

# Ассемблер x86-64 (AMD64). Базовая техника



the  
netwide  
assembler

```
(gdb) disasm
Dump of assembler code for function S_enter:
0x000000000407091 <+0>: push   %rbp
0x000000000407092 <+1>: mov    %rsp,%rbp //basic type: string
0x000000000407095 <+4>: sub   $0x20,%rsp
0x000000000407099 <+8>: mov   %rdi,-0x8(%rbp) string = S_Symbol("string");
0x00000000040709d <+12>: mov   %rsi,0x10(%rbp)ter(table, ty_string, Ty_String());
=> 0x0000000004070a1 <+16>: mov   %rdx,-0x18(%rbp)
0x0000000004070a5 <+20>: mov   -0x18(%rbp),%rdx rdn table;
0x0000000004070a9 <+24>: mov   -0x10(%rbp),%rcx
0x0000000004070ad <+28>: mov   -0x8(%rbp),%rax
0x0000000004070b1 <+32>: mov   %rcx,%rsi
0x0000000004070b4 <+35>: mov   %rax,%rdi table E_base_venv(void)
0x0000000004070b7 <+38>: callq 0x40618e <TAB_enter>
0x0000000004070bc <+43>: nop
0x0000000004070bd <+44>: leaveq %rsi S_table venv;
0x0000000004070be <+45>: retq
End of assembler dump.
(gdb) x $rbp-0x18
0x7fffffffcc8: 0x0061a030
(gdb) p $rdx
$17 = 640235246
(gdb) p/x $rdx
$18 = 0x61b130
(gdb) nt
Breakpoint 5, S_enter (t=0x61ad20, syn=0x61b160, value=0x61b7d0) at symbol.c:53
53 TAB_enter(t,syn,value);
(gdb) disasm
Dump of assembler code for function S_enter:
0x000000000407091 <+0>: push   %rbp
0x000000000407092 <+1>: mov    %rsp,%rbp
0x000000000407095 <+4>: sub   $0x20,%rsp
0x000000000407099 <+8>: mov   %rdi,-0x8(%rbp)l1t = Ty_Int();
0x00000000040709d <+12>: mov   %rsi,-0x10(%rbp)
=> 0x0000000004070a1 <+16>: mov   %rdx,-0x18(%rbp)
0x0000000004070a5 <+20>: mov   -0x18(%rbp),%rdx %malls = checked_malloc(sizeof(*formals
0x0000000004070a9 <+24>: mov   -0x10(%rbp),%rcx%alls->head = Ty_Int();
0x0000000004070ad <+28>: mov   -0x8(%rbp),%rax%alls->tail = NULL;
0x0000000004070b1 <+32>: mov   %rcx,%rsi S_enter(venv,S_Symbol("exit"),E_FunEntry
0x0000000004070b4 <+35>: mov   %rax,%rdi
0x0000000004070b7 <+38>: callq 0x40618e <TAB_enter>
0x0000000004070bc <+43>: nop
0x0000000004070bd <+44>: leaveq %rsi S_enter(venv,S_Symbol("not"),E_FunEntry
0x0000000004070be <+45>: retq
End of assembler dump.
(gdb) x $rbp-0x18
0x7fffffffcc8: 0x0061b7d0
```



# Используемые источники

Сайт программы: <https://nasm.us/>

Википедия: Ассемблер - <https://ru.wikipedia.org/wiki/Ассемблер>

Команды x86, x86-64: <http://ccfit.nsu.ru/~kireev/lab2/lab2com.htm>

Регистры x86, x86-64: <http://ccfit.nsu.ru/~kireev/lab2/lab2reg.htm>

Онлайн компиляторы, использующие NASM x86-64:

[https://www.mycompiler.io/new/asm-x86\\_64](https://www.mycompiler.io/new/asm-x86_64)

<https://ideone.com/>

Язык ассемблера для Intel x86 - Урок #4 - Целые числа, регистр EFLAGS, переполнение и арифметика: <https://www.youtube.com/watch?v=dfv2VN6Xk0o>

Learn programming through books and examples: <https://riptutorial.com/>

***Лекционный материал по теме. Презентации, видео, примеры:***

<http://softcraft.ru/edu/comparch/lect/09-IntelAsmBase/>

## Язык С. Статически типизированное решение с использованием АТД

```
typedef struct rectangle {  
    int x, y; // ширина, высота  
} rectangle;
```

```
void InRndRectangle(rectangle *r) {  
    r->x = Random();  
    r->y = Random();  
}
```

```
void OutRectangle(rectangle *r, FILE *ofst) {  
    fprintf(ofst, "It is Rectangle: x = %d, y = %d. Perimeter = %g\n",  
        r->x, r->y, PerimeterRectangle(r));  
}
```

```
double PerimeterRectangle(rectangle *r) {  
    return 2.0 * (r->x + r->y);  
}
```

## Язык С. Статически типизированное решение с использованием АД

```
typedef enum key {RECTANGLE, TRIANGLE} key;
typedef struct shape {
    key k;
    union {
        rectangle r;
        triangle t;
    };
} shape;
```

```
bool InShape(shape *s, FILE *ifst) {
    int k;
    fscanf(ifst, "%d", &k);
    switch(k) {
        case 1:
            s->k = RECTANGLE;
            InRectangle(&(s->r), ifst);
            return true;
        case 2:
            s->k = TRIANGLE;
            InTriangle(&(s->t), ifst);
            return true;
        default:
            return false;
    }
}
```

```
void OutShape(shape *s, FILE *ofst) {
    switch(s->k) {
        case RECTANGLE:
            OutRectangle(&(s->r), ofst); break;
        case TRIANGLE:
            OutTriangle(&(s->t), ofst); break;
        default:
            fprintf(ofst, "Incorrect figure!\n");
    }
}
```

```
double PerimeterShape(shape *s) {
    switch(s->k) {
        case RECTANGLE:
            return PerimeterRectangle(&(s->r));
            break;
        case TRIANGLE:
            return PerimeterTriangle(&(s->t));
            break;
        default:
            return 0.0;
    }
}
```

## Язык C. Статически типизированное решение с использованием АД

```
typedef struct container {
    int len; // текущая длина
    shape cont[max_len];
} container;

void InContainer(container *c, FILE *ifst) {
    while(!feof(ifst)) {
        if(InShape(&((c->cont)[c->len]), ifst)) {
            c->len++;
        }
    }
}

void OutContainer(container *c, FILE *ofst) {
    fprintf(ofst, "Container contains %d elements.\n", c->len);
    for(int i = 0; i < c->len; i++) {
        fprintf(ofst, "%d: ", i);
        OutShape(&(c->cont)[i], ofst);
    }
}

double PerimeterSumContainer(container *c) {
    double sum = 0.0;
    for(int i = 0; i < c->len; i++) {
        sum += PerimeterShape(&(c->cont)[i]);
    }
    return sum;
}
```

## Язык С. Бестиповое решение

```
//-----  
// data.h - Описание данных бестиповой программы  
//-----  
  
//-----  
// Раздел констант  
//-----  
  
// Константа, определяющая размер целого числа  
int const intSize = sizeof(int);  
// Константа, задающая размер для прямоугольника  
int const rectSize = 2 * sizeof(int);  
// Константа, задающая размер для треугольника  
int const trianSize = 3 * sizeof(int);  
// Константа, задающая размер для треугольника  
int const shapeSize = sizeof(int) + (rectSize >= trianSize ? rectSize : trianSize);  
// Константа, определяющая размерность массива фигур  
int const maxSize = 10000 * shapeSize;  
// Константа, задающая признак прямоугольника  
int const RECTANGLE = 1;  
// Константа, задающая признак треугольника  
int const TRIANGLE = 2;
```

```
int main(int argc, char* argv[]) {  
    // Массив используемый для хранения данных  
    //unsigned int cont[maxSize / intSize];  
    //int cont[maxSize / intSize];  
    unsigned char cont[maxSize];  
    // Количество элементов в массиве  
    int len = 0;
```

## Язык С. Бестиповое решение

```
// Ввод параметров прямоугольника из файла
void InRectangle(void *r, FILE *ifst) {
    fscanf(ifst, "%d%d", (int*)r, (int*)(r+intSize));
}
```

```
// Ввод параметров треугольника из файла
void InTriangle(void *t, FILE *ifst) {
    fscanf(ifst, "%d%d%d", (int*)t,
          (int*)(t+intSize), (int*)(t+2*intSize));
}
```

```
// Ввод параметров обобщенной фигуры из файла
int InShape(void *s, FILE *ifst) {
    int k;
    fscanf(ifst, "%d", &k);
    switch(k) {
        case 1:
            *((int*)s) = RECTANGLE;
            InRectangle(s+intSize, ifst);
            return 1;
        case 2:
            *((int*)s) = TRIANGLE;
            InTriangle(s+intSize, ifst);
            return 1;
        default:
            return 0;
    }
}
```

```
// Ввод содержимого контейнера из указанного файла
void InContainer(void *c, int *len, FILE *ifst) {
    void *tmp = c;
    while(!feof(ifst)) {
        if(InShape(tmp, ifst)) {
            tmp = tmp + shapeSize;
            (*len)++;
        }
    }
}
```



## Язык С. Бестиповое решение

```
// Вычисление периметра прямоугольника
```

```
double PerimeterRectangle(void *r) {  
    return 2.0 * (*((int*)r) + (*((int*)(r+intSize))));  
}
```

```
// Вычисление периметра треугольника
```

```
double PerimeterTriangle(void *t) {  
    return (double)(*((int*)t) +  
            (*((int*)(t+intSize)) +  
            (*((int*)(t+2*intSize))));  
}
```

