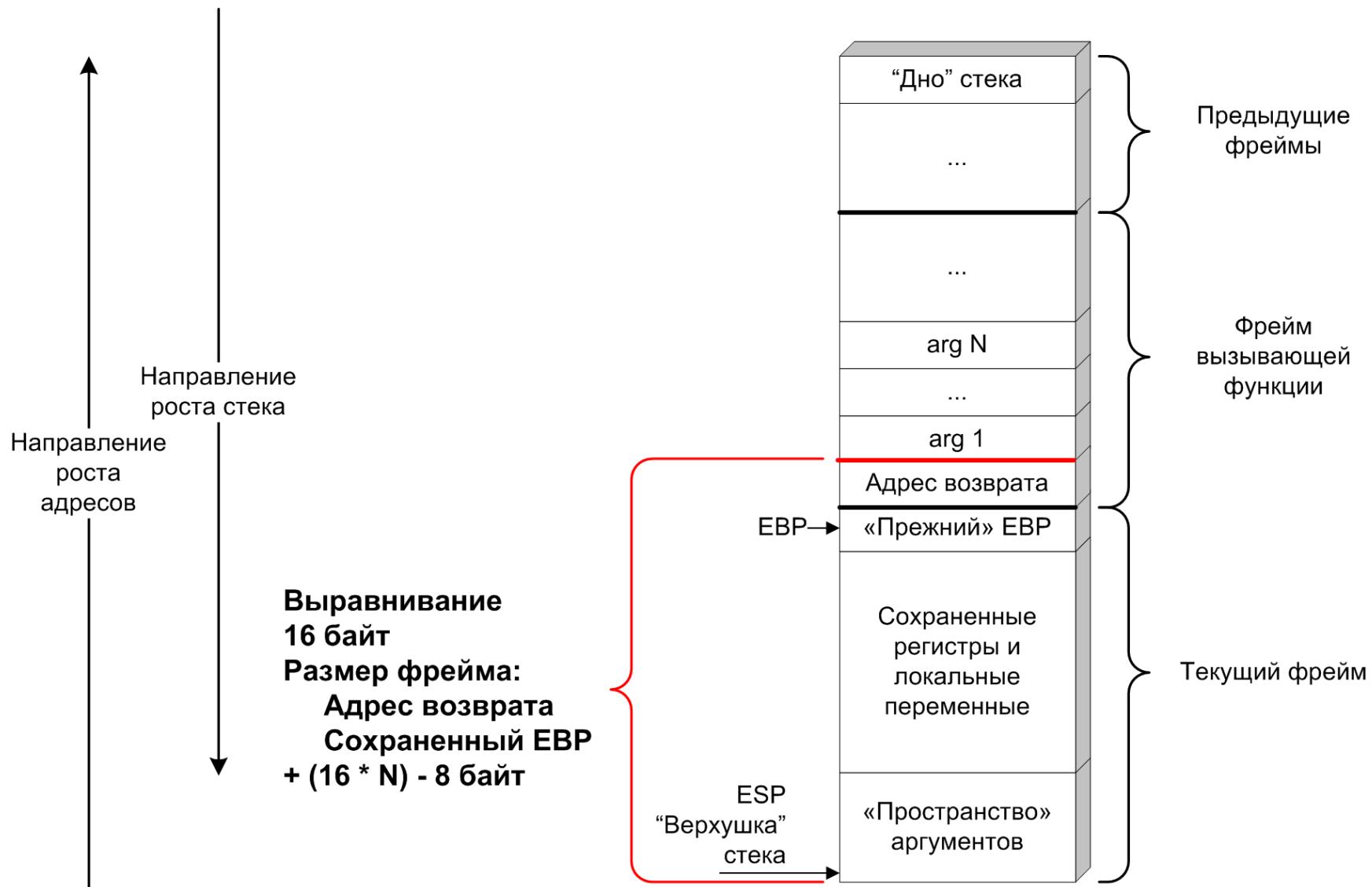


# Лекция 15

30 марта



# Стандартная библиотека языка Си

- 24 заголовочных файла
- stdlib.h
  - Преобразование типов: atoi, strtod, ...
  - Генерация псевдослучайных последовательностей
  - Выделение и освобождение памяти
  - Сортировка и поиск
  - Математика
- stdio.h
  - Функции для файловых операций
  - Функции для операций ввода-вывода
- string.h
- ...

