

子程序的实现细节

源代码 main.c

```
1  #include <stdio.h>
2
3  struct tuple5{
4      int a0;
5      int a1;
6      int a2;
7      int a3;
8      int a4;
9  };
10
11 int func1(int a, int b){
12     if(a > b){
13         int c;
14         c = a - b;
15         return c;
16     }else{
17         int d;
18         int c = 1;
19         d = b - a;
20         return d;
21     }
22     return 0;
23 }
24
25 struct tuple5 func2(int a, int b){
26     struct tuple5 t;
27     if(a > b){
28         t.a0 = t.a1 = t.a2 = t.a3 = t.a4 = a;
29     }else{
30         t.a0 = t.a1 = t.a2 = t.a3 = t.a4 = b;
31     }
32     return t;
33 }
34
35 struct tuple5 func3(int a, int b){
36     struct tuple5 t;
37     if(a < b){
38         t.a0 = t.a1 = t.a2 = t.a3 = t.a4 = a;
39     }else{
40         t.a0 = t.a1 = t.a2 = t.a3 = t.a4 = b;
41     }
42     return t;
43 }
44
45 int main(){
46     int a = 0;
47     a = func1(1, 2);
48     struct tuple5 t1;
49     t1 = func2(2, 3);
50     struct tuple5 t2;
```

```

51     t2 = func3(2, 3);
52     printf("%d\n", a);
53     printf("%d\n", t1.a0+t2.a0);
54     return 0;
55 }

```

使用 -O0 选项

```
1 | gcc -O0 -S main.c -o o0.s
```

```

1      .file   "main.c"
2      .text
3      .globl func1
4      .def    func1; .sc1    2; .type    32; .endef
5      .seh_proc func1
6  func1:
7      pushq   %rbp
8      .seh_pushreg   %rbp
9      movq    %rsp, %rbp
10     .seh_setframe  %rbp, 0
11     subq    $16, %rsp
12     .seh_stackalloc 16
13     .seh_endprologue
14     movl    %ecx, 16(%rbp)
15     movl    %edx, 24(%rbp)
16     movl    16(%rbp), %eax
17     cmpl    24(%rbp), %eax
18     jle     .L2
19     movl    16(%rbp), %eax
20     subl    24(%rbp), %eax
21     movl    %eax, -12(%rbp)
22     movl    -12(%rbp), %eax
23     jmp     .L3
24  .L2:
25     movl    $1, -4(%rbp)
26     movl    24(%rbp), %eax
27     subl    16(%rbp), %eax
28     movl    %eax, -8(%rbp)
29     movl    -8(%rbp), %eax
30  .L3:
31     addq    $16, %rsp
32     popq    %rbp
33     ret
34     .seh_endproc
35     .globl func2
36     .def    func2; .sc1    2; .type    32; .endef
37     .seh_proc func2
38  func2:
39     pushq   %rbp
40     .seh_pushreg   %rbp
41     movq    %rsp, %rbp
42     .seh_setframe  %rbp, 0
43     subq    $32, %rsp
44     .seh_stackalloc 32

```

```

45     .seh_endprologue
46     movq    %rcx, 16(%rbp)
47     movl    %edx, 24(%rbp)
48     movl    %r8d, 32(%rbp)
49     movl    24(%rbp), %eax
50     cmpl    32(%rbp), %eax
51     jle     .L5
52     movl    24(%rbp), %eax
53     movl    %eax, -16(%rbp)
54     movl    -16(%rbp), %eax
55     movl    %eax, -20(%rbp)
56     movl    -20(%rbp), %eax
57     movl    %eax, -24(%rbp)
58     movl    -24(%rbp), %eax
59     movl    %eax, -28(%rbp)
60     movl    -28(%rbp), %eax
61     movl    %eax, -32(%rbp)
62     jmp     .L6
63 .L5:
64     movl    32(%rbp), %eax
65     movl    %eax, -16(%rbp)
66     movl    -16(%rbp), %eax
67     movl    %eax, -20(%rbp)
68     movl    -20(%rbp), %eax
69     movl    %eax, -24(%rbp)
70     movl    -24(%rbp), %eax
71     movl    %eax, -28(%rbp)
72     movl    -28(%rbp), %eax
73     movl    %eax, -32(%rbp)
74 .L6:
75     movq    16(%rbp), %rcx
76     movq    -32(%rbp), %rax
77     movq    -24(%rbp), %rdx
78     movq    %rax, (%rcx)
79     movq    %rdx, 8(%rcx)
80     movl    -16(%rbp), %eax
81     movl    %eax, 16(%rcx)
82     movq    16(%rbp), %rax
83     addq    $32, %rsp
84     popq    %rbp
85     ret
86     .seh_endproc
87     .globl func3
88     .def     func3; .sc1    2; .type    32; .endef
89     .seh_proc func3
90 func3:
91     pushq   %rbp
92     .seh_pushreg    %rbp
93     movq    %rsp, %rbp
94     .seh_setframe   %rbp, 0
95     subq    $32, %rsp
96     .seh_stackalloc 32
97     .seh_endprologue
98     movq    %rcx, 16(%rbp)
99     movl    %edx, 24(%rbp)
100    movl    %r8d, 32(%rbp)
101    movl    24(%rbp), %eax
102    cmpl    32(%rbp), %eax

