# 1 题目

用C语言编译器编译有函数调用的代码，做O0和O2两个选项，查看两次汇编结果，分析两个实现细节：

# 2 程序与汇编

## 2.1 测试程序与编译器

**test.c**

```
#include <stdio.h>
struct test
{
    /* data */
    int data;
    int type;
    char id;
};

struct test max(int num1, int num2)
{
    if (num1 > num2)
    {
        int data = num1;
    }
    else
    {
        int data = num2;
    }
    struct test result;
    result.data = num1;
    result.type = 1;
    result.id = 'z';
    return result;
}

struct test min(int num1, int num2)
{
    if (num1 > num2)
    {
        int data = num1;
    }
    else
```

```
    {
        int data = num2;
    }
    struct test result;
    result.data = num2;
    result.type = 1;
    result.id = 'j';
    return result;
};

int main()
{
    int a = 1;
    int b = 2;
    struct test result;

    result = max(a, b);
    result = min(a, b);

    return 0;
}
```

**编译器**

X86-64 gcc 9.2

## 2.2 使用-O0编译

**汇编结果**

```
max(int, int):
        push    rbp
        mov     rbp, rsp
        mov     DWORD PTR [rbp-36], edi
        mov     DWORD PTR [rbp-40], esi
        mov     eax, DWORD PTR [rbp-36]
        cmp     eax, DWORD PTR [rbp-40]
        jle     .L2
        mov     eax, DWORD PTR [rbp-36]
        mov     DWORD PTR [rbp-8], eax
        jmp     .L3
.L2:
        mov     eax, DWORD PTR [rbp-40]
        mov     DWORD PTR [rbp-4], eax
.L3:
        mov     eax, DWORD PTR [rbp-36]
        mov     DWORD PTR [rbp-32], eax
        mov     DWORD PTR [rbp-28], 1
        mov     BYTE PTR [rbp-24], 122
```

```
        mov     rax, QWORD PTR [rbp-32]
        mov     QWORD PTR [rbp-20], rax
        mov     eax, DWORD PTR [rbp-24]
        mov     DWORD PTR [rbp-12], eax
        mov     rax, QWORD PTR [rbp-20]
        mov     ecx, DWORD PTR [rbp-12]
        mov     rdx, rcx
        pop     rbp
        ret
min(int, int):
        push    rbp
        mov     rbp, rsp
        mov     DWORD PTR [rbp-36], edi
        mov     DWORD PTR [rbp-40], esi
        mov     eax, DWORD PTR [rbp-36]
        cmp     eax, DWORD PTR [rbp-40]
        jle     .L6
        mov     eax, DWORD PTR [rbp-36]
        mov     DWORD PTR [rbp-8], eax
        jmp     .L7
.L6:
        mov     eax, DWORD PTR [rbp-40]
        mov     DWORD PTR [rbp-4], eax
.L7:
        mov     eax, DWORD PTR [rbp-40]
        mov     DWORD PTR [rbp-32], eax
        mov     DWORD PTR [rbp-28], 1
        mov     BYTE PTR [rbp-24], 106
        mov     rax, QWORD PTR [rbp-32]
        mov     QWORD PTR [rbp-20], rax
        mov     eax, DWORD PTR [rbp-24]
        mov     DWORD PTR [rbp-12], eax
        mov     rax, QWORD PTR [rbp-20]
        mov     ecx, DWORD PTR [rbp-12]
        mov     rdx, rcx
        pop     rbp
        ret
main:
        push    rbp
        mov     rbp, rsp
        sub     rsp, 32
        mov     DWORD PTR [rbp-4], 1
        mov     DWORD PTR [rbp-8], 2
        mov     edx, DWORD PTR [rbp-8]
        mov     eax, DWORD PTR [rbp-4]
        mov     esi, edx
        mov     edi, eax
        call    max(int, int)
        mov     QWORD PTR [rbp-20], rax
```

```
        mov     eax, DWORD PTR [rbp-12]
        and     eax, 0
        or      eax, edx
        mov     DWORD PTR [rbp-12], eax
        mov     edx, DWORD PTR [rbp-8]
        mov     eax, DWORD PTR [rbp-4]
        mov     esi, edx
        mov     edi, eax
        call    min(int, int)
        mov     QWORD PTR [rbp-20], rax
        mov     eax, DWORD PTR [rbp-12]
        and     eax, 0
        or      eax, edx
        mov     DWORD PTR [rbp-12], eax
        mov     eax, 0
        leave
        ret
```