# STDCALL

```
#include <stdio.h>

__attribute__((stdcall))
int sum(int x, int y);

int main() {
    int a = 1, b = 2, c;
    c = sum(a, b);
    printf("%d\n", c);
    return 0;
}

__attribute__((stdcall))
int sum(int x, int y) {
    int t = x + y;
    return t;
}
```

sum:

push	ebp
mov	ebp, esp
sub	esp, 16
mov	edx, dword [ebp+12]
mov	eax, dword [ebp+8]
add	eax, edx
mov	dword [ebp-4], eax
mov	eax, dword [ebp-4]
leave	
ret	8

# STDCALL

```
#include <stdio.h>

__attribute__((stdcall))
int sum(int x, int y);

int main() {
    int a = 1, b = 2, c;
    c = sum(a, b);
    printf("%d\n", c);
    return 0;
}

__attribute__((stdcall))
int sum(int x, int y) {
    int t = x + y;
    return t;
}
```

CMAIN:

```
; ...
mov    eax, dword [ebp-12]
mov    dword [esp+4], eax
mov    eax, dword [ebp-16]
mov    dword [esp], eax
call   sum
sub   esp, 8
mov    dword [ebp-8], eax
; ...
```

# FASTCALL

```
#include <stdio.h>

__attribute__((fastcall)) int
sub(int x, int y);

int main() {
    int c = sub(1, 2);
    printf("%d\n", c);
    return 0;
}

__attribute__((fastcall)) int
sub(int x, int y) {
    int t = x - y;
    return t;
}
```

```
CMAIN:
    ; ...
    mov    edx, 2
    mov    ecx, 1
    call   sub
    mov    dword [esp+20], eax
    ; ...

sub:
    sub   ecx, edx
    mov    eax, ecx
    ret
```

# Выравнивание стека - итоги

- На стек помещаем только машинные слова (4 байта)
- Выравнивание стека по границе в 16 байт  
IA-32/Linux/gcc
  - выполняется в функции main
  - остальные функции поддерживают выравнивание, формируя фрейм определенного размера
  - для листовых функций необязательно
- Не только производительность: команда MOVDQA
  - Быстрое копирование блока данных размером 16 байт
  - Аварийное завершение работы (#GP), если начальный адрес блока не выровнен по границе в 16 байт

# Отказ от указателя фрейма

-fomit-frame-pointer

```
int f(int x, int y) {
    int numerator =
        (x + y) * (x - y);
    int denominator =
        x * x + y * y;
    if (0 == denominator) {
        denominator = 1;
    }
    return (100 * numerator) /
        denominator;
}
```

Регистр	Значение
esi	y
ecx	x

f:

```
; пролог
sub    esp, 8
mov    dword [esp+4], esi
mov    esi, dword [esp+16]
mov    ecx, dword [esp+12]
mov    dword [esp], ebx
; ...
```

Сохраненный регистр	адрес
esi	[esp + 4]
ebx	[esp]

# Отказ от указателя фрейма

```
int f(int x, int y) {
    int numerator =
        (x + y) * (x - y);
    int denominator =
        x * x + y * y;
    if (0 == denominator) {
        denominator = 1;
    }
    return (100 * numerator) /
        denominator;
}
```

Регистр	Значение
esi	y
ecx	x

```
f:
; ...
mov    edx, esi
imul   edx, esi ; edx = y^2
mov    eax, ecx
imul   eax, ecx ; eax = x^2
mov    ebx, edx
add    ebx, eax
; ebx = x^2 + y^2
jne    .L2
mov    ebx, 1
.L2
; ...
```

# Отказ от указателя фрейма

```
int f(int x, int y) {
    int numerator =
        (x + y) * (x - y);
    int denominator =
        x * x + y * y;
    if (0 == denominator) {
        denominator = 1;
    }
    return (100 * numerator) /
        denominator;
}
```

```
f:
; ...
.L2
    lea     edx, [esi+ecx]
    sub     ecx, esi
    imul   edx, ecx
; ...
```

Регистр	Значение
esi	y
ecx	x
ebx	$x^2 + y^2$

# Отказ от указателя фрейма

```
int f(int x, int y) {
    int numerator =
        (x + y) * (x - y);
    int denominator =
        x * x + y * y;
    if (0 == denominator) {
        denominator = 1;
    }
    return (100 * numerator) /
        denominator;
}
```

f:

```
; ...
imul    edx, edx, 100
mov     eax, edx
sar     edx, 31
idiv    ebx
; ...
```

