Лекция 15

2 апреля

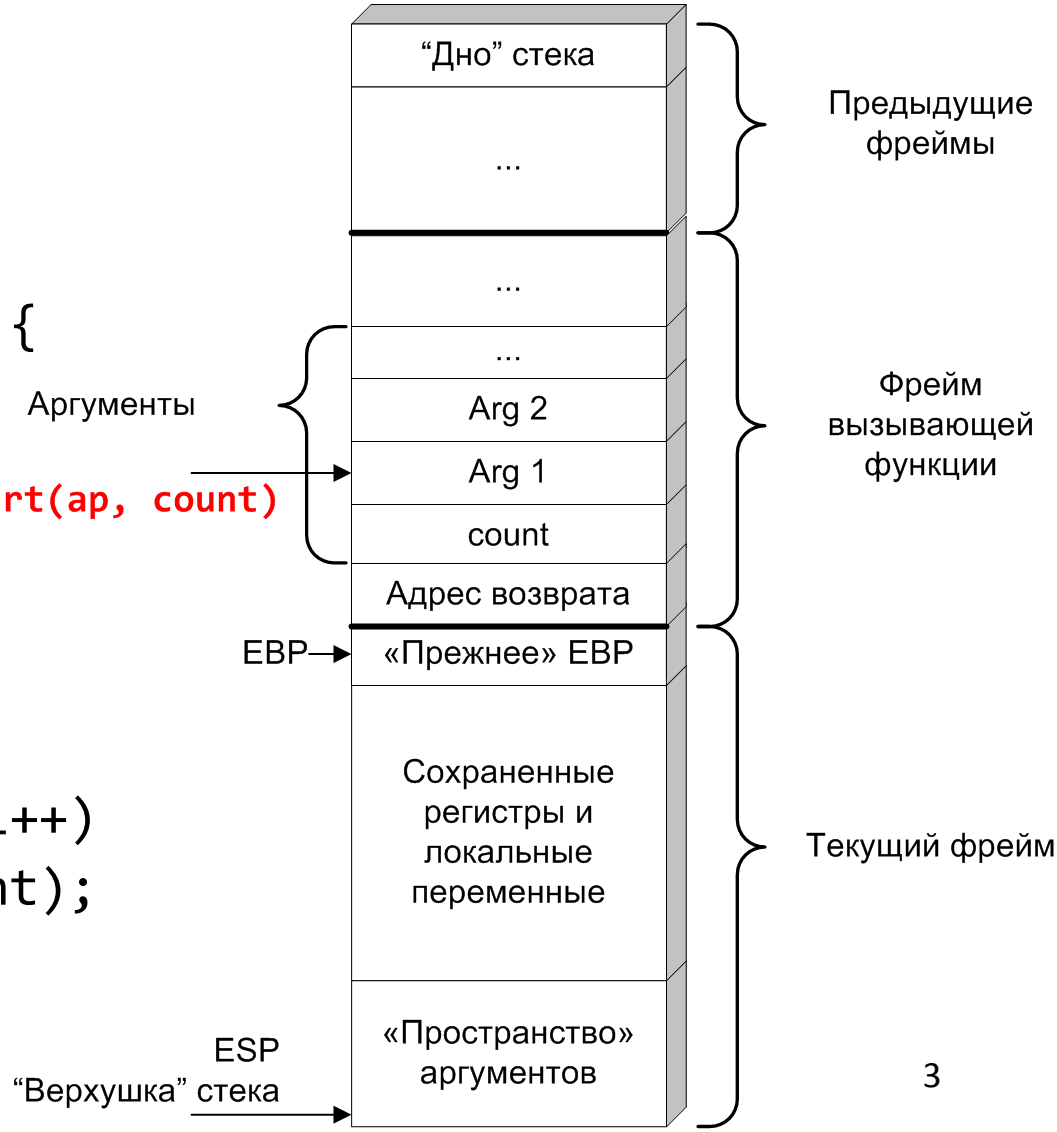
Переменное количество параметров

- 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

```c
#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;
}
```

```asm
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

```c
#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;
}
```

```asm
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

```c
#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;
}
```

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

Вызов функции по указателю

```
#include <stdio.h>

typedef int (*arith)(int, int);

#define OP(x) \
  int x (int, int);\

#include "eval.def"
#undef OP

#define OP(x) x, \

arith ops[] = {
#include "eval.def"
   NULL
};
#undef OP

#define OP(x) CODE_ ## x, \

typedef enum {
#include "eval.def"
   CODE_LAST
} opcode;
#undef OP
```

```
int eval(opcode c, int x, int y) {
   return ops[c](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;
}
```

```
OP(sum)
OP(sub)
OP(mul)
OP(div)
```

eval.def

Вызов функции по указателю

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

int

}
```

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

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

```
global sum
sum:

lo
ub
```

```
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, res;
   opcode code = CODE_sum;
   res = eval(code, a, b);
   printf("%d\n", res);
   return 0;
}
```

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

CEXTERN printf

section .text
global CMAIN
CMAIN:
   push    ebp
   mov     ebp, esp
   and     esp, -16
   sub     esp, 32
```

```
   mov     dword [esp+28], 1
   mov     dword [esp+24], 2
   mov     dword [esp+16], 0
   mov     eax, dword [esp+24]
   mov     dword [esp+8], eax
   mov     eax, dword [esp+28]
   mov     dword [esp+4], eax
   mov     eax, dword [esp+16]
   mov     dword [esp], eax
   call    eval
   mov     dword [esp+20], eax
   mov     eax, .LC0
   mov     edx, dword [esp+20]
   mov     dword [esp+4], edx
   mov     dword [esp], eax
   call    printf
   mov     eax, 0
   leave
   ret
```

Вызов функции по указателю

```
int eval(opcode c, int x, int y) {
   return ops[c](x, y);
}
```

```
section .data
 ops: dd sum, sub, mul, div, 0

section .text
global eval
eval:
   push  ebp
   mov   ebp, esp
   sub   esp, 24
   mov   eax, dword [ebp+8]
   mov   edx, dword [ops+eax*4]
   mov   eax, dword [ebp+16]
   mov   dword [esp+4], eax
   mov   eax, dword [ebp+12]
   mov   dword [esp], eax
   call  edx
   leave
   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

```c
#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
```

```
f:
   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
   jl      .L3
   xor     eax, eax
   leave
   ret
```

`alloca` не входит в стандарт языка Си