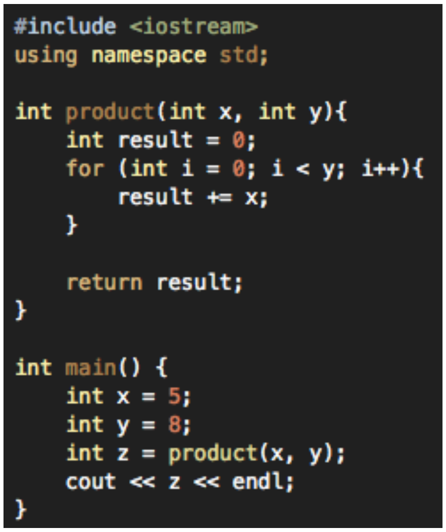
Liya Xu
lx2hy
Section 103
2014/4/8

In-Lab 9

Optimized Code

For the optimization comparison question, I wrote the following C++ code snippet. It uses a loop to calculate the product of x and y, which is defined in the main. The function is called in the main.



```
#include <iostream>
using namespace std;

int product(int x, int y){
    int result = 0;
    for (int i = 0; i < y; i++){
        result += x;
    }

    return result;
}

int main() {
    int x = 5;
    int y = 8;
    int z = product(x, y);
    cout << z << endl;
}
```

<u>Non-optimized code:</u>

```
_Z7productii:
.LFB966:
        .cfi_startproc
        push    ebp
        .cfi_def_cfa_offset 8
        .cfi_offset 5, -8
        mov     ebp, esp
        .cfi_def_cfa_register 5
        sub     esp, 16
        mov     DWORD PTR [ebp-8], 0
        mov     DWORD PTR [ebp-4], 0
        jmp     .L2
.L3:
        mov     eax, DWORD PTR [ebp+8]
        add     DWORD PTR [ebp-8], eax
        add     DWORD PTR [ebp-4], 1
.L2:
        mov     eax, DWORD PTR [ebp-4]
        cmp     eax, DWORD PTR [ebp+12]
        setl    al
        test    al, al
        jne     .L3
        mov     eax, DWORD PTR [ebp-8]
        leave
        .cfi_restore 5
        .cfi_def_cfa 4, 4
        ret
        .cfi_endproc
.LFE966:
        .size   _Z7productii, .-_Z7productii
        .globl  main
        .type   main, @function
main:
.LFB967:
        .cfi_startproc
        push    ebp
        .cfi_def_cfa_offset 8
        .cfi_offset 5, -8
        mov     ebp, esp
        .cfi_def_cfa_register 5
        and     esp, -16
        sub     esp, 32
        mov     DWORD PTR [esp+20], 5
        mov     DWORD PTR [esp+24], 8
        mov     eax, DWORD PTR [esp+24]
        mov     DWORD PTR [esp+4], eax
        mov     eax, DWORD PTR [esp+20]
        mov     DWORD PTR [esp], eax
        call    _Z7productii
        mov     DWORD PTR [esp+28], eax
        mov     eax, DWORD PTR [esp+28]
        mov     DWORD PTR [esp+4], eax
        mov     DWORD PTR [esp], OFFSET FLAT:_ZSt4cout
        call    _ZNSolsEi
        mov     DWORD PTR [esp+4], OFFSET
FLAT:_ZSt4endlIcSt11char_traitsIcEERSt13basic_ostreamIT_T0_ES6_
        mov     DWORD PTR [esp], eax
        call    _ZNSolsEPFRSoS_E
        mov     eax, 0
        leave
        .cfi_restore 5
        .cfi_def_cfa 4, 4
        ret
        .cfi_endproc
.LFE967:
        .size   main, .-main
        .type   _Z41__static_initialization_and_destruction_0ii, @function
```

Optimized code:

```
_Z7productii:
.LFB1006:
        .cfi_startproc
        mov     edx, DWORD PTR [esp+8]
        xor     eax, eax
        mov     ecx, DWORD PTR [esp+4]
        imul    ecx, edx
        test    edx, edx
        cmovg   eax, ecx
        ret
        .cfi_endproc
.LFE1006:
        .size   _Z7productii, .-_Z7productii
        .section.text.startup,"ax",@progbits
        .p2align 4,,15
        .globl  main
        .type   main, @function
main:
.LFB1007:
        .cfi_startproc
        push    ebp
        .cfi_def_cfa_offset 8
        .cfi_offset 5, -8
        mov     ebp, esp
        .cfi_def_cfa_register 5
        and     esp, -16
        sub     esp, 16
        mov     DWORD PTR [esp+4], 40
        mov     DWORD PTR [esp], OFFSET FLAT:_ZSt4cout
        call    _ZNSolsEi
        mov     DWORD PTR [esp], eax
        call    _ZSt4endlIcSt11char_traitsIcEERSt13basic_ostreamIT_T0_ES6_
        xor     eax, eax
        leave
        .cfi_restore 5
        .cfi_def_cfa 4, 4
        ret
        .cfi_endproc
.LFE1007:
        .size   main, .-main
        .p2align 4,,15
        .type   _GLOBAL__sub_I__Z7productii, @function
```

1. The optimized code looks cleaner and more concise. For the loop optimization, it makes use of more registers other than offsets of ebp. I guess it allows for faster access if using more registers.
2. In the code, I did not use the multiplication to directly multiply x and y together, but rather did it in a loop to add x y times. I also noticed that in the non-optimized code, it involves a lot of jumping around, which is the implementation for the for-loop. In the optimized code, it simplifies the code by directly multiply the two parameters.
3. In the main of the optimized code, it created 16 bytes for the local variables, while in the "normal code it created 32 bytes. I guess this is because in the non-optimized code, it has to create room for the two local variables while in the optimized code it only has one.
4. Comparing the "normal" main code with the optimized main code, I found out that in the optimized code it didn't call the function at all. It moved the value of the product directly into [esp+4]. In my opinion, this is a kind of optimization that optimizes the program's execution speed.