Регистр	Значение
esi	y
ecx	x
ebx	$x^2 + y^2$
edx	$(x + y) * (x - y)$

# Отказ от указателя фрейма

```
int f(int x, int y) {  
    int numerator =  
        (x + y) * (x - y);  
    int denominator =  
        x * x + y * y;  
    if (0 == denominator) {  
        denominator = 1;  
    }  
    return (100 * numerator) /  
        denominator;  
}
```

```
f:  
; ...  
; эпилог  
mov    esi, dword [esp+4]  
mov    ebx, dword [esp]  
add    esp, 8  
ret
```

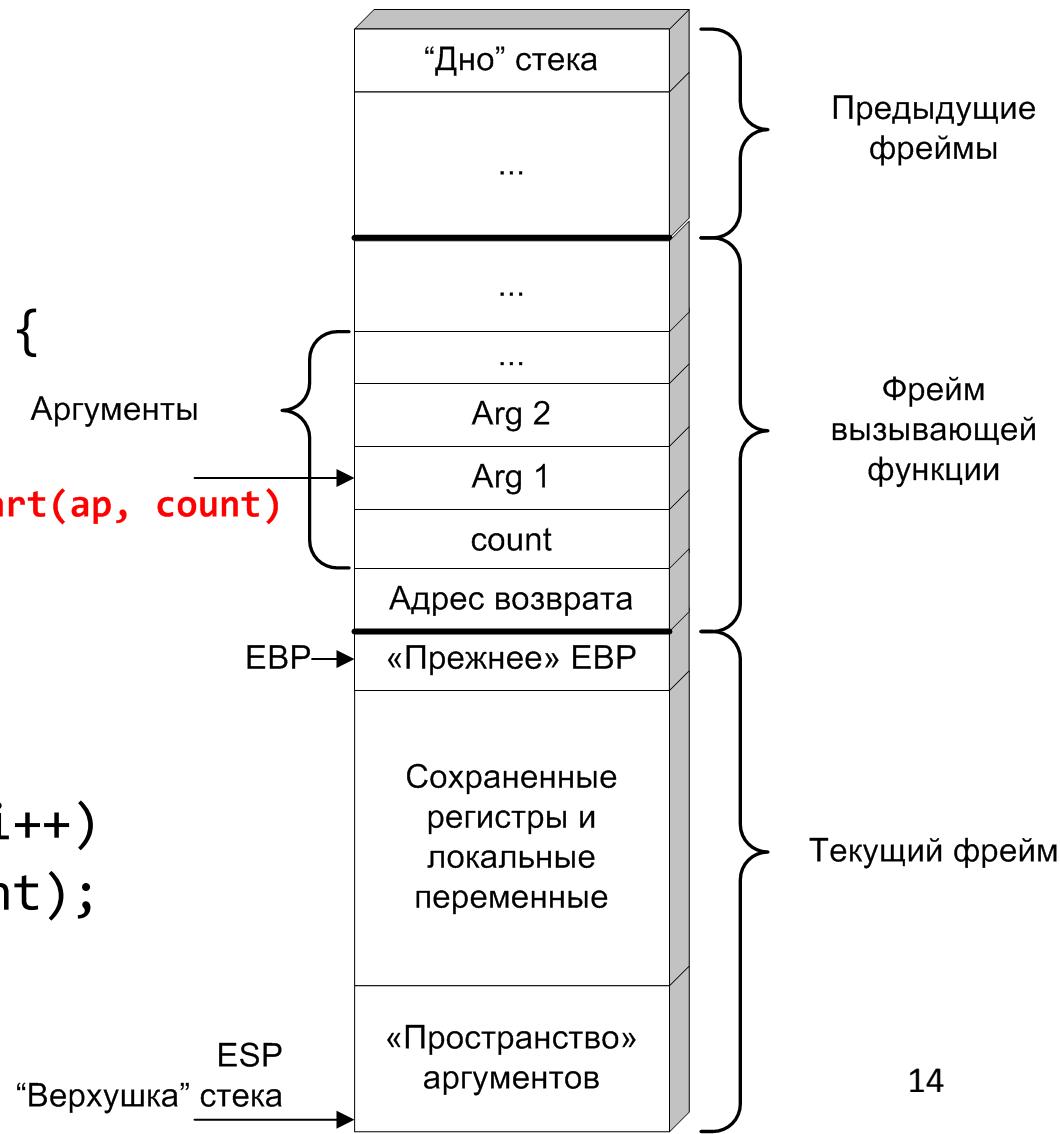
# Переменное количество параметров

- Многоточие (...) помещается в конце списка параметров.
- Тип данных
  - va\_list
- Макрокоманды
  - va\_start(va\_list, last fixed param)
  - va\_arg(va\_list, cast type)
  - va\_end(va\_list)