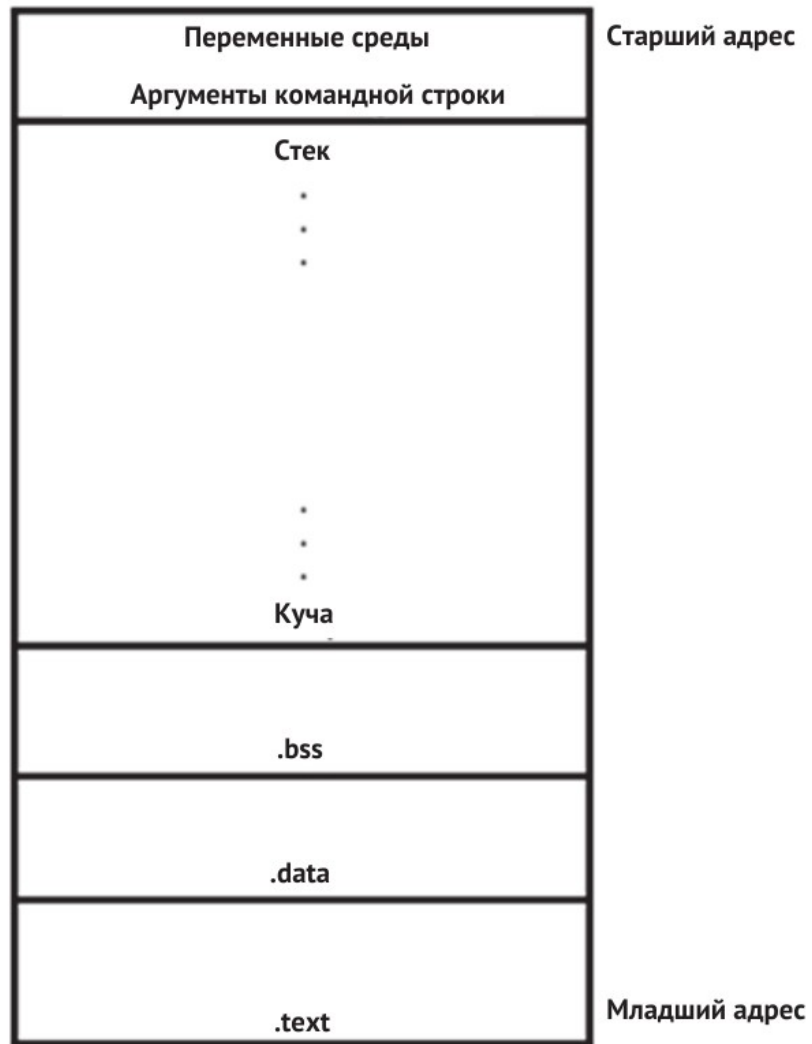
```
// Вычисление периметра фигуры
```

```
double PerimeterShape(void *s) {  
    int k = (*((int*)s);  
    if(k == RECTANGLE) {  
        return PerimeterRectangle(s+intSize);  
    }  
    else if(k == TRIANGLE) {  
        return PerimeterTriangle(s+intSize);  
    }  
    else {  
        return 0.0;  
    }  
}
```

```
// Вычисление суммы периметров всех фигур в контейнере
```

```
double PerimeterSumContainer(void *c, int len) {  
    double sum = 0.0;  
    void *tmp = c;  
    for(int i = 0; i < len; i++) {  
        sum += PerimeterShape(tmp);  
        tmp = tmp + shapeSize;  
    }  
    return sum;  
}
```

## Организация памяти, выделяемой программе



# Модуль тестирования программы вычисления периметров и суммы периметров

```
1 ; main.asm
2 extern PerimeterRectangle
3 extern PerimeterTriangle
4 extern PerimeterShape
5 extern PerimeterSumContainer
6
7 global RECTANGLE
8 global TRIANGLE
9
10 extern printf ; declare the function as alien
11 section .data
12 i dd 10
13 fmtstr db "%d -> %f",10,0 ; printf format
14 pfmt db "Perimeter = %d, %f",10,0 ; printf format
15 pcfmt db "Common Perimeter = %f",10,0 ; printf format
16 RECTANGLE dd 1
17 TRIANGLE dd 2
18 shapeRect dd 1
19 rect dd 5,7 ; прямоугольник с двумя сторонами
20 empty dd 0 ; пустое слово в фигуре для прямоугольника
21 shapeTrian dd 2
22 trian dd 3,4,5 ; три стороны треугольника
23 ; Массив фигур
24 cont dd 1,2,3,0, 2,3,4,5, 1,2,8,0, 2,2,2,2, 2,5,7,4, 1,5,6,0
25 len dd 6
26 section .bss
27 d resq 1
28 intP resd 1 ; под целочисленное значение периметра
29 p resq 1 ; периметр как действительное
30 section .text
31 global main
32 ;-----
33 main:
34 push rbp
35 mov rbp, rsp
36
37 mov eax, [i]
38 cvtsi2sd xmm0, eax
39 movsd [d], xmm0
40
41 mov rdi, fmtstr ; first argument for printf
42 mov rsi, [i] ; second argument for printf
43 movsd xmm0, [d] ; third argument for printf
44 mov rax, 1 ; xmm registers involved
45 call printf ; call the function
46
47 ; Вычисление периметра прямоугольника
48 mov rdi, rect
49 call PerimeterRectangle
50 mov [intP], eax
51 movsd [p], xmm0
52
53 ; Вывод периметра прямоугольника
54 mov rdi, pfmt
55 mov rsi, rax
56 movsd xmm0, [p] ; third argument for printf
57 mov rax, 1 ; only integer registers involved
58 call printf ; call the function
59
60 ; Вычисление периметра треугольника
61 mov rdi, trian
62 call PerimeterTriangle
63 mov [intP], eax
64 movsd [p], xmm0
65
66 ; Вывод периметра треугольника
67 mov rdi, pfmt
68 mov rsi, rax
69 movsd xmm0, [p] ; third argument for printf
70 mov rax, 1 ; only integer registers involved
71 call printf ; call the function
72
73 ; Вычисление периметра фигуры-прямоугольника
74 mov rdi, shapeRect
75 call PerimeterShape
76 mov [intP], eax
77 movsd [p], xmm0
78
79 ; Вывод периметра фигуры-прямоугольника
80 mov rdi, pfmt
81 mov rsi, [intP]
82 movsd xmm0, [p] ; third argument for printf
83 mov rax, 1 ; only integer registers involved
84 call printf ; call the function
85
86 ; Вычисление периметра фигуры-треугольника
87 mov rdi, shapeTrian
88 call PerimeterShape
89 mov [intP], eax
90 movsd [p], xmm0
91
92 ; Вывод периметра фигуры-треугольника
93 mov rdi, pfmt
94 mov rsi, [intP]
95 movsd xmm0, [p] ; third argument for printf
96 mov rax, 1 ; only integer registers involved
97 call printf ; call the function
98
99 ; Вычисление общего периметра 2-х фигур
100 mov rdi, shapeRect
101 mov rsi, 2
102 call PerimeterSumContainer
103 movsd [p], xmm0
104
105 ; Вывод суммарного периметра 2-х фигур
106 mov rdi, pcfmt
107 movsd xmm0, [p] ; third argument for printf
108 mov rax, 1 ; only integer registers involved
109 call printf ; call the function
110
111 ; Вычисление общего периметра нескольких фигур
112 mov rdi, cont
113 mov rsi, [len]
114 call PerimeterSumContainer
115 movsd [p], xmm0
116
117 ; Вывод периметра нескольких фигур
118 mov rdi, pcfmt
119 movsd xmm0, [p] ; third argument for printf
120 mov rax, 1 ; only integer registers involved
121 call printf ; call the function
122
123 pop rbp
124 mov rax, 60
125 mov rdi, 0
126 syscall
```

# Модуль вычисления периметров фигур и суммы периметров

```
1 ;-----  
2 ; perimeter.asm - единица компиляции, вбирающая функции вычисления периметра  
3 ;-----  
4
```

```
5 extern RECTANGLE  
6 extern TRIANGLE
```

```
7  
8 ;-----  
9 ; Вычисление периметра прямоугольника  
10 ;double PerimeterRectangle(void *r) {  
11 ;   return 2.0 * *((int*)r)  
12 ;     + *((int*)(r+intSize));  
13 ;}  
14 global PerimeterRectangle  
15 PerimeterRectangle:  
16 section .text  
17 push rbp  
18 mov rbp, rsp  
19  
20 ; В rdi адрес прямоугольника  
21 mov eax, [rdi]  
22 add eax, [rdi+4]  
23 shl eax, 1  
24 cvtsi2sd xmm0, eax  
25  
26 leave  
27 ret
```

```
28 ;-----  
29 ;double PerimeterTriangle(void *t) {  
30 ;   return (double)*((int*)t)  
31 ;     + *((int*)(t+intSize))  
32 ;     + *((int*)(t+2*intSize));  
33 ;}  
34  
35 global PerimeterTriangle  
36 PerimeterTriangle:  
37 section .text  
38 push rbp  
39 mov rbp, rsp  
40  
41 ; В rdi адрес треугольника  
42 mov eax, [rdi]  
43 add eax, [rdi+4]  
44 add eax, [rdi+8]  
45 cvtsi2sd xmm0, eax
```

```
46 leave  
47 ret  
48  
49
```

```
50 ;-----  
51 ; Вычисление периметра фигуры  
52 ;double PerimeterShape(void *s) {  
53 ;   int k = *((int*)s);  
54 ;   if(k == RECTANGLE) {  
55 ;     return PerimeterRectangle(s+intSize);  
56 ;   }  
57 ;   else if(k == TRIANGLE) {  
58 ;     return PerimeterTriangle(s+intSize);  
59 ;   }  
60 ;   else {  
61 ;     return 0.0;  
62 ;   }  
63 ;}  
64 global PerimeterShape  
65 PerimeterShape:  
66 section .text  
67 push rbp  
68 mov rbp, rsp  
69  
70 ; В rdi адрес фигуры  
71 mov eax, [rdi]  
72 cmp eax, [RECTANGLE]  
73 je rectPerimeter  
74 cmp eax, [TRIANGLE]  
75 je trianPerimeter  
76 xor eax, eax  
77 cvtsi2sd xmm0, eax  
78 jmp return  
79 rectPerimeter:  
80 ; Вычисление периметра прямоугольника  
81 add rdi, 4  
82 call PerimeterRectangle  
83 jmp return  
84 trianPerimeter:  
85 ; Вычисление периметра треугольника  
86 add rdi, 4  
87 call PerimeterTriangle  
88 return:  
89 leave  
90 ret  
91
```

```
92 ;-----  
93 ;// Вычисление суммы периметров всех фигур в контейнере  
94 ;double PerimeterSumContainer(void *c, int len) {  
95 ;   double sum = 0.0;  
96 ;   void *tmp = c;  
97 ;   for(int i = 0; i < len; i++) {  
98 ;     sum += PerimeterShape(tmp);  
99 ;     tmp = tmp + shapeSize;  
100 ;   }  
101 ;   return sum;  
102 ;}  
103 global PerimeterSumContainer  
104 PerimeterSumContainer:  
105 section .data  
106 .sum dq 0.0  
107 section .text  
108 push rbp  
109 mov rbp, rsp  
110  
111 ; В rdi адрес начала контейнера  
112 mov rbx, rsi ; число фигур  
113 xor rcx, rcx ; счетчик фигур  
114 movsd xmm1, [.sum] ; перенос накопителя суммы в регистр 1  
115 .loop:  
116 cmp rcx, rbx ; проверка на окончание цикла  
117 jge .return ; Перебрали все фигуры  
118  
119 mov r10, rdi ; сохранение начала фигуры  
120 call PerimeterShape ; Получение периметра первой фигуры  
121 addsd xmm1, xmm0 ; накопление суммы  
122 inc rcx ; индекс следующей фигуры  
123 add r10, 16 ; адрес следующей фигуры  
124 mov rdi, r10 ; восстановление для передачи параметра  
125 jmp .loop  
126 .return:  
127 movsd xmm0, xmm1  
128 leave  
129 ret
```

# NASM. Главный модуль. Секция данных.

```
; main.asm
extern PerimeterRectangle
extern PerimeterTriangle
extern PerimeterShape
extern PerimeterSumContainer

global RECTANGLE
global TRIANGLE

extern printf ; declare the function as alien
section .data
    i      dd      10
    fmtstr db      "%d -> %f",10,0      ; printf format
    pfmt   db      "Perimeter = %d, %f",10,0 ; printf format
    pcfmt  db      "Common Perimeter = %f",10,0 ; printf format
    RECTANGLE dd 1
    TRIANGLE  dd 2
    shapeRect dd 1
    rect      dd 5,7 ; прямоугольник с двумя сторонами
    empty     dd 0 ; пустое слово в фигуре для прямоугольника
    shapeTrian dd 2
    trian     dd 3,4,5 ; три стороны треугольника
    ; Массив фигур
    cont      dd 1,2,3,0, 2,3,4,5, 1,2,8,0, 2,2,2,2, 2,5,7,4, 1,5,6,0
    len       dd 6
section .bss
    d         resq 1
    intP      resd 1 ; под целочисленное значение периметра
    p         resq 1 ; периметр как действительное
```

## NASM. Главный модуль. Фрагмент проверки команд

```
section .text
    global main
;-----
main:
    push    rbp
    mov     rbp, rsp

    mov     eax, [i]
    cvtsi2sd    xmm0, eax
    movsd  [d], xmm0

    mov     rdi, fmtstr ; first argument for printf
    mov     rsi, [i]    ; second argument for printf
    movsd  xmm0, [d]    ; third argument for printf
    mov     rax, 1      ; xmm registers involved
    call   printf       ; call the function
```

# Соглашения о вызове внешних функций

## ***Целочисленные аргументы:***

- 1-й аргумент передается в регистре **rdi**;
- 2-й аргумент передается в регистре **rsi**;
- 3-й аргумент передается в регистре **rdx**;
- 4-й аргумент передается в регистре **rcx**;
- 5-й аргумент передается в регистре **r8**;
- 6-й аргумент передается в регистре **r9**.