## 2.2 使用-O2编译

**汇编结果**

```
max(int, int):
        mov     DWORD PTR [rsp-20], edi
        mov     DWORD PTR [rsp-16], 1
        mov     rax, QWORD PTR [rsp-20]
        mov     BYTE PTR [rsp-12], 122
        mov     edx, DWORD PTR [rsp-12]
        ret
min(int, int):
        mov     DWORD PTR [rsp-20], esi
        mov     DWORD PTR [rsp-16], 1
        mov     rax, QWORD PTR [rsp-20]
        mov     BYTE PTR [rsp-12], 106
        mov     edx, DWORD PTR [rsp-12]
        ret
main:
        xor     eax, eax
        ret
```

# 3 分析实现细节

| 寄存器 | 作用 |
| --- | --- |
| %rbp | 用作数据存储，调用子函数之间需要备份它 |
| %rsp | 栈指针寄存器，指向栈顶 |
| %eax | 加法乘法指令的缺省寄存器； 存放函数返回值 |
| %esi<br>%edi | 分别叫做"源/目标索引寄存器"(source/destination index),因为在很多字符串操作指令中, DS:ESI指向源串,而ES:EDI指向目标串 |
| %rax | 作为函数返回值使用 |

| 指令 | 作用 |
| --- | --- |
| movq | 完成8个字节的复制 |
| movl | 完成4个字节的复制 |
| cmpl | 将两个操作数相减，但计算结果并不保存，只是根据计算结果改变eflags寄存器中的标志位。如果两个操作数相等，则计算结果为0，eflags中的ZF位置1 |
| callq | 移到子函数 |
| movabsq | 将一个64位的值直接存到一个64位寄存器中 |

# 3.1 子句与变量定义

`函数中有更进一步的代码块，如if语句的子句，并且在这些子句中有变量定义时，这些变量的空间是在什么时候分配的，是在函数开始的时候还是进入子句的时候；不同的子句中的变量，同名与否，是否会共用空间；`

程序在if语句`if (num1 < num2)`的子句中定义一个变量`int data`

**1. -O0**

```
        mov     eax, DWORD PTR [rbp-36]
        cmp     eax, DWORD PTR [rbp-40]
        jle     .L2
```

实现if (num1 > num2)

```
        mov     eax, DWORD PTR [rbp-36]
        mov     DWORD PTR [rbp-8], eax
```

实现`int data = num1;`,if子句中的变量data分配的空间是DWORD PTR [rbp-8]，并且被赋值为num1。

```
        mov     eax, DWORD PTR [rbp-40]
        mov     DWORD PTR [rbp-4], eax
```

实现`int data = num2;`else子句中的变量data分配的空间是DWORD PTR [rbp-4]，并且被赋值为num2。

从上可以得出使用-O0选项编译时，函数有更进一步的代码块时，语句的子句有变量定义时，变量的空间在进入子句的时候分配；同名与否，都不会共用空间。

**2. -O2**

使用-O2选项省去了整个if-else结构，因为子句中的变量在后面没有被用到，也不需要返回。因此-O2优化了程序，没有为子句中的不必要的变量分配空间。

将max函数修改为

```
struct test max(int num1, int num2)
{
    struct test result;
    if (num1 > num2)
    {
        int data = num1+1;
        result.data = data;
    }
    else
    {
        int data = num2+1;
        result.data = data;
    }
    //result.data = num1;
    result.type = 1;
    result.id = 'z';
    return result;
}
```