# Переменное количество параметров

```
#include <stdarg.h>

int average(int count, ...) {
    va_list ap;
    int sum = 0;
    if (0 == count) {
        return -1;
    }
    va_start(ap, count);
    for (int i=0; i<count; i++)
        sum += va_arg(ap, int);
    va_end(ap);
    return sum/count;
}
```



# Переменное количество параметров

```
#include <stdarg.h>

int average(int count, ...) {
    va_list ap;
    int sum = 0;
    if (0 == count) {
        return -1;
    }
    va_start(ap, count);
    for (int i=0; i<count; i++)
        sum += va_arg(ap, int);
    va_end(ap);
    return sum/count;
}
```

average:

```
push ebp
mov  ebp, esp
push ebx
mov  ecx, dword [ebp+8]
test ecx, ecx
jne  .L11
mov  eax, -1
pop  ebx
pop  ebp
ret
.L11:
; ...
```

# Переменное количество параметров

```
#include <stdarg.h>

int average(int count, ...) {
    va_list ap;
    int sum = 0;
    if (0 == count) {
        return -1;
    }
    va_start(ap, count);
    for (int i=0; i<count; i++)
        sum += va_arg(ap, int);
    va_end(ap);
    return sum/count;
}
```

average:

```
; ...
.L11:
    xor    eax, eax
    xor    edx, edx
    test   ecx, ecx
    lea    ebx, [ebp+12]
    jle   .L5
.L8:
    add    edx, dword [ebx+eax*4]
    add    eax, 1
    cmp    ecx, eax
    jg    .L8
.L5:
```

# Переменное количество параметров

```
#include <stdarg.h>

int average(int count, ...) {
    va_list ap;
    int sum = 0;
    if (0 == count) {
        return -1;
    }
    va_start(ap, count);
    for (int i=0; i<count; i++)
        sum += va_arg(ap, int);
    va_end(ap);
    return sum/count;
}
```

average:

; ...

.L11:

xor eax, eax

xor edx, edx

test ecx, ecx

lea ebx, [ebp+12]

jle .L5

.L8:

add edx, dword [ebx+eax\*4]

add eax, 1

cmp ecx, eax

jg .L8

.L5:

# Переменное количество параметров

```
#include <stdarg.h>

int average(int count, ...) {
    va_list ap;
    int sum = 0;
    if (0 == count) {
        return -1;
    }
    va_start(ap, count);
    for (int i=0; i<count; i++)
        sum += va_arg(ap, int);
    va_end(ap);
    return sum/count;
}
```

average:  
; ...  
.L5:  
    mov    eax, edx  
    sar    edx, 31  
    idiv   ecx  
    pop    ebx  
    pop    ebp  
    ret

# Вызов функции по указателю

```
#include <stdio.h>

int sum(int, int);
int sub(int, int);
int mul(int, int);
int div(int, int);

typedef int (*arith)(int, int);
int eval(arith pf, int x, int y);

int main() {
    int a = 1, b = 2, c;
    arith pf = sum;
    c = eval(pf, a, b);
    printf("%d\n", c);
    return 0;
}
```

```
int eval(arith pf, int x, int y) {
    return pf(x, y);
}

int sum(int x, int y) {
    return x + y;
}

int sub(int x, int y) {
    return x - y;
}

int mul(int x, int y) {
    return x * y;
}

int div(int x, int y) {
    return x / y;
}
```

# Вызов функции по указателю

```
int sum(int x, int y) {  
    return x + y;  
}
```

```
int sub(int x, int y) {  
    return x - y;  
}
```

```
global sum  
sum:  
    push    ebp  
    mov     ebp, esp  
    mov     eax, dword [ebp+12]  
    add     eax, dword [ebp+8]  
    pop     ebp  
    ret
```

```
global sub  
sub:  
    push    ebp  
    mov     ebp, esp  
    mov     eax, dword [ebp+8]  
    sub     eax, dword [ebp+12]  
    pop     ebp  
    ret
```