Дополнительные аргументы передаются через стек в обратном порядке, чтобы можно было извлечь их в правильном порядке.

## ***Аргументы с плавающей точкой:***

- 1-й аргумент записывается в регистр **xmm0**;
- 2-й аргумент записывается в регистр **xmm1**;
- 3-й аргумент записывается в регистр **xmm2**;
- 4-й аргумент записывается в регистр **xmm3**;
- 5-й аргумент записывается в регистр **xmm4**;
- 6-й аргумент записывается в регистр **xmm5**;
- 7-й аргумент записывается в регистр **xmm6**;
- 8-й аргумент записывается в регистр **xmm7**.

## NASM. Разработка фрагмента кода вычисления периметра прямоугольника

```
;-----  
; perimeter.asm - единица компиляции, вбирающая функции вычисления периметра  
;-----  
  
extern RECTANGLE  
extern TRIANGLE  
  
;-----  
; Вычисление периметра прямоугольника  
;double PerimeterRectangle(void *r) {  
;    return 2.0 * (*((int*)r) + *((int*)(r+intSize)));  
;}  
global PerimeterRectangle  
PerimeterRectangle:  
section .text  
push rbp  
mov rbp, rsp  
  
    ; В rdi адрес прямоугольника  
mov eax, [rdi]  
add eax, [rdi+4]  
shl eax, 1  
cvtsi2sd    xmm0, eax  
  
leave  
ret
```



## *NASM. Главный модуль. Фрагмент проверки прямоугольника*

; Вычисление периметра прямоугольника

```
mov     rdi, rect
call    PerimeterRectangle
mov     [intP], eax
movsd   [p], xmm0
```

; Вывод периметра прямоугольника

```
mov     rdi, pfmt
mov     rsi, rax
movsd   xmm0, [p]    ; third argument for printf
mov     rax, 1       ; only integer registers involved
call    printf       ; call the function
```

```
rect    dd          5,7    ; прямоугольник с двумя сторонами
```

## *NASM. Разработка фрагмента кода вычисления периметра треугольника*

```
;-----  
; double PerimeterTriangle(void *t) {  
;     return (double)(*((int*)t) + *((int*)(t+intSize)) + *((int*)(t+2*intSize)));  
; }  
global PerimeterTriangle  
PerimeterTriangle:  
section .text  
push rbp  
mov rbp, rsp  
  
    ; В rdi адрес треугольника  
    mov eax, [rdi]  
    add eax, [rdi+4]  
    add eax, [rdi+8]  
    cvttsi2sd    xmm0, eax  
  
leave  
ret
```

## *NASM. Главный модуль. Фрагмент проверки треугольника*

; Вычисление периметра треугольника

```
mov    rdi, trian
call   PerimeterTriangle
mov    [intP], eax
movsd  [p], xmm0
```

; Вывод периметра треугольника

```
mov    rdi, pfmt
mov    rsi, rax
movsd  xmm0, [p]    ; third argument for printf
mov    rax, 1        ; only integer registers involved
call   printf       ; call the function
```

```
trian  dd    3,4,5    ; три стороны треугольника
```

## NASM. Разработка фрагмента кода вычисления периметра фигуры

```
global PerimeterShape
PerimeterShape:
section .text
push rbp
mov rbp, rsp

; В rdi адрес фигуры
mov eax, [rdi]
cmp eax, [RECTANGLE]
je rectPerimeter
cmp eax, [TRIANGLE]
je trianPerimeter
xor eax, eax
cvtsi2sd xmm0, eax
jmp return

rectPerimeter:
; Вычисление периметра прямоугольника
add rdi, 4
call PerimeterRectangle
jmp return

trianPerimeter:
; Вычисление периметра треугольника
add rdi, 4
call PerimeterTriangle

return:
leave
ret
```

```
-----
; Вычисление периметра фигуры
;double PerimeterShape(void *s) {
;   int k = *((int*)s);
;   if(k == RECTANGLE) {
;       return PerimeterRectangle(s+intSize);
;   }
;   else if(k == TRIANGLE) {
;       return PerimeterTriangle(s+intSize);
;   }
;   else {
;       return 0.0;
;   }
;}
```

## Инструкции перехода и значения флагов

Инструкция	Флаги	Описание	Использование
je	ZF=1	Переход, если равно	Знаковые, беззнаковые
jne	ZF=0	Переход, если не равно	Знаковые, беззнаковые
jg	$((SF \text{ XOR } OF) \text{ OR } ZF) = 0$	Переход, если больше	Знаковые
jge	$(SF \text{ XOR } OF) = 0$	Переход, если больше или равно	Знаковые
jl	$(SF \text{ XOR } OF) = 1$	Переход, если меньше	Знаковые
jle	$(SF \text{ XOR } OF) \text{ OR } ZF = 1$	Переход, если меньше или равно	Знаковые
ja	$(CF \text{ OR } ZF) = 0$	Переход по условию «больше»	Беззнаковые
jae	CF=0	Переход по условию «больше или равно»	Беззнаковые
jb	CF=1	Переход по условию «меньше»	Беззнаковые
jbe	$(CF \text{ OR } ZF) = 1$	Переход по условию «меньше или равно»	Беззнаковые

# NASM. Главный модуль. Фрагмент проверки фигуры

```
; Вычисление периметра фигуры-прямоугольника
mov     rdi, shapeRect
call    PerimeterShape
mov     [intP], eax
movsd   [p], xmm0

; Вывод периметра фигуры-прямоугольника
mov     rdi, pfmt
mov     rsi, [intP]
movsd   xmm0, [p] ; third argument for printf
mov     rax, 1 ; only integer registers involved
call    printf ; call the function
```

```
; Вычисление периметра фигуры-треугольника
mov     rdi, shapeTrian
call    PerimeterShape
mov     [intP], eax
movsd   [p], xmm0
```

```
; Вывод периметра фигуры-треугольника
mov     rdi, pfmt
mov     rsi, [intP]
movsd   xmm0, [p] ; third argument for printf
mov     rax, 1 ; only integer registers involved
call    printf ; call the function
```

```
shapeRect dd 1
rect dd 5,7 ; прямоугольник с двумя сторонами
empty dd 0 ; пустое слово в фигуре для прямоугольника
shapeTrian dd 2
trian dd 3,4,5 ; три стороны треугольника
```

# NASM. Разработка фрагмента кода вычисления периметра

```
global PerimeterSumContainer
PerimeterSumContainer:
section .data
    .sum    dq    0.0
section .text
push rbp
mov rbp, rsp

; В rdi адрес начала контейнера
mov rbx, rsi          ; число фигур
xor rcx, rcx          ; счетчик фигур
movsd xmm1, [.sum]    ; перенос накопителя суммы в регистр 1
.loop:
cmp rcx, rbx          ; проверка на окончание цикла
jge .return           ; Перебрали все фигуры

mov r10, rdi           ; сохранение начала фигуры
call PerimeterShape   ; Получение периметра первой фигуры
addsd xmm1, xmm0       ; накопление суммы
inc rcx                ; индекс следующей фигуры
add r10, 16            ; адрес следующей фигуры
mov rdi, r10           ; восстановление для передачи параметра
jmp .loop

.return:
movsd xmm0, xmm1
leave
ret
```

```
-----
; // Вычисление суммы периметров всех фигур в контейнере
; double PerimeterSumContainer(void *c, int len) {
;     double sum = 0.0;
;     void *tmp = c;
;     for(int i = 0; i < len; i++) {
;         sum += PerimeterShape(tmp);
;         tmp = tmp + shapeSize;
;     }
;     return sum;
; }
; }
```

## *NASM. Главный модуль. Фрагмент проверки 2-х фигур*

; Вычисление общего периметра 2-х фигур

```
mov    rdi, shapeRect
mov    rsi, 2
call   PerimeterSumContainer
movsd  [p], xmm0
```

; Вывод суммарного периметра 2-х фигур

```
mov    rdi, pcfmt
movsd  xmm0, [p] ; third argument for printf
mov    rax, 1 ; only integer registers involved
call   printf ; call the function
```

```
shapeRect dd 1
rect      dd 5,7 ; прямоугольник с двумя сторонами
empty     dd 0 ; пустое слово в фигуре для прямоугольника
shapeTrian dd 2
trian     dd 3,4,5 ; три стороны треугольника
```



## *NASM. Главный модуль. Фрагмент проверки нескольких фигур*

```
; Вывод периметра нескольких фигур
mov     rdi, pcfmt
movsd   xmm0, [p]    ; third argument for printf
mov     rax, 1        ; only integer registers involved
call    printf       ; call the function

pop     rbp
mov     rax, 60
mov     rdi, 0
syscall

; Массив фигур
cont    dd    1,2,3,0, 2,3,4,5, 1,2,8,0, 2,2,2,2, 2,5,7,4, 1,5,6,0
len     dd    6
```

## *Makefile для сборки многофайлового проекта*

```
# makefile for perimeter.asm
task: perimeter.o main.o
»      gcc -o perimeter main.o perimeter.o -no-pie
main.o: main.asm
»      nasm -f elf64 -g -F dwarf main.asm -l main.lst
perimeter.o: perimeter.asm
»      nasm -f elf64 -g -F dwarf perimeter.asm -l perimeter.lst
```