```

```

103     jge .L9
104     movl    24(%rbp), %eax
105     movl    %eax, -16(%rbp)
106     movl    -16(%rbp), %eax
107     movl    %eax, -20(%rbp)
108     movl    -20(%rbp), %eax
109     movl    %eax, -24(%rbp)
110     movl    -24(%rbp), %eax
111     movl    %eax, -28(%rbp)
112     movl    -28(%rbp), %eax
113     movl    %eax, -32(%rbp)
114     jmp     .L10
115 .L9:
116     movl    32(%rbp), %eax
117     movl    %eax, -16(%rbp)
118     movl    -16(%rbp), %eax
119     movl    %eax, -20(%rbp)
120     movl    -20(%rbp), %eax
121     movl    %eax, -24(%rbp)
122     movl    -24(%rbp), %eax
123     movl    %eax, -28(%rbp)
124     movl    -28(%rbp), %eax
125     movl    %eax, -32(%rbp)
126 .L10:
127     movq    16(%rbp), %rcx
128     movq    -32(%rbp), %rax
129     movq    -24(%rbp), %rdx
130     movq    %rax, (%rcx)
131     movq    %rdx, 8(%rcx)
132     movl    -16(%rbp), %eax
133     movl    %eax, 16(%rcx)
134     movq    16(%rbp), %rax
135     addq    $32, %rsp
136     popq    %rbp
137     ret
138     .seh_endproc
139     .def     __main; .scl    2; .type    32; .endef
140     .section .rdata,"dr"
141 .LC0:
142     .ascii  "%d\12\0"
143     .text
144     .globl  main
145     .def     main; .scl    2; .type    32; .endef
146     .seh_proc  main
147 main:
148     pushq   %rbp
149     .seh_pushreg    %rbp
150     movq     %rsp, %rbp
151     .seh_setframe   %rbp, 0
152     subq     $96, %rsp
153     .seh_stackalloc 96
154     .seh_endprologue
155     call     __main
156     movl     $0, -4(%rbp)
157     movl     $2, %edx
158     movl     $1, %ecx
159     call     func1
160     movl     %eax, -4(%rbp)

```

```

161     leaq    -32(%rbp), %rax
162     movl    $3, %r8d
163     movl    $2, %edx
164     movq    %rax, %rcx
165     call    func2
166     leaq    -64(%rbp), %rax
167     movl    $3, %r8d
168     movl    $2, %edx
169     movq    %rax, %rcx
170     call    func3
171     movl    -4(%rbp), %eax
172     movl    %eax, %edx
173     leaq    .LC0(%rip), %rcx
174     call    printf
175     movl    -32(%rbp), %edx
176     movl    -64(%rbp), %eax
177     addl    %edx, %eax
178     movl    %eax, %edx
179     leaq    .LC0(%rip), %rcx
180     call    printf
181     movl    $0, %eax
182     addq    $96, %rsp
183     popq    %rbp
184     ret
185     .seh_endproc
186     .ident   "GCC: (GNU) 10.2.0"
187     .def     printf; .sc1    2; .type    32; .endef

```

在函数 `func1()` 中有 `if` 语句, 可以看到, `if` 语句中定义的变量的空间是在 `if` 语句里分配的, 不同子句中的变量, 不管同名与否, 不会公用空间 (比如两个子句中的 `int c`, 一个在 `-12(%rbp)`, 一个在 `-4(%rbp)`)

```

1      cmpl    24(%rbp), %eax
2      jle    .L2
3      movl    16(%rbp), %eax
4      subl    24(%rbp), %eax
5      movl    %eax, -12(%rbp)
6      movl    -12(%rbp), %eax
7      jmp     .L3
8  .L2:
9      movl    $1, -4(%rbp)
10     movl    24(%rbp), %eax
11     subl    16(%rbp), %eax
12     movl    %eax, -8(%rbp)
13     movl    -8(%rbp), %eax
14  .L3:

```

函数 `func2()` 要返回一个比较大的结构体, 这里选择了直接对调用时的左值进行赋值 (先取了局部变量的地址, 然后装到 `rcx` 里, 在函数里对 `rcx` 所指向的空间进行赋值) 而没有用多余的地址保存返回值

```

1  .L6:
2      movq    16(%rbp), %rcx
3      movq    -32(%rbp), %rax
4      movq    -24(%rbp), %rdx
5      movq    %rax, (%rcx)
6      movq    %rdx, 8(%rcx)
7      movl    -16(%rbp), %eax
8      movl    %eax, 16(%rcx)
9      movq    16(%rbp), %rax
10     addq    $32, %rsp
11     popq    %rbp
12     ret