使用-O2得到的编译结果为

```
max(int, int):
        lea     edx, [rdi+1]
        lea     eax, [rsi+1]
        cmp     edi, esi
        mov     DWORD PTR [rsp-16], 1
        cmovg   eax, edx
        mov     BYTE PTR [rsp-12], 122
        mov     edx, DWORD PTR [rsp-12]
        mov     DWORD PTR [rsp-20], eax
        mov     rax, QWORD PTR [rsp-20]
        ret
```

编译结果中的2-3行为程序第6行的`data`分配空间，没有为第11行的`data`分配空间。因此，使用-O2选项编译时，函数有更进一步的代码块时，语句的子句有变量定义时，变量的空间在函数开始的时候分配。

进一步将max函数修改为

```
struct test max(int num1, int num2)
{
    struct test result;
    if (num1 > num2)
    {
        int data = num1+1;
        result.data = data;
    }
    if(num1 = 2)
    {
        int data = num2+1;
        result.data = data;
    }
    //result.data = num1;
    result.type = 1;
    result.id = 'z';
    return result;
}
```

使用-O2得到的编译结果为

```
max(int, int):
        add     esi, 1
        mov     DWORD PTR [rsp-16], 1
        mov     DWORD PTR [rsp-20], esi
        mov     rax, QWORD PTR [rsp-20]
        mov     BYTE PTR [rsp-12], 122
        mov     edx, DWORD PTR [rsp-12]
        ret
```

只有第11行的变量`data`得到了分配空间。

子句中的变量同名与否，都不会共用空间。

## 3.2 返回较大结构

当函数返回一个较大的结构时，返回的数据是如何安排空间的。当一个函数调用两个这样的函数的时候，空间是如何安排的。

程序中有一个**struct test**的结构，是函数**max**和**min**的返回类型。

**1. -O0**

```
        mov     edx, DWORD PTR [rbp-8]
        mov     eax, DWORD PTR [rbp-4]
        mov     esi, edx
        mov     edi, eax
        call    max(int, int)
        mov     QWORD PTR [rbp-20], rax
        mov     eax, DWORD PTR [rbp-12]
        and     eax, 0
        or      eax, edx
        mov     DWORD PTR [rbp-12], eax
        mov     edx, DWORD PTR [rbp-8]
        mov     eax, DWORD PTR [rbp-4]
        mov     esi, edx
        mov     edi, eax
        call    min(int, int)
        mov     QWORD PTR [rbp-20], rax
        mov     eax, DWORD PTR [rbp-12]
        and     eax, 0
        or      eax, edx
        mov     DWORD PTR [rbp-12], eax
```

表示调用max和min函数得到返回的结果。函数的返回值被存储在一个连续的空间上。

修改main函数，将函数返回值写回两个不同的变量：

```
int main()
{
    int a = 1;
    int b = 2;
    struct test result, result2;

    result = max(a, b);
    result2 = min(a, b);

    return 0;
}
```