# Вызов функции по указателю

```
int mul(int x, int y) {  
    return x * y;  
}
```

```
int div(int x, int y) {  
    return x / y;  
}
```

```
global mul  
mul:  
    push    ebp  
    mov     ebp, esp  
    mov     eax, dword [ebp+12]  
    imul    eax, dword [ebp+8]  
    pop     ebp  
    ret
```

```
global div  
div:  
    push    ebp  
    mov     ebp, esp  
    mov     edx, dword [ebp+8]  
    mov     eax, edx ; cdq  
    sar     edx, 31 ;  
    idiv    dword [ebp+12]  
    pop     ebp  
    ret
```

# Вызов функции по указателю

```
int main() {
    int a = 1, b = 2, c;
    arith pf = sum;
    c = eval(pf, a, b);
    printf("%d\n", c);
    return 0;
}
```

```
%include 'io.inc'
section .rodata
LC0: db '%d', 10, 0
```

```
CEXTERN printf
```

```
section .text
global CMAIN
CMAIN:
    lea      ecx, [esp+4]
    and      esp, -16
    push    dword [ecx-4]
```

push	ebp
mov	ebp, esp
push	ecx
sub	esp, 20
mov	dword [esp+8], 2
mov	dword [esp+4], 1
mov	dword [esp], sum
call	eval
mov	dword [esp+4], eax
mov	dword [esp], LC0
call	printf
mov	eax, 0
add	esp, 20
pop	ecx
pop	ebp
lea	esp, [ecx-4]
ret	

# Вызов функции по указателю

```
int eval(arith pf, int x, int y) {  
    return pf(x, y);  
}
```

	global eval
eval:	
	push    ebp
	mov     ebp, esp
	sub     esp, 8
	mov     edx, dword [ebp+12]
	mov     eax, dword [ebp+16]
	mov     ecx, dword [ebp+8]
	mov     dword [esp], edx
	mov     dword [esp+4], eax
	call    ecx
	mov     esp, ebp
	pop    ebp
	ret

# Ассемблерные вставки

- Нет единого стандарта
- Пример: gcc
  - Наиболее развитый механизм
  - Естественный синтаксис ассемблера для компилятора gcc - **AT&T**

```
__asm__ ("mov %1, %%eax\n"
         "mov %%eax, %0\n"
         ...
```

```
int f() /* тоже самое для синтаксиса Intel
    int a=10, b;
    __asm__ (".intel_syntax noprefix\n"
             "mov eax, %1\n"
             "mov %0, eax\n" /* ассемблерная вставка */
             : "=r"(b)        /* выходные операнды */
             : "r"(a)         /* входные операнды */
             : "%eax"         /* разрушаемые регистры */
             );
    return b;
}
```

```
f:
    push    ebp
    mov     edx, 10
    mov     ebp, esp
#APP
# 3 "asm_inline.c" 1
.intel_syntax noprefix
    mov eax, edx
    mov edx, eax
# 0 "" 2
#NO_APP
    pop    ebp
    mov    eax, edx
    ret
```

# Динамическое выделение памяти на стеке

```
#include <alloca.h>

int f(int dataSize, int iter)
{
    for (int i = 0; i < iter; ++i)
    {
        char *p = alloca( dataSize );
// выделенная память не будет
// освобождена после закрывающей
// скобки на следующей строке
    }
    return 0;
}
// память, выделенная alloca(),
// освобождается здесь
```

alloca не входит в стандарт языка Си

<pre>f:</pre>	<pre>push    ebp mov     ebp, esp sub     esp, 8 xor     edx, edx mov     eax, dword [ebp+8] mov     ecx, dword [ebp+12] add     eax, 30 and     eax, -16 jmp     .L2 .L3: sub     esp, eax inc     edx .L2: cmp     edx, ecx j1      .L3 xor     eax, eax leave ret</pre>
---------------	--

# Вызов функций – заключение

- Порядок вызова функций образует стек (call / ret)
  - Если Р вызывает Q, то Q завершается до завершения Р
- Рекурсия (в том числе косвенная ) корректно реализуется через общее соглашение о вызове функций
  - Фрейм используется для размещения локальных переменных и сохранения значений регистров
  - Аргументы для вызова очередной функции размещаются на «верхушке» стека
  - Результат возвращается через регистр eax
- Параметры передаются по значению