# C+NASM. Совместное использование

```
-----  
; perimeter.asm - единица компиляции, вбирающая функции вычисления периметра  
-----  
  
extern RECTANGLE  
extern TRIANGLE  
  
-----  
; Вычисление периметра прямоугольника  
;double PerimeterRectangle(void *r) {  
;    return 2.0 * (*((int*)r) + (*((int*)(r+intSize)));  
;}  
global PerimeterRectangle  
PerimeterRectangle:  
section .text  
push rbp  
mov rbp, rsp  
  
    ; В rdi адрес прямоугольника  
    mov eax, [rdi]  
    add eax, [rdi+4]  
    shl eax, 1  
    cvtsi2sd    xmm0, eax  
  
leave  
ret  
  
-----  
; double PerimeterTriangle(void *t) {  
;    return (double)*((int*)t) + (*((int*)(t+intSize)) + (*((int*)(t+2*intSize)));  
;}  
global PerimeterTriangle  
PerimeterTriangle:  
section .text  
push rbp  
mov rbp, rsp  
  
    ; В rdi адрес треугольника  
    mov eax, [rdi]  
    add eax, [rdi+4]  
    add eax, [rdi+8]  
    cvtsi2sd    xmm0, eax  
  
leave  
ret
```

```
-----  
; Вычисление периметра фигуры  
;double PerimeterShape(void *s) {  
;    int k = *((int*)s);  
;    if(k == RECTANGLE) {  
;        return PerimeterRectangle(s+intSize);  
;    }  
;    else if(k == TRIANGLE) {  
;        return PerimeterTriangle(s+intSize);  
;    }  
;    else {  
;        return 0.0;  
;    }  
;}  
global PerimeterShape  
PerimeterShape:  
section .text  
push rbp  
mov rbp, rsp  
  
    ; В rdi адрес фигуры  
    mov eax, [rdi]  
    cmp eax, [RECTANGLE]  
    je rectPerimeter  
    cmp eax, [TRIANGLE]  
    je trianPerimeter  
    xor eax, eax  
    cvtsi2sd    xmm0, eax  
    jmp    return  
rectPerimeter:  
    ; Вычисление периметра прямоугольника  
    add    rdi, 4  
    call    PerimeterRectangle  
    jmp    return  
trianPerimeter:  
    ; Вычисление периметра треугольника  
    add    rdi, 4  
    call    PerimeterTriangle  
return:  
leave  
ret
```

# C+NASM. Совместное использование

```
-----  
; // Вычисление суммы периметров всех фигур в контейнере  
; double PerimeterSumContainer(void *c, int len) {  
;     double sum = 0.0;  
;     void *tmp = c;  
;     for(int i = 0; i < len; i++) {  
;         sum += PerimeterShape(tmp);  
;         tmp = tmp + shapeSize;  
;     }  
;     return sum;  
; }  
global PerimeterSumContainer  
PerimeterSumContainer:  
section .data  
    .sum    dq    0.0  
section .text  
push rbp  
mov rbp, rsp  
  
    ; В rdi адрес начала контейнера  
mov rbx, rsi        ; число фигур  
xor rcx, rcx        ; счетчик фигур  
movsd xmm1, [.sum]  ; перенос накопителя суммы в регистр 1  
.loop:  
    cmp rcx, rbx        ; проверка на окончание цикла  
    jge .return        ; Перебрали все фигуры  
  
    mov r10, rdi        ; сохранение начала фигуры  
    call PerimeterShape ; Получение периметра первой фигуры  
    addsd xmm1, xmm0    ; накопление суммы  
    inc rcx            ; индекс следующей фигуры  
    add r10, 16        ; адрес следующей фигуры  
    mov rdi, r10        ; восстановление для передачи параметра  
    jmp .loop  
.return:  
    movsd xmm0, xmm1  
leave  
ret
```

## C+NASM. Makefile

```
# makefile for perimeter.asm
task: main.c input.c inrnd.c output.c perimeter.o
»      gcc -g -o task main.c input.c inrnd.c output.c perimeter.o -no-pie
perimeter.o: perimeter.asm
»      nasm -f elf64 -g -F dwarf perimeter.asm -l perimeter.lst
```

## Выполнение программ

10000 элементов	C & АДД	C & void	C & NASM
1	0.000298	0.000306	0.00018
2	0.000294	0.000297	0.000184
3	0.000296	0.000296	0.000182

# Добавление модуля вывода данных в поток

```
1 ; output.asm - вывод данных в поток
2 extern printf
3 extern fprintf
4
5 extern PerimeterRectangle
6 extern PerimeterTriangle
7
8 extern RECTANGLE
9 extern TRIANGLE
10
11 ;-----
12 ;// Вывод параметров прямоугольника в файл
13 ;void OutRectangle(void *r, FILE *ofst) {
14 ;   fprintf(ofst, "It is Rectangle: x = %d, y = %d. Perimeter = %g\n",
15 ;           *((int*)r), *((int*)(r+intSize)), PerimeterRectangle(r));
16 ;}
17 global OutRectangle
18 OutRectangle:
19 section .data
20     .outfmt db "It is Rectangle: x = %d, y = %d. Perimeter = %g",10,0
21     ;.outfmt db "It is Rectangle",10,0
22 section .bss
23     .prect resq 1
24     .FILE resq 1 ; временное хранение указателя на файл
25     .p resq 1 ; вычисленный периметр прямоугольника
26 section .text
27 push rbp
28 mov rbp, rsp
29
30 ; Сохранены принятых аргументов
31 mov [.prect], rdi ; сохраняется адрес прямоугольника
32 mov [.FILE], rsi ; сохраняется указатель на файл
33
34 ; Вычисление периметра прямоугольника (адрес уже в rdi)
35 call PerimeterRectangle
36 movsd [.p], xmm0 ; сохранение (может лишнее) периметра
37
38 ; Вывод информации о прямоугольнике в файл
39 mov rdi, [.FILE]
40 mov rsi, .outfmt ; Формат - 2-й аргумент
41 mov rax, [.prect] ; адрес прямоугольника
42 mov edx, [rax] ; x
43 mov ecx, [rax+4] ; y
44 movsd xmm0, [.p]
45 mov rax, 1 ; есть числа с плавающей точкой
46 call fprintf
47
48 leave
49 ret
50
```

```
51 ;-----
52 ; // Вывод параметров треугольника в файл
53 ; void OutTriangle(void *t, FILE *ofst) {
54 ;   fprintf(ofst, "It is Triangle: a = %d, b = %d, c = %d. Perimeter = %g\n",
55 ;           *((int*)t), *((int*)(t+intSize)), *((int*)(t+2*intSize)),
56 ;           PerimeterTriangle(t));
57 ; }
58 global OutTriangle
59 OutTriangle:
60 section .data
61     .outfmt db "It is Triangle: a = %d, b = %d, c = %d. Perimeter = %g",10,0
62 section .bss
63     .ptrian resq 1
64     .FILE resq 1 ; временное хранение указателя на файл
65     .p resq 1 ; вычисленный периметр треугольника
66 section .text
67 push rbp
68 mov rbp, rsp
69
70 ; Сохранены принятых аргументов
71 mov [.ptrian], rdi ; сохраняется адрес треугольника
72 mov [.FILE], rsi ; сохраняется указатель на файл
73
74 ; Вычисление периметра треугольника (адрес уже в rdi)
75 call PerimeterTriangle
76 movsd [.p], xmm0 ; сохранение (может лишнее) периметра
77
78 ; Вывод информации о треугольнике в файл
79 mov rdi, [.FILE]
80 mov rsi, .outfmt ; Формат - 2-й аргумент
81 mov rax, [.ptrian] ; адрес треугольника
82 mov edx, [rax] ; x
83 mov ecx, [rax+4] ; b
84 mov r8, [rax+8] ; c
85 movsd xmm0, [.p]
86 mov rax, 1 ; есть числа с плавающей точкой
87 call fprintf
88
89 leave
90 ret
91
```

# Добавление модуля вывода данных в поток

```
92 ;-----  
93 ; // Вывод параметров текущей фигуры в файл  
94 ; void OutShape(void *s, FILE *ofst) {  
95 ;     int k = *((int*)s);  
96 ;     if(k == RECTANGLE) {  
97 ;         OutRectangle(s+intSize, ofst);  
98 ;     }  
99 ;     else if(k == TRIANGLE) {  
100 ;         OutTriangle(s+intSize, ofst);  
101 ;     }  
102 ;     else {  
103 ;         fprintf(ofst, "Incorrect figure!\n");  
104 ;     }  
105 ; }  
106 global OutShape  
107 OutShape:  
108 section .data  
109     .erShape db "Incorrect figure!",10,0  
110 section .text  
111 push rbp  
112 mov rbp, rsp  
113  
114 ; В rdi адрес фигуры  
115 mov eax, [rdi]  
116 cmp eax, [RECTANGLE]  
117 je .rectOut  
118 cmp eax, [TRIANGLE]  
119 je .trianOut  
120 mov rdi, .erShape  
121 mov rax, 0  
122 call fprintf  
123 jmp .return  
124 .rectOut:  
125 ; Вывод прямоугольника  
126 add rdi, 4  
127 call OutRectangle  
128 jmp .return  
129 .trianOut:  
130 ; Вывод треугольника  
131 add rdi, 4  
132 call OutTriangle  
133 .return:  
134 leave  
135 ret  
136
```

```
137 ;-----  
138 ; // Вывод содержимого контейнера в файл  
139 ; void OutContainer(void *c, int len, FILE *ofst) {  
140 ;     void *tmp = c;  
141 ;     fprintf(ofst, "Container contains %d elements.\n", len);  
142 ;     for(int i = 0; i < len; i++) {  
143 ;         fprintf(ofst, "%d: ", i);  
144 ;         OutShape(tmp, ofst);  
145 ;         tmp = tmp + shapeSize;  
146 ;     }  
147 ; }  
148 global OutContainer  
149 OutContainer:  
150 section .text  
151 push rbp  
152 mov rbp, rsp  
153  
154 ; В rdi адрес начала контейнера  
155 mov rbx, rsi ; число фигур  
156 xor rcx, rcx ; счетчик фигур = 0  
157 mov rsi, rdx ; перенос указателя на файл  
158 .loop:  
159 cmp rcx, rbx ; проверка на окончание цикла  
160 jge .return ; Перебрали все фигуры  
161  
162 push rbx  
163 push rcx  
164 push rdi  
165 push rsi  
166 call OutShape ; Получение периметра первой фигуры  
167 pop rsi  
168 pop rdi  
169 pop rcx  
170 pop rbx  
171 inc rcx ; индекс следующей фигуры  
172 add rdi, 16 ; адрес следующей фигуры  
173 jmp .loop  
174 .return:  
175 leave  
176 ret
```