得到汇编结果

```
max(int, int):
        push    rbp
        mov     rbp, rsp
        mov     DWORD PTR [rbp-36], edi
        mov     DWORD PTR [rbp-40], esi
        mov     eax, DWORD PTR [rbp-36]
        cmp     eax, DWORD PTR [rbp-40]
        jle     .L2
        mov     eax, DWORD PTR [rbp-36]
```

```
        mov     DWORD PTR [rbp-8], eax
        jmp     .L3
.L2:
        mov     eax, DWORD PTR [rbp-40]
        mov     DWORD PTR [rbp-4], eax
.L3:
        mov     eax, DWORD PTR [rbp-36]
        mov     DWORD PTR [rbp-32], eax
        mov     DWORD PTR [rbp-28], 1
        mov     BYTE PTR [rbp-24], 122
        mov     rax, QWORD PTR [rbp-32]
        mov     QWORD PTR [rbp-20], rax
        mov     eax, DWORD PTR [rbp-24]
        mov     DWORD PTR [rbp-12], eax
        mov     rax, QWORD PTR [rbp-20]
        mov     ecx, DWORD PTR [rbp-12]
        mov     rdx, rcx
        pop     rbp
        ret
min(int, int):
        push    rbp
        mov     rbp, rsp
        mov     DWORD PTR [rbp-36], edi
        mov     DWORD PTR [rbp-40], esi
        mov     eax, DWORD PTR [rbp-36]
        cmp     eax, DWORD PTR [rbp-40]
        jle     .L6
        mov     eax, DWORD PTR [rbp-36]
        mov     DWORD PTR [rbp-8], eax
        jmp     .L7
.L6:
        mov     eax, DWORD PTR [rbp-40]
        mov     DWORD PTR [rbp-4], eax
.L7:
        mov     eax, DWORD PTR [rbp-40]
        mov     DWORD PTR [rbp-32], eax
        mov     DWORD PTR [rbp-28], 1
        mov     BYTE PTR [rbp-24], 106
        mov     rax, QWORD PTR [rbp-32]
        mov     QWORD PTR [rbp-20], rax
        mov     eax, DWORD PTR [rbp-24]
        mov     DWORD PTR [rbp-12], eax
        mov     rax, QWORD PTR [rbp-20]
        mov     ecx, DWORD PTR [rbp-12]
        mov     rdx, rcx
        pop     rbp
        ret
main:
        push    rbp
```

```
59          mov     rbp, rsp
60          sub     rsp, 32
61          mov     DWORD PTR [rbp-4], 1
62          mov     DWORD PTR [rbp-8], 2
63          mov     edx, DWORD PTR [rbp-8]
64          mov     eax, DWORD PTR [rbp-4]
65          mov     esi, edx
66          mov     edi, eax
67          call    max(int, int)
68          mov     QWORD PTR [rbp-20], rax
69          mov     eax, DWORD PTR [rbp-12]
70          and     eax, 0
71          or      eax, edx
72          mov     DWORD PTR [rbp-12], eax
73          mov     edx, DWORD PTR [rbp-8]
74          mov     eax, DWORD PTR [rbp-4]
75          mov     esi, edx
76          mov     edi, eax
77          call    min(int, int)
78          mov     QWORD PTR [rbp-32], rax
79          mov     eax, DWORD PTR [rbp-24]
80          and     eax, 0
81          or      eax, edx
82          mov     DWORD PTR [rbp-24], eax
83          mov     eax, 0
84          leave
85          ret
```

可以从63-82行看出，函数返回值从函数返回值寄存器中取出再写入一个新的空间，struct结构内部的数据得到了一片连续的空间。result和result1得到了不同的空间。

**2. -O2**

函数的返回值直接在函数中写入结果寄存器，没有在main函数中重新从函数返回值寄存器中取出写入一个新的结果寄存器。在连续调用的两个函数中，返回值写回的是同一个变量`result`，两个函数都将结果写入同一片寄存器空间。函数的返回值被存储在一个连续的空间上。

修改main函数，将函数返回值写回两个不同的变量：

```
int main()
{
    int a = 1;
    int b = 2;
    struct test result, result2;

    result = max(a, b);
    result2 = min(a, b);

    return 0;
}
```

得到汇编结果

```
max(int, int):
        mov     DWORD PTR [rsp-20], edi
        mov     DWORD PTR [rsp-16], 1
        mov     rax, QWORD PTR [rsp-20]
        mov     BYTE PTR [rsp-12], 122
        mov     edx, DWORD PTR [rsp-12]
        ret
min(int, int):
        mov     DWORD PTR [rsp-20], esi
        mov     DWORD PTR [rsp-16], 1
        mov     rax, QWORD PTR [rsp-20]
        mov     BYTE PTR [rsp-12], 106
        mov     edx, DWORD PTR [rsp-12]
        ret
main:
        xor     eax, eax
        ret
```

发现函数返回值仍然在函数中直接返回，不经过main函数，且写入相同的空间。