```
a = 5, b = 3
a = 3, b = 5
```





Функции как данные

В языке C с функциями можно работать как с данными:

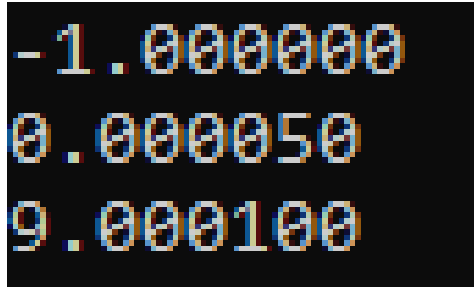
```
double (*my_func)(double, double) = pow;  
double x = my_func(3, 2);
```

Функция – набор байтов в памяти, кодирующих тело этой функции с помощью машинных команд. Значит, можно просто взять адрес этого «набора байтов»

Более формально:

```
double (*my_func)(double, double) = &pow;  
double x = (*my_func)(3, 2);
```

Пример



```
-1.000000  
0.000050  
9.000100
```

```
double diff(double x, double (*f)(double))  
{  
    return (f(x + 0.0001) - f(x)) / 0.0001;  
}  
  
double my_func(double x)  
{  
    return x * x + 5*x;  
}  
  
int main()  
{  
    printf("%lf\n", diff(M_PI, sin));  
    printf("%lf\n", diff(M_PI, cos));  
    printf("%lf\n", diff(2, my_func));  
}
```

Пример работы с указателем на функцию

```
int checkint(int* a, int* b)
{
    if (*a > *b)
        return 1;
    return 0;
}
```

```
int checkdouble(double* a, double* b)
{
    if (*a > *b)
        return 1;
    return 0;
}
```

Пример работы с указателем на функцию

```
void printok(void* a, void* b, int (*check)(void*, void*))
{
    if (check(a, b))
        printf("ok\n");
    else
        printf("ne ok\n");
}

int main()
{
    int a = 4, b = 9;
    double c = 4.4, d = 2.9;
    printok(&a, &b, checkint);
    printok(&c, &d, checkdouble);
    return 0;
}
```



Составные типы данных: структуры

Структура – объединение нескольких объектов (могут быть разного типа), под одним именем. (сложное из простого)

Объектами могут быть переменные, массивы, указатели и другие структуры.



Структуры: синтаксис

```
struct Название {  
    тип_поля название_поля;  
    тип_поля название_поля;  
};
```

```
int main()  
{  
    struct Название название_экземпляра = { значение, значение };  
    название_экземпляра.название_поля=...;  
}
```

Структуры: синтаксис

Инициализация полей структуры при ее объявлении:

```
struct Vector {  
    double x, y, z;  
};  
  
int main()  
{  
    struct Vector v = { 3,0,4 };  
    double len = sqrt(v.x * v.x + v.y * v.y + v.z * v.z);  
    return 0;  
}
```

Структуры: синтаксис

Инициализация полей структуры после ее объявления:

```
struct Vector {  
    double x, y, z;  
};  
  
int main()  
{  
    struct Vector v;  
    v.x=3;  
    v.y=0;  
    v.z=4;  
    return 0;  
}
```

Структуры: удобный синтаксис

`typedef СтарыйТип НовыйТип;` - объявление типа НовыйТип как синонима для Старый тип

```
struct Vector{  
double x, y, z;  
};
```

```
struct Vector v = { 3, 0, 4 };
```

```
typedef struct Vector{  
double x, y, z;  
} Vector;
```

```
Vector q = { 3, 0, 4 };
```

Еще немного typedef'a

```
void printok(void* a, void* b, int (*check)(void*, void*))
{
    if (check(a, b))
        printf("ok\n");
    else
        printf("ne ok\n");
}
```

```
typedef int (*CmpFunc)(void*, void*);

void printok(void* a, void* b, CmpFunc check)
{
    if (check(a, b))
        printf("ok\n");
    else
        printf("ne ok\n");
}
```



Структуры в структурах

```
typedef struct Vector{  
    double x, y, z;  
} Vector;
```

```
typedef struct Body{  
    Vector position;  
    Vector speed;  
} Body;
```

Структуры в структурах

```
typedef struct Group {  
    char *name;  
    int size;  
} Group;
```

Есть ли в таком подходе проблемы???

```
typedef struct Student{  
    char *name;  
    Group group;  
} Student;
```


Структуры в структурах

```
typedef struct Group {  
    char *name;  
    int size;  
    Student* starosta;  
} Group;
```

```
typedef struct Student{  
    char *name;  
    Group *group;  
} Student;
```

Иначе будет столько групп, сколько студентов...

Все, что не принадлежит этой структуре – делаем указателем.



Вроде все

**ТЫ НЕ ПОЛУЧИШЬ ОШИБКУ
КОМПИЛЯЦИИ,**

**ЕСЛИ НЕ БУДЕШЬ
КОМПИЛИРОВАТЬ КОД**