## Добавление модуля ввода данных из потока

```
1 ; input.asm - ввод данных из потока
2 extern printf
3 extern fscanf
4
5 extern RECTANGLE
6 extern TRIANGLE
7
8 ;-----
9 ; // Ввод параметров прямоугольника из файла
10 ; void InRectangle(void *r, FILE *ifst) {
11 ;     fscanf(ifst, "%d%d", (int*)r, (int*)(r+intSize));
12 ; }
13 global InRectangle
14 InRectangle:
15 section .data
16     .infmt db "%d%d",0
17 section .bss
18     .FILE resq 1 ; временное хранение указателя на файл
19     .prect resq 1 ; адрес прямоугольника
20 section .text
21 push rbp
22 mov rbp, rsp
23
24 ; Сохранение принятых аргументов
25 mov [.prect], rdi ; сохраняется адрес прямоугольника
26 mov [.FILE], rsi ; сохраняется указатель на файл
27
28 ; Ввод прямоугольника из файла
29 mov rdi, [.FILE]
30 mov rsi, .infmt ; Формат - 1-й аргумент
31 mov rdx, [.prect] ; &x
32 mov rcx, [.prect]
33 add rcx, 4 ; &y = &x + 4
34 mov rax, 0 ; нет чисел с плавающей точкой
35 call fscanf
36
37 leave
38 ret
39
```

```
40 ; // Ввод параметров треугольника из файла
41 ; void InTriangle(void *t, FILE *ifst) {
42 ;     fscanf(ifst, "%d%d%d", (int*)t,
43 ;         (int*)(t+intSize), (int*)(t+2*intSize));
44 ; }
45 global InTriangle
46 InTriangle:
47 section .data
48     .infmt db "%d%d%d",0
49 section .bss
50     .FILE resq 1 ; временное хранение указателя на файл
51     .trian resq 1 ; адрес треугольника
52 section .text
53 push rbp
54 mov rbp, rsp
55
56 ; Сохранение принятых аргументов
57 mov [.trian], rdi ; сохраняется адрес треугольника
58 mov [.FILE], rsi ; сохраняется указатель на файл
59
60 ; Ввод треугольника из файла
61 mov rdi, [.FILE]
62 mov rsi, .infmt ; Формат - 1-й аргумент
63 mov rdx, [.trian] ; &a
64 mov rcx, [.trian]
65 add rcx, 4 ; &b = &a + 4
66 mov r8, [.trian]
67 add r8, 8 ; &c = &x + 8
68 mov rax, 0 ; нет чисел с плавающей точкой
69 call fscanf
70
71 leave
72 ret
```

# Добавление модуля ввода данных из потока

```
74 ; // Ввод параметров обобщенной фигуры из файла
75 ; int InShape(void *s, FILE *ifst) {
76 ;     int k;
77 ;     fscanf(ifst, "%d", &k);
78 ;     switch(k) {
79 ;     case 1:
80 ;         *((int*)s) = RECTANGLE;
81 ;         InRectangle(s+intSize, ifst);
82 ;         return 1;
83 ;     case 2:
84 ;         *((int*)s) = TRIANGLE;
85 ;         InTriangle(s+intSize, ifst);
86 ;         return 1;
87 ;     default:
88 ;         return 0;
89 ;     }
90 ; }
91 global InShape
92 InShape:
93 section .data
94     .tagFormat    db     "%d",0
95     .tagOutFmt    db     "Tag is: %d",10,0
96 section .bss
97     .FILE         resq   1    ; временное хранение указателя на файл
98     .pshape       resq   1    ; адрес фигуры
99     .shapeTag     resd   1    ; признак фигуры
100 section .text
101 push rbp
102 mov rbp, rsp
103
104 ; Сохранение принятых аргументов
105 mov     [.pshape], rdi    ; сохраняется адрес фигуры
106 mov     [.FILE], rsi     ; сохраняется указатель на файл
107
108 ; чтение признака фигуры и его обработка
109 mov     rdi, [.FILE]
110 mov     rsi, .tagFormat
111 mov     rdx, [.pshape]   ; адрес начала фигуры (ее признак)
112 xor     rax, rax        ; нет чисел с плавающей точкой
113 call   fscanf
114
```

```
115     mov rcx, [.pshape]    ; загрузка адреса начала фигуры
116     mov eax, [rcx]       ; и получение прочитанного признака
117     cmp eax, [RECTANGLE]
118     je .rectIn
119     cmp eax, [TRIANGLE]
120     je .trianIn
121     xor eax, eax        ; Некорректный признак - обнуление кода возврата
122     jmp .return
123 .rectIn:
124     ; Ввод прямоугольника
125     mov rdi, [.pshape]
126     add rdi, 4
127     mov rsi, [.FILE]
128     call InRectangle
129     mov rax, 1 ; Код возврата - true
130     jmp .return
131 .trianIn:
132     ; Ввод треугольника
133     mov rdi, [.pshape]
134     add rdi, 4
135     mov rsi, [.FILE]
136     call InTriangle
137     mov rax, 1 ; Код возврата - true
138 .return:
139
140 leave
141 ret
142
```

# Добавление модуля ввода данных из потока

```
143 ; // Ввод содержимого контейнера из указанного файла
144 ; void InContainer(void *c, int *len, FILE *ifst) {
145 ;     void *tmp = c;
146 ;     while(!feof(ifst)) {
147 ;         if(InShape(tmp, ifst)) {
148 ;             tmp = tmp + shapeSize;
149 ;             (*len)++;
150 ;         }
151 ;     }
152 ; }
153 global InContainer
154 InContainer:
155 section .bss
156     .pcont resq 1 ; адрес контейнера
157     .plen resq 1 ; адрес для сохранения числа введенных элементов
158     .FILE resq 1 ; указатель на файл
159 section .text
160 push rbp
161 mov rbp, rsp
162
163     mov [.pcont], rdi ; сохраняется указатель на контейнер
164     mov [.plen], rsi ; сохраняется указатель на длину
165     mov [.FILE], rdx ; сохраняется указатель на файл
166     ; В rdi адрес начала контейнера
167     xor rbx, rbx ; число фигур = 0
168     mov rsi, rdx ; перенос указателя на файл
169 .loop:
170     ; сохранение рабочих регистров
171     push rdi
172     push rbx
173
174     mov rsi, [.FILE]
175     mov rax, 0 ; нет чисел с плавающей точкой
176     call InShape ; ввод фигуры
177     cmp rax, 0 ; проверка успешности ввода
178     jle .return ; выход, если признак меньше или равен 0
179
180     pop rbx
181     inc rbx
182
183     pop rdi
184     add rdi, 16 ; адрес следующей фигуры
185
186     jmp .loop
187 .return:
188     mov rax, [.plen] ; перенос указателя на длину
189     mov [rax], ebx ; занесение длины
190 leave
191 ret
```

# Добавление модуля генерации случайных данных

```
1 ; input.asm - ввод данных из потока
2 extern printf
3 extern fscanf
4
5 extern RECTANGLE
6 extern TRIANGLE
7
8 ;-----
9 ; // Ввод параметров прямоугольника из файла
10 ; void InRectangle(void *r, FILE *ifst) {
11 ;     fscanf(ifst, "%d%d", (int*)r, (int*)(r+intSize));
12 ; }
13 global InRectangle
14 InRectangle:
15 section .data
16     .infmt db "%d%d",0
17 section .bss
18     .FILE resq 1 ; временное хранение указателя на файл
19     .prect resq 1 ; адрес прямоугольника
20 section .text
21 push rbp
22 mov rbp, rsp
23
24 ; Сохранение принятых аргументов
25 mov [.prect], rdi ; сохраняется адрес прямоугольника
26 mov [.FILE], rsi ; сохраняется указатель на файл
27
28 ; Ввод прямоугольника из файла
29 mov rdi, [.FILE]
30 mov rsi, .infmt ; Формат - 1-й аргумент
31 mov rdx, [.prect] ; &x
32 mov rcx, [.prect]
33 add rcx, 4 ; &y = &x + 4
34 mov rax, 0 ; нет чисел с плавающей точкой
35 call fscanf
36
37 leave
38 ret
39
```

```
40 ; // Ввод параметров треугольника из файла
41 ; void InTriangle(void *t, FILE *ifst) {
42 ;     fscanf(ifst, "%d%d%d", (int*)t,
43 ;         (int*)(t+intSize), (int*)(t+2*intSize));
44 ; }
45 global InTriangle
46 InTriangle:
47 section .data
48     .infmt db "%d%d%d",0
49 section .bss
50     .FILE resq 1 ; временное хранение указателя на файл
51     .trian resq 1 ; адрес треугольника
52 section .text
53 push rbp
54 mov rbp, rsp
55
56 ; Сохранение принятых аргументов
57 mov [.trian], rdi ; сохраняется адрес треугольника
58 mov [.FILE], rsi ; сохраняется указатель на файл
59
60 ; Ввод треугольника из файла
61 mov rdi, [.FILE]
62 mov rsi, .infmt ; Формат - 1-й аргумент
63 mov rdx, [.trian] ; &a
64 mov rcx, [.trian]
65 add rcx, 4 ; &b = &a + 4
66 mov r8, [.trian]
67 add r8, 8 ; &c = &x + 8
68 mov rax, 0 ; нет чисел с плавающей точкой
69 call fscanf
70
71 leave
72 ret
```

# Добавление модуля генерации случайных данных

```
1 ; inrnd.asm - порождение случайных данных
2 extern printf
3 extern rand
4
5 extern RECTANGLE
6 extern TRIANGLE
7
8
9 ;-----
10 ; // rnd.c - содержит генератор случайных чисел в диапазоне от 1 до 20
11 ; int Random() {
12 ;     return rand() % 20 + 1;
13 ; }
14 global Random
15 Random:
16 section .data
17     .i20 dq 20
18     .rndNumFmt db "Random number = %d",10,0
19 section .text
20 push rbp
21 mov rbp, rsp
22
23     xor    rax, rax ;
24     call  rand ; запуск генератора случайных чисел
25     xor    rdx, rdx ; обнуление перед делением
26     idiv  qword[.i20] ; (/%) -> остаток в rdx
27     mov   rax, rdx
28     inc   rax ; должно сформироваться случайное число
29
30 leave
31 ret
```

```
33 ;-----
34 ;// Случайный ввод параметров прямоугольника
35 ;void InRndRectangle(void *r) {
36 ;    int x = Random();
37 ;    *((int*)r) = x;
38 ;    int y = Random();
39 ;    *((int*)(r+intSize)) = y;
40 ;//     printf("    Rectangle %d %d\n", *((int*)r), *((int*)r+1));
41 ;}
42 global InRndRectangle
43 InRndRectangle:
44 section .bss
45     .prect resq 1 ; адрес прямоугольника
46 section .text
47 push rbp
48 mov rbp, rsp
49
50 ; В rdi адрес прямоугольника
51 mov [.prect], rdi
52 ; Генерация сторон прямоугольника
53 call Random
54 mov rbx, [.prect]
55 mov [rbx], eax
56 call Random
57 mov rbx, [.prect]
58 mov [rbx+4], eax
59
60 leave
61 ret
```