```

`main()` 函数要调用两个返回较大结构体的函数, 可以看到编译器对此没有什么特殊的处理

```

1  leaq    -32(%rbp), %rax
2  movl    $3, %r8d
3  movl    $2, %edx
4  movq    %rax, %rcx
5  call    func2
6  leaq    -64(%rbp), %rax
7  movl    $3, %r8d
8  movl    $2, %edx
9  movq    %rax, %rcx
10 call    func3

```

使用 `-O2` 选项

```
1 | gcc -O2 -S main.c -o o2.s
```

```

1  .file     "main.c"
2  .text
3  .p2align  4
4  .globl   func1
5  .def     func1; .sc1    2; .type   32; .endef
6  .seh_proc func1
7  func1:
8  .seh_endprologue
9  movl    %ecx, %r8d
10 movl    %edx, %eax
11 subl    %edx, %r8d
12 subl    %ecx, %eax
13 cmpl    %edx, %ecx
14 cmovg    %r8d, %eax
15 ret
16 .seh_endproc
17 .p2align  4
18 .globl   func2
19 .def     func2; .sc1    2; .type   32; .endef
20 .seh_proc func2
21 func2:
22 .seh_endprologue
23 cmpl    %r8d, %edx

```

```

24     movq    %rcx, %rax
25     cmovl   %r8d, %edx
26     movl    %edx, (%rcx)
27     movl    %edx, 4(%rcx)
28     movl    %edx, 8(%rcx)
29     movl    %edx, 12(%rcx)
30     movl    %edx, 16(%rcx)
31     ret
32     .seh_endproc
33     .p2align 4
34     .globl  func3
35     .def    func3; .sc1    2; .type    32; .endef
36     .seh_proc  func3
37 func3:
38     .seh_endprologue
39     cmpl    %r8d, %edx
40     movq    %rcx, %rax
41     cmovg   %r8d, %edx
42     movl    %edx, (%rcx)
43     movl    %edx, 4(%rcx)
44     movl    %edx, 8(%rcx)
45     movl    %edx, 12(%rcx)
46     movl    %edx, 16(%rcx)
47     ret
48     .seh_endproc
49     .def    __main; .sc1    2; .type    32; .endef
50     .section .rdata,"dr"
51 .LC0:
52     .ascii  "%d\12\0"
53     .section .text.startup,"x"
54     .p2align 4
55     .globl  main
56     .def    main; .sc1    2; .type    32; .endef
57     .seh_proc  main
58 main:
59     subq    $40, %rsp
60     .seh_stackalloc 40
61     .seh_endprologue
62     call    __main
63     movl    $1, %edx
64     leaq    .LC0(%rip), %rcx
65     call    printf
66     movl    $5, %edx
67     leaq    .LC0(%rip), %rcx
68     call    printf
69     xorl    %eax, %eax
70     addq    $40, %rsp
71     ret
72     .seh_endproc
73     .ident   "GCC: (GNU) 10.2.0"
74     .def     printf; .sc1    2; .type    32; .endef

```

在函数 `func1()` 中有 `if` 语句, 可以看到 `-o2` 中, 并没有给局部变量分配内存空间, 仅仅利用了寄存器

```

1 func1:
2     .seh_endprologue
3     movl    %ecx, %r8d
4     movl    %edx, %eax
5     subl    %edx, %r8d
6     subl    %ecx, %eax
7     cmpl    %edx, %ecx
8     cmovg   %r8d, %eax
9     ret

```

在函数 `func2()` 中要返回一个比较大的结构体, 这时处理的基本思路还是直接对调用者中要被赋值的局部变量的空间进行数据填充, 但是做了许多优化, 没有使用额外的空间, 也把指令精简了许多

```

1 func2:
2     .seh_endprologue
3     cmpl    %r8d, %edx
4     movq    %rcx, %rax
5     cmovl    %r8d, %edx
6     movl    %edx, (%rcx)
7     movl    %edx, 4(%rcx)
8     movl    %edx, 8(%rcx)
9     movl    %edx, 12(%rcx)
10    movl    %edx, 16(%rcx)
11    ret

```

`main()` 函数要调用两个返回较大结构体的函数, 可以看到编译器做了很大的优化

```

1 main:
2     subq    $40, %rsp
3     .seh_stackalloc 40
4     .seh_endprologue
5     call    __main
6     movl    $1, %edx
7     leaq    .LC0(%rip), %rcx
8     call    printf
9     movl    $5, %edx
10    leaq    .LC0(%rip), %rcx
11    call    printf
12    xorl    %eax, %eax
13    addq    $40, %rsp
14    ret

```

把 `-o0` 和 `-o2` 对比可以发现, `-o2` 的优化程度远远大于 `-o0`