# Добавление модуля генерации случайных данных

```
63 ;-----  
64 ;// Случайный ввод параметров треугольника  
65 ;void InRndTriangle(void *) {  
66     ;int a, b, c;  
67     ;a = *((int*)t) = Random();  
68     ;b = *((int*)(t+intSize)) = Random();  
69     ;do {  
70         ;c = *((int*)(t+2*intSize)) = Random();  
71     ;} while((c >= a + b) || (a >= c + b) || (b >= c + a));  
72 ;//     printf("    Triangle %d %d %d\n", *((int*)t), *((int*)t+1), *((int*)t+2));  
73 ;}  
74 global InRndTriangle  
75 InRndTriangle:  
76 section .bss  
77     .ptrian resq 1    ; адрес треугольника  
78 section .text  
79 push rbp  
80 mov rbp, rsp  
81  
82     ; В rdi адрес треугольника  
83 mov     [.ptrian], rdi  
84     ; Генерация сторон треугольника  
85 call    Random  
86 mov     rbx, [.ptrian]  
87 mov     [rbx], eax  
88 call    Random  
89 mov     rbx, [.ptrian]  
90 mov     [rbx+4], eax  
91 .repeat:  
92 call    Random  
93 mov     rbx, [.ptrian]  
94 mov     [rbx+8], eax    ; c  
95     ; Проверка корректности сторон  
96 mov     ecx, [rbx]    ; a  
97 mov     edx, [rbx+4]    ; b  
98 mov     ebx, eax    ; c  
99 add     eax, ecx    ; c+a  
100 cmp     edx, eax    ; b - (c+a)  
101 jge     .repeat  
102 mov     eax, ecx  
103 add     eax, edx    ; a+b  
104 cmp     ebx, eax    ; c - (a+b)  
105 jge     .repeat  
106 mov     eax, edx  
107 add     eax, ebx    ; b+c  
108 cmp     ecx, eax    ; a - (b+c)  
109 jge     .repeat  
110  
111 leave  
112 ret
```

```
114 ;-----  
115 ;// Случайный ввод обобщенной фигуры  
116 ;int InRndShape(void *) {  
117     ;int k = rand() % 2 + 1;  
118     ;switch(k) {  
119         ;case 1:  
120             ;*((int*)s) = RECTANGLE;  
121             ;InRndRectangle(s+intSize);  
122             ;return 1;  
123         ;case 2:  
124             ;*((int*)s) = TRIANGLE;  
125             ;InRndTriangle(s+intSize);  
126             ;return 1;  
127         ;default:  
128             ;return 0;  
129     ;}  
130 ;}  
131 global InRndShape  
132 InRndShape:  
133 section .data  
134     .rndNumFmt db "Random number = %d",10,0  
135 section .bss  
136     .pshape resq 1    ; адрес фигуры  
137     .key resd 1    ; ключ  
138 section .text  
139 push rbp  
140 mov rbp, rsp  
141  
142     ; В rdi адрес фигуры  
143 mov     [.pshape], rdi  
144  
145     ; Формирование признака фигуры  
146 xor     rax, rax    ;  
147 call    rand    ; запуск генератора случайных чисел  
148 and     eax, 1    ; очистка результата кроме младшего разряда (0 или 1)  
149 inc     eax    ; формирование признака фигуры (1 или 2)  
150  
151 mov     rdi, [.pshape]  
152 mov     [rdi], eax    ; запись ключа в фигуру  
153 cmp     eax, [RECTANGLE]  
154 je     .rectInRnd  
155 cmp     eax, [TRIANGLE]  
156 je     .trianInRnd  
157 xor     eax, eax    ; код возврата = 0  
158 jmp     .return  
159 .rectInRnd:  
160     ; Генерация прямоугольника  
161 add     rdi, 4  
162 call    InRndRectangle  
163 mov     eax, 1    ; код возврата = 1  
164 jmp     .return  
165 .trianInRnd:  
166     ; Генерация треугольника  
167 add     rdi, 4  
168 call    InRndTriangle  
169 mov     eax, 1    ; код возврата = 1  
170 .return:  
171 leave  
172 ret
```

# Добавление модуля генерации случайных данных

```
174 ;-----  
175 ;// Случайный ввод содержимого контейнера  
176 ;void InRndContainer(void *c, int *len, int size) {  
177     ;void *tmp = c;  
178     ;while(*len < size) {  
179         ;if(InRndShape(tmp)) {  
180             ;tmp = tmp + shapeSize;  
181             ;(*len)++;  
182         ;}  
183     ;}  
184 ;}  
185 global InRndContainer  
186 InRndContainer:  
187 section .bss  
188     .pcont resq 1 ; адрес контейнера  
189     .plen resq 1 ; адрес для сохранения числа введенных элементов  
190     .psize resd 1 ; число порождаемых элементов  
191 section .text  
192 push rbp  
193 mov rbp, rsp  
194  
195     mov [.pcont], rdi ; сохраняется указатель на контейнер  
196     mov [.plen], rsi ; сохраняется указатель на длину  
197     mov [.psize], edx ; сохраняется число порождаемых элементов  
198     ; В rdi адрес начала контейнера  
199     xor ebx, ebx ; число фигур = 0  
200 .loop:  
201     cmp ebx, edx  
202     jge .return  
203     ; сохранение рабочих регистров  
204     push rdi  
205     push rbx  
206     push rdx  
207  
208     call InRndShape ; ввод фигуры  
209     cmp rax, 0 ; проверка успешности ввода  
210     jle .return ; выход, если признак меньше или равен 0  
211  
212     pop rdx  
213     pop rbx  
214     inc rbx  
215  
216     pop rdi  
217     add rdi, 16 ; адрес следующей фигуры  
218  
219     jmp .loop  
220 .return:  
221     mov rax, [.plen] ; перенос указателя на длину  
222     mov [rax], ebx ; занесение длины  
223 Leave  
224 ret
```

# Реализация главной функции

```
1 ;-----  
2 ; main.asm - содержит главную функцию,  
3 ; обеспечивающую простое тестирование  
4 ;-----  
5 ; main.asm  
6  
7 global RECTANGLE  
8 global TRIANGLE  
9  
10 %include "macros.mac"  
11  
12 section .data  
13     RECTANGLE    dd    1  
14     TRIANGLE    dd    2  
15     oneDouble   dq    1.0  
16     erMsg1      db    "Incorrect number of arguments = %d: ",10,0  
17     rndGen      db    "-n",0  
18     fileGen     db    "-f",0  
19     errMsg1     db    "incorrect command line!", 10," Waited:",10  
20     .           db    "    command -f infile outfile01 outfile02",10," Or:",10  
21     .           db    "    command -n number outfile01 outfile02",10,0  
22     errMsg2     db    "incorrect qualifier value!", 10," Waited:",10  
23     .           db    "    command -f infile outfile01 outfile02",10," Or:",10  
24     .           db    "    command -n number outfile01 outfile02",10,0  
25     len         dd    0           ; Количество элементов в массиве  
26 section .bss  
27     argc        resd   1  
28     num         resd   1  
29     sum         resq   1  
30     start       resq   1           ; начало отсчета времени  
31     delta       resq   1           ; интервал отсчета времени  
32     startTime   resq   2           ; начало отсчета времени  
33     endTime     resq   2           ; конец отсчета времени  
34     deltaTime   resq   2           ; интервал отсчета времени  
35     ifst        resq   1           ; указатель на файл, открываемый файл для чтения фигур  
36     ofst1       resq   1           ; указатель на файл, открываемый файл для записи контейнера  
37     ofst2       resq   1           ; указатель на файл, открываемый файл для записи периметра  
38     cont        resb  160000     ; Массив используемый для хранения данных  
39
```



# Макроопределения

```
1 ; macros.mac - файл с макроопределениями
2
3 extern fopen
4 extern fscanf
5 extern fclose
6 extern fprintf
7 extern printf
8 extern stdout
9 extern time
10 extern srand
11 extern strcmp
12 extern atoi
13 extern clock
14
15 extern InContainer
16 extern InRndContainer
17 extern OutContainer
18 extern PerimeterSumContainer
19
20 ;-----
21 ; Вывод строки символов из буфера
22 %macro PrintStrBuf 2
23     mov rdi, %2
24     mov rsi, %1
25     xor rax, rax
26     call fprintf
27 %endmacro
28
29 ;-----
30 ; Вывод строки, передаваемой непосредственно макросу
31 %macro PrintStr 2
32     section .data
33         %%arg1 db %1,0 ; first argument
34     section .text ; the printf arguments
35         mov rdi, %2
36         mov rsi, %%arg1
37         mov rax, 0 ; no floating point
38         call fprintf
39 %endmacro
```

# Макроопределения

```
41 ;-----  
42 ; Вывод строки, передаваемой непосредственно макросу  
43 ; с переводом на следующую строку  
44 %macro PrintStrLn 2  
45     section .data  
46         %%arg1 db %1,10,0 ; first argument  
47     section .text ; the printf arguments  
48         mov rdi, %2  
49         mov rsi, %%arg1  
50         mov rax, 0 ; no floating point  
51         call fprintf  
52 %endmacro  
53  
54 ;-----  
55 ; Вывод целого числа  
56 %macro PrintInt 2  
57     section .data  
58         %%arg1 db "%d",0 ; first argument  
59     section .text ; the printf arguments  
60         mov rdi, %2  
61         mov rsi, %%arg1  
62         mov rdx, %1  
63         mov rax, 0 ; no floating point  
64         call fprintf  
65 %endmacro  
66  
67 ;-----  
68 ; Вывод 64-разрядного беззнакового целого числа  
69 %macro PrintLLUns 2  
70     section .data  
71         %%arg1 db "%llu",0 ; first argument  
72     section .text ; the printf arguments  
73         mov rdi, %2  
74         mov rsi, %%arg1  
75         mov rdx, %1  
76         mov rax, 0 ; no floating point  
77         call fprintf  
78 %endmacro  
79
```

```
80 ;-----  
81 ; Вывод действительного числа двойной точности  
82 %macro PrintDouble 2  
83     section .data  
84         %%arg1 db "%g",0 ; first argument  
85     section .text ; the printf arguments  
86         mov rdi, %2  
87         mov rsi, %%arg1  
88         movsd xmm0, %1  
89         mov rax, 1 ; no floating point  
90         call fprintf  
91 %endmacro  
92  
93 ;-----  
94 ; Вывод содержимого контейнера  
95 %macro PrintContainer 3  
96     mov rdi, %1  
97     mov esi, %2  
98     mov rdx, %3  
99     mov rax, 0 ; нет чисел с плавающей точкой  
100     call OutContainer  
101 %endmacro  
102  
103 ;-----  
104 ; Вычисление суммы периметров фигур в контейнере  
105 %macro ContainerSum 3  
106     mov rdi, %1  
107     mov rsi, %2  
108     call PerimeterSumContainer  
109     movsd %3, xmm0  
110 %endmacro
```

## Макроопределения

```
112 ;-----  
113 ; Открытие файла для чтения, записи  
114 %macro FileOpen 3  
115     section .data  
116         %%rw db %2,0 ; признак доступа  
117     section .text ; the printf arguments  
118         mov rdi, %1 ; адрес строки открываемого файла  
119         mov rsi, %%rw ; строка определяющая чтение-запись. Задается явно  
120         ; Или аналог:  
121         ;lea rsi, [%%rw] ; строка определяющая чтение-запись. Задается явно  
122         mov rax, 0 ; нет чисел с плавающей точкой  
123         call fopen  
124         mov [%3], rax  
125 %endmacro  
126  
127 ;-----  
128 ; Закрытие открытого файла  
129 %macro FileClose 1  
130     mov rdi, %1 ; передача указателя на закрываемый файл  
131     mov rax, 0 ; нет чисел с плавающей точкой  
132     call fclose  
133 %endmacro
```

## Реализация главной функции

```
40 section .text
41     global main
42 main:
43     push rbp
44     mov rbp, rsp
45
46     mov dword [argc], edi ;rdi contains number of arguments
47     mov r12, rdi ;rdi contains number of arguments
48     mov r13, rsi ;rsi contains the address to the array of arguments
49
50 .printArguments:
51     PrintStrLn "The command and arguments:", [stdout]
52     mov rbx, 0
53 .printLoop:
54     PrintStrBuf qword [r13+rbx*8], [stdout]
55     PrintStr 10, [stdout]
56     inc rbx
57     cmp rbx, r12
58     jl .printLoop
59
60     cmp r12, 5 ; проверка количества аргументов
61     je .next1
62     PrintStrBuf errorMessage1, [stdout]
63     jmp .return
64 .next1:
65     PrintStrLn "Start", [stdout]
```

## Реализация главной функции

```
66 ; Проверка второго аргумента
67 mov rdi, rndGen ; признак для сравнения
68 mov rsi, [r13+8] ; второй аргумент командной строки
69 mov rcx, 3 ; сравнение, включая ноль (обе строки завершились)
70 cld ; обнуление флага направления (инкремент)
71 repe cmpsb
72 ;;call strcmp
73 ;;cmp rax, 0 ; строки равны "-n"
74 je .next2
75 mov rdi, fileGen
76 mov rsi, [r13+8] ; второй аргумент командной строки
77 ;;mov rcx, 3 ; сравнение, включая ноль (обе строки завершились)
78 ;;cld ; обнуление флага направления (инкремент)
79 ;;repe cmpsb
80 call strcmp
81 cmp rax, 0 ; строки равны "-f"
82 je .next3
83 PrintStrBuf errMessage2, [stdout]
84 jmp .return
```

## Реализация главной функции

```
85 .next2:
86     ; Генерация случайных фигур
87     mov rdi, [r13+16]
88     call atoi
89     mov [num], eax
90     PrintInt [num], [stdout]
91     PrintStrLn "", [stdout]
92     mov eax, [num]
93     cmp eax, 1
94     jl .fall1
95     cmp eax, 10000
96     jg .fall1
97     ; Начальная установка генератора случайных чисел
98     xor    rdi, rdi
99     xor    rax, rax
100    call   time
101    mov    rdi, rax
102    xor    rax, rax
103    call   srand
104    ; Заполнение контейнера случайными фигурами
105    mov    rdi, cont    ; передача адреса контейнера
106    mov    rsi, len     ; передача адреса для длины
107    mov    edx, [num]  ; передача количества порождаемых фигур
108    call   InRndContainer
109    jmp   .task2
110
```

## Реализация главной функции

```
111 .next3:
112     ; Получение фигур из файла
113     FileOpen [r13+16], "r", ifst
114     ; Заполнение контейнера фигурами из файла
115     mov     rdi, cont           ; адрес контейнера
116     mov     rsi, len           ; адрес для установки числа элементов
117     mov     rdx, [ifst]        ; указатель на файл
118     xor     rax, rax
119     call    InContainer        ; ввод данных в контейнер
120     FileClose [ifst]
121
122 .task2:
123     ; Вывод содержимого контейнера
124     PrintStrLn "Filled container:", [stdout]
125     PrintContainer cont, [len], [stdout]
126
127     FileOpen [r13+24], "w", ofst1
128     PrintStrLn "Filled container:", [ofst1]
129     PrintContainer cont, [len], [ofst1]
130     FileClose [ofst1]
```

## Реализация главной функции

```
132 ; Вычисление времени старта
133 mov rax, 228 ; 228 is system call for sys_clock_gettime
134 xor edi, edi ; 0 for system clock (preferred over "mov rdi, 0")
135 lea rsi, [startTime]
136 syscall ; [time] contains number of seconds
137 ; ; ; ; ; [time + 8] contains number of nanoseconds
138
139 ContainerSum cont, [len], [sum]
140
141 ; Вычисление времени завершения
142 mov rax, 228 ; 228 is system call for sys_clock_gettime
143 xor edi, edi ; 0 for system clock (preferred over "mov rdi, 0")
144 lea rsi, [endTime]
145 syscall ; [time] contains number of seconds
146 ; ; ; ; ; [time + 8] contains number of nanoseconds
147
148 ; Получение времени работы
149 mov rax, [endTime]
150 sub rax, [startTime]
151 mov rbx, [endTime+8]
152 mov rcx, [startTime+8]
153 cmp rbx, rcx
154 jge .subNanoOnly
155 ; иначе занимаем секунду
156 dec rax
157 add rbx, 1000000000
158 .subNanoOnly:
159 sub rbx, [startTime+8]
160 mov [deltaTime], rax
161 mov [deltaTime+8], rbx
```



## Реализация главной функции

```
163     ; Вывод периметра нескольких фигур
164     PrintStr "Perimeter sum = ", [stdout]
165     PrintDouble [sum], [stdout]
166     PrintStr ". Calculaton time = ", [stdout]
167     PrintLLUns [deltaTime], [stdout]
168     PrintStr " sec, ", [stdout]
169     PrintLLUns [deltaTime+8], [stdout]
170     PrintStr " nsec", [stdout]
171     PrintStr 10, [stdout]
172
173     FileOpen [r13+32], "w", ofst2
174     PrintStr "Perimeter sum = ", [ofst2]
175     PrintDouble [sum], [ofst2]
176     PrintStr ". Calculaton time = ", [ofst2]
177     PrintLLUns [deltaTime], [ofst2]
178     PrintStr " sec, ", [ofst2]
179     PrintLLUns [deltaTime+8], [ofst2]
180     PrintStr " nsec", [ofst2]
181     PrintStr 10, [ofst2]
182     FileClose [ofst2]
183
184     PrintStrLn "Stop", [stdout]
185     jmp .return
186 .fall1:
187     PrintStr "incorrect numer of figures = ", [stdout]
188     PrintInt [num], [stdout]
189     PrintStrLn ". Set 0 < number <= 10000", [stdout]
190 .return:
191     leave
192     ret
```

## Makefile

```
1 # makefile for task.asm
2 task: main.o input.o inrnd.o output.o perimeter.o
3     gcc -g -o task main.o input.o inrnd.o output.o perimeter.o -no-pie
4 main.o: main.asm macros.mac
5     nasm -f elf64 -g -F dwarf main.asm -l main.lst
6 input.o: input.asm
7     nasm -f elf64 -g -F dwarf input.asm -l input.lst
8 inrnd.o: inrnd.asm
9     nasm -f elf64 -g -F dwarf inrnd.asm -l inrnd.lst
10 output.o: output.asm
11     nasm -f elf64 -g -F dwarf output.asm -l output.lst
12 perimeter.o: perimeter.asm
13     nasm -f elf64 -g -F dwarf perimeter.asm -l perimeter.lst
```

## Пример выполнения

```
$ make
nasm -f elf64 -g -F dwarf main.asm -l main.lst
nasm -f elf64 -g -F dwarf input.asm -l input.lst
nasm -f elf64 -g -F dwarf inrnd.asm -l inrnd.lst
nasm -f elf64 -g -F dwarf output.asm -l output.lst
nasm -f elf64 -g -F dwarf perimeter.asm -l perimeter.lst
gcc -g -o task main.o input.o inrnd.o output.o perimeter.o -no-pie
$ ./task -n 5 1 2
The command and arguments:
./task
-n
5
1
2
Start
5
Filled container:
0: It is Triangle: a = 10, b = 9, c = 17. Perimeter = 36
1: It is Triangle: a = 13, b = 12, c = 17. Perimeter = 42
2: It is Triangle: a = 14, b = 15, c = 2. Perimeter = 31
3: It is Triangle: a = 19, b = 18, c = 15. Perimeter = 52
4: It is Rectangle: x = 19, y = 9. Perimeter = 56
Perimeter sum = 217. Calculaton time = 0 sec, 1500 nsec
Stop
$ ./task -f tests/test01.txt 1 2
The command and arguments:
./task
-f
tests/test01.txt
1
2
Start
Filled container:
0: It is Rectangle: x = 3, y = 4. Perimeter = 14
1: It is Triangle: a = 1, b = 1, c = 1. Perimeter = 3
2: It is Rectangle: x = 30, y = 40. Perimeter = 140
3: It is Triangle: a = 2, b = 2, c = 1. Perimeter = 5
4: It is Rectangle: x = 13, y = 14. Perimeter = 54
5: It is Triangle: a = 10, b = 10, c = 10. Perimeter = 30
6: It is Rectangle: x = 330, y = 49. Perimeter = 758
7: It is Triangle: a = 3, b = 4, c = 5. Perimeter = 12
Perimeter sum = 1016. Calculaton time = 0 sec, 1857 nsec
Stop
[al@parsys 11-asm]$
```

## Обработка строк символов. Вычисление длины строки

```
65 ; FUNCTIONS =====
66
67 ;-----
68 ; Вычисление длины строки с учетом конечного нулевого символа
69 strlen0:
70 ; Адрес сообщения уже загружен в rdi
71 mov ecx, -1 ; ecx должен быть < 0
72 xor al, al ; конечный символ = 0
73 cld ; направление обхода от начала к концу
74 repne scasb ; while(msg[rDI]) != al) {rdi++, rcx--}
75 neg ecx
76 sub ecx, 1 ; ecx = length(msg)
77 mov eax, ecx
78 ret
79
80 ;-----
81 ; Вычисление длины строки как в языке C
82 strlen:
83 ; Адрес сообщения уже загружен в rdi
84 mov ecx, -1 ; ecx должен быть < 0
85 xor al, al ; конечный символ = 0
86 cld ; направление обхода от начала к концу
87 repne scasb ; while(msg[rDI]) != al) {rdi++, rcx--}
88 neg ecx
89 sub ecx, 2 ; ecx = length(msg)
90 mov eax, ecx
91 ret
```

# Обработка строк символов. Вычисление длины строки

```
1 ; test-strlen.asm - вычисление длины строки
2 %include "macros.mac"
3
4 section .data
5     NL      equ      10
6
7     msg     db      "Hello World!", 10,0
8     info    db      "Message: ",0
9     emptyMsg db      0
10    fDecimal db      "%d",10,0
11    len     dd      0
12
13 section .bss
14    strBuf   resb    256
15
16 section .text
17     global main
18 ;-----
19 main:
20     push    rbp
21     mov     rbp, rsp
22
23     PrintStrBuf info, [stdout]
24     PrintStrBuf msg, [stdout]
25
26     mov     rdi, msg
27     call   strlen0
28     mov     [len], eax
29     PrintStr "msg length with zero symbol = ", [stdout]
30     PrintInt [len], [stdout]
31     PrintStr NL, [stdout]
32
```

```
33     mov     rdi, msg
34     call   strlen
35     mov     [len], eax
36     PrintStr "msg length as C-strlen = ", [stdout]
37     PrintInt [len], [stdout]
38     PrintStr NL, [stdout]
39
40     PrintStrBuf info, [stdout]
41     PrintStrBuf emptyMsg, [stdout]
42     PrintStr NL, [stdout]
43
44     mov     rdi, emptyMsg
45     call   strlen0
46     mov     [len], eax
47     PrintStr "emptyMsg length with zero symbol = ", [stdout]
48     PrintInt [len], [stdout]
49     PrintStr NL, [stdout]
50
51     mov     rdi, emptyMsg
52     call   strlen
53     mov     [len], eax
54     PrintStr "emptyMsg length as C-strlen = ", [stdout]
55     PrintInt [len], [stdout]
56     PrintStr NL, [stdout]
57
58     PrintStrLn "Bye!", [stdout]
59
60     pop     rbp
61     mov     rax, 60
62     mov     rdi, 0
63     syscall
64
```

## Обработка строк символов. Вычисление длины строки

```
Message: Hello World!  
msg length with zero symbol = 14  
msg length as C-strlen = 13  
Message:  
emptyMsg length with zero symbol = 1  
emptyMsg length as C-strlen = 0  
Bye!
```

## Обработка строк символов. Копирование строки

```
95 ;-----  
96 ; Копирование строки как в языке C  
97 strcpy:  
98     ; Адрес приемника уже загружен в rdi  
99     ; Адрес источника уже загружен в rsi  
100    cld                ; направление обхода от начала к концу  
101    xor al, al         ; конечный символ = 0  
102    xor ecx, ecx       ; счетчик символов = 0  
103    .loopstrcpy:  
104    cmp al, [rsi]  
105    je .endloopstrcpy  
106    movsb              ; dst[] = src  
107    inc ecx            ; ecx++  
108    jmp .loopstrcpy  
109    .endloopstrcpy:  
110    mov [edi], al  
111    mov eax, ecx  
112    inc eax  
113    ret  
114  
115 ;-----  
116 ; Копирование строки с использованием strlen0 и rep  
117 strcpy2:  
118     ; Адрес приемника уже загружен в rdi  
119    mov r10, rdi       ; перенос во временное хранилище  
120     ; Адрес источника уже загружен в rsi  
121    mov r11, rsi       ; перенос во временное хранилище  
122  
123    mov rdi, rsi  
124    call strlen0      ; вычисление длины строки с нулем  
125  
126    mov ecx, eax  
127    mov rdi, r10  
128    mov rsi, r11  
129    cld                ; направление обхода от начала к концу  
130    rep movsb  
131    ret
```

## Обработка строк символов. Копирование строки

```
1 ; test-strcpy.asm - копирование строк
2 %include "macros.mac"
3
4 section .data
5     NL      equ      10
6
7     msg     db      "Hello World!", 10,0
8     info    db      "Message: ",0
9     emptyMsg db      0
10    fDecimal db      "%d",10,0
11    len     dd      0
12
13 section .bss
14    strBuf  resb    256
15
16 section .text
17     global main
18 ;-----
19 main:
20     push   rbp
21     mov    rbp, rsp
22
23     PrintStrBuf info, [stdout]
24     PrintStrBuf msg, [stdout]
25
26     mov    rdi, msg
27     call   strlen@
28     mov    [len], eax
29     PrintStr "msg length with zero symbol = ", [stdout]
30     PrintInt [len], [stdout]
31     PrintStr NL, [stdout]
32
```

```
33     mov    rdi, info
34     call   strlen@
35     mov    [len], eax
36     PrintStr "msg length with zero symbol = ", [stdout]
37     PrintInt [len], [stdout]
38     PrintStr NL, [stdout]
39
40     mov    rdi, strBuf
41     mov    rsi, msg
42     call   strcpy
43     mov    [len], eax
44     PrintStr "strcpy: strBuf = ", [stdout]
45     PrintStrBuf strBuf, [stdout]
46     PrintStr "strBuf len = ", [stdout]
47     PrintInt [len], [stdout]
48     PrintStr NL, [stdout]
49
50     mov    rdi, strBuf
51     mov    rsi, info
52     call   strcpy2
53     mov    [len], eax
54     PrintStr "strcpy2: strBuf = ", [stdout]
55     PrintStrBuf strBuf, [stdout]
56     PrintStr NL, [stdout]
57     PrintStr "strBuf len = ", [stdout]
58     PrintInt [len], [stdout]
59     PrintStr NL, [stdout]
60
61     PrintStrLn "Bye!", [stdout]
62     pop    rbp
63     mov    rax, 60
64     mov    rdi, 0
65     syscall
```



## Обработка строк символов. Копирование строки

```
Message: Hello World!  
msg length with zero symbol = 14  
msg length with zero symbol = 10  
strcpy: strBuf = Hello World!  
strBuf len = 14  
strcpy2: strBuf = Message:  
strBuf len = 10  
Bye!
```

## Обработка строк символов. Проверка на равенство

```
135 ;-----  
136 ; Сравнение двух строк на равенство  
137 streq:  
138     ; Адрес приемника уже загружен в rdi  
139     ; Адрес источника уже загружен в rsi  
140     cld                ; направление обхода от начала к концу  
141     xor eax, eax      ;  
142     .loopStrEq:  
143     cmp al, [rsi]  
144     je .endSrc  
145     cmp al, [rdi]  
146     je .isNe  
147     cmps  
148     je .loopStrEq  
149     jmp .isNe  
150     .endSrc:  
151     cmp al, [edi]  
152     je .isEq  
153     jmp .isNe  
154     .isEq:  
155     mov eax, 1        ; eax != 0 - true  
156     ret  
157     .isNe:  
158     ;xor eax, eax  
159     ret
```

# Обработка строк символов. Проверка на равенство

```
1 ; test-streq.asm - сравнение строк на равенство
2 %include "macros.mac"
3
4 section .data
5     NL      equ      10
6
7     msg     db      "Hello World!", 10,0
8     info    db      "Message: ",0
9     emptyMsg db      0
10    fDecimal db      "%d",10,0
11    len     dd      0
12    flag    dd      0
13
14 section .bss
15    strBuf  resb    256
16
17 section .text
18    global main
19 ;-----
20 main:
21    push    rbp
22    mov     rbp, rsp
23
24    PrintStrBuf info, [stdout]
25    PrintStr    NL, [stdout]
26    PrintStrBuf msg, [stdout]
27
28    mov     rdi, info
29    mov     rsi, msg
30    call   streq      ; != (false)
31    mov     [flag], eax
32    PrintStr    "Compare flag = ", [stdout]
33    PrintInt   [flag], [stdout]
34    PrintStr    NL, [stdout]
35
```

```
36    mov     rdi, msg
37    mov     rsi, msg
38    call   streq      ; == (true)
39    mov     [flag], eax
40    PrintStr    "Compare flag = ", [stdout]
41    PrintInt   [flag], [stdout]
42    PrintStr    NL, [stdout]
43
44    mov     rdi, strBuf
45    mov     rsi, msg
46    call   strcpy
47    mov     rdi, msg
48    mov     rsi, strBuf
49    call   streq      ; == (true)
50    mov     [flag], eax
51    PrintStr    "Compare flag = ", [stdout]
52    PrintInt   [flag], [stdout]
53    PrintStr    NL, [stdout]
54
55    mov     rdi, info
56    mov     rsi, strBuf
57    call   streq      ; != (false)
58    mov     [flag], eax
59    PrintStr    "Compare flag = ", [stdout]
60    PrintInt   [flag], [stdout]
61    PrintStr    NL, [stdout]
62
63    PrintStrLn "Bye!", [stdout]
64    pop     rbp
65    mov     rax, 60
66    mov     rdi, 0
67    syscall
```

## Обработка строк символов. Проверка на равенство

```
Message:  
Hello World!  
Compare flag = 0  
Compare flag = 1  
Compare flag = 1  
Compare flag = 0  
Bye!
```

## Обработка строк символов. Аналог strcmp языка C

```
169 ;-----  
170 ; Сравнение двух строк по аналогии с strcmp языка C  
171 strcmp:  
172     ; Адрес приемника уже загружен в rdi  
173     ; Адрес источника уже загружен в rsi  
174     cld                ; направление обхода от начала к концу  
175     xor eax, eax      ;  
176     .loopStrEq:  
177     cmp al, [rsi]  
178     je .endSrc  
179     cmp al, [rdi]  
180     je .isGt  
181     cmpsb  
182     je .loopStrEq  
183     jl .isLt  
184     jmp .isGt  
185     .endSrc:  
186     cmp al, [edi]  
187     je .isEq  
188     jmp .isLt  
189     .isEq:  
190     mov eax, 0        ; [rsi] == [rdi]  
191     ret  
192     .isLt:  
193     mov eax, -1       ; [rsi] < [rdi]  
194     ret  
195     .isGt:  
196     mov eax, 1        ; [rsi] > [rdi]  
197     ret
```

# Обработка строк символов. Аналог strcmp языка C

```
1 ; test-strcmp.asm - сравнение строк в стиле C-strcmp
2 %include "macros.mac"
3
4 section .data
5     NL      equ      10
6
7     msg     db      "Hello World!", 10,0
8     info    db      "Message: ",0
9     emptyMsg db      0
10    fDecimal db      "%d",10,0
11    len     dd      0
12    flag    dd      0
13
14 section .bss
15    strBuf  resb    256
16
17 section .text
18     global main
19 ;-----
20 main:
21     push   rbp
22     mov    rbp, rsp
23
24     PrintStrBuf info, [stdout]
25     PrintStr  NL, [stdout]
26     PrintStrBuf msg, [stdout]
27
28     mov    rsi, msg
29     mov    rdi, info
30     call   strcmp ; < (-1)
31     mov    [flag], eax
32     PrintStr  "Compare flag = ", [stdout]
33     PrintInt  [flag], [stdout]
34     PrintStr  NL, [stdout]
35
```

```
36     mov    rsi, info
37     mov    rdi, msg
38     call   strcmp ; > (1)
39     mov    [flag], eax
40     PrintStr  "Compare flag = ", [stdout]
41     PrintInt  [flag], [stdout]
42     PrintStr  NL, [stdout]
43
44     mov    rsi, msg
45     mov    rdi, msg
46     call   strcmp ; == (0)
47     mov    [flag], eax
48     PrintStr  "Compare flag = ", [stdout]
49     PrintInt  [flag], [stdout]
50     PrintStr  NL, [stdout]
51
52     mov    rsi, msg
53     mov    rdi, strBuf
54     call   strcpy
55     mov    rdi, msg
56     mov    rsi, strBuf
57     call   strcmp ; == (0)
58     mov    [flag], eax
59     PrintStr  "Compare flag = ", [stdout]
60     PrintInt  [flag], [stdout]
61     PrintStr  NL, [stdout]
62
63     mov    rsi, info
64     mov    rdi, strBuf
65     call   strcmp ; < (-1)
66     mov    [flag], eax
67     PrintStr  "Compare flag = ", [stdout]
68     PrintInt  [flag], [stdout]
69     PrintStr  NL, [stdout]
70
71     PrintStrLn "Bye!", [stdout]
72     pop    rbp
73     mov    rax, 60
74     mov    rdi, 0
75     syscall
```

## Обработка строк символов. Аналог `strcmp` языка C

```
Message:  
Hello World!  
Compare flag = -1  
Compare flag = 1  
Compare flag = 0  
Compare flag = 0  
Compare flag = 1  
Bye!
```

# SoftCraft

разноликое программирование

ОТПРАВНАЯ ТОЧКА

ПРОЕКТИРОВАНИЕ

ПАРАДИГМЫ

СИСТЕМЫ  
ПРОГРАММИРОВАНИЯ

ТЕХНИКА  
КОДИРОВАНИЯ

ИСКУССТВЕННЫЙ  
ИНТЕЛЛЕКТ

ТЕОРИЯ

## Ассемблер x86-64 (AMD64). Базовая техника

[Начальная страница курса](#)

### Содержание лекции

1. Статически типизированное решение с использованием АДТ (язык C)
2. Бестиповое решение (язык C)
3. NASM. Пример пошаговой разработки модуля вычисления периметров и суммы периметров
4. Совместное использование C и NASM. Интеграция модуля вычисления периметра в программу на языке C

<http://softcraft.ru/edu/comparch/lect/09-IntelAsmBase/>