Inheritance

For this question, I wrote a simple C++ program that has a class Q1, and class Q2 that extends Q1. In Q1 there are two fields—an integer and a double. In Q2 there is an extra field integer a. Both of them have constructors that initialize the field and destructors. In the main method I created object Q2 with a parameter of 5.

```
#include <iostream>
using namespace std;

class Q1{
private:
        int x;
        double z;
public:
        Q1(int i, double d);
        ~Q1();
};

Q1::Q1(int i, double d){
        x = i;
        z = d;
}
Q1::~Q1(){

}

class Q2 : public Q1{
private:
        int a;
public:
        Q2(int t);
        ~Q2();
};

Q2::Q2(int t):Q1(t, 7.5){
        a = t;
}
Q2::~Q2(){

}

int main(){
        Q2 q(5);
        return 0;
}
```

Then I generated the assembly code according to the C++ code. Here is the Q1 class assembly code. The bold and underlined part is the data layout of class Q1.

```
        _ZN2Q1C2Eid: (Q1 constructor)
        .LFB967:
                .cfi_startproc
                push    ebp
                .cfi_def_cfa_offset 8
                .cfi_offset 5, -8
                mov     ebp, esp
                .cfi_def_cfa_register 5
                sub     esp, 8
                mov     eax, DWORD PTR [ebp+16]
                mov     DWORD PTR [ebp-8], eax
                mov     eax, DWORD PTR [ebp+20]
                mov     DWORD PTR [ebp-4], eax
                mov     eax, DWORD PTR [ebp+8]
                mov     edx, DWORD PTR [ebp+12]
                mov     DWORD PTR [eax], edx
                mov     eax, DWORD PTR [ebp+8]
                fld     QWORD PTR [ebp-8]
                fstp    QWORD PTR [eax+4]
                leave
                .cfi_restore 5
                .cfi_def_cfa 4, 4
                ret
                .cfi_endproc
        .LFE967:
                .size    _ZN2Q1C2Eid, .-_ZN2Q1C2Eid
                .align 2
                .globl  _ZN2Q1D2Ev
                .type   _ZN2Q1D2Ev, @function
        _ZN2Q1D2Ev: (Q1 destructor)
        .LFB970:
                .cfi_startproc
                push    ebp
                .cfi_def_cfa_offset 8
                .cfi_offset 5, -8
                mov     ebp, esp
                .cfi_def_cfa_register 5
                pop     ebp
                .cfi_def_cfa 4, 4
                .cfi_restore 5
                ret
                .cfi_endproc
        .LFE970:
                .size    _ZN2Q1D2Ev, .-_ZN2Q1D2Ev
                .align 2
                .globl  _ZN2Q2C2Ei
                .type   _ZN2Q2C2Ei, @function
```

This is the assembly code for the Q2 class. The underlined part shows the additional data layout of class Q2. The bold line of code shows that it calls the Q1 constructor. It calls the constructor before setting up the additional field in Q2. This is because Q2 extends Q1. In the end of destructor part of Q2, it calls the destructor of Q1 after Q2 destructs all the things.

```
_ZN2Q2C2Ei: (Q2 constructor)
.LFB973:
        .cfi_startproc
        push    ebp
        .cfi_def_cfa_offset 8
        .cfi_offset 5, -8
        mov     ebp, esp
        .cfi_def_cfa_register 5
        sub     esp, 16
        mov     eax, DWORD PTR [ebp+8]
        fld     QWORD PTR .LC1
        fstp    QWORD PTR [esp+8]
        mov     edx, DWORD PTR [ebp+12]
        mov     DWORD PTR [esp+4], edx
        mov     DWORD PTR [esp], eax
        call    _ZN2Q1C2Eid
        mov     eax, DWORD PTR [ebp+8]
        mov     edx, DWORD PTR [ebp+12]
        mov     DWORD PTR [eax+12], edx
        leave
        .cfi_restore 5
        .cfi_def_cfa 4, 4
        ret
        .cfi_endproc
.LFE973:
        .size   _ZN2Q2C2Ei, .-_ZN2Q2C2Ei
        .align 2
        .globl  _ZN2Q2D2Ev
        .type   _ZN2Q2D2Ev, @function
_ZN2Q2D2Ev: (Q2 destructor)
.LFB976:
        .cfi_startproc
        push    ebp
        .cfi_def_cfa_offset 8
        .cfi_offset 5, -8
        mov     ebp, esp
        .cfi_def_cfa_register 5
        sub     esp, 4
        mov     eax, DWORD PTR [ebp+8]
        mov     DWORD PTR [esp], eax
        call    _ZN2Q1D2Ev
        leave
        .cfi_restore 5
        .cfi_def_cfa 4, 4
        ret
        .cfi_endproc
.LFE976:
        .size   _ZN2Q2D2Ev, .-_ZN2Q2D2Ev
        .globl  main
        .type   main, @function
```

In the main method, I created a Q2, and in the yellow highlighted part it called the Q2 constructor. And in the blue highlighted part it called the Q2 destructor.

```
main:
.LFB978:
        .cfi_startproc
        push    ebp
        .cfi_def_cfa_offset 8
        .cfi_offset 5, -8
        mov     ebp, esp
        .cfi_def_cfa_register 5
        push    ebx
        and     esp, -8
        sub     esp, 24
        mov     DWORD PTR [esp+4], 5
        lea     eax, [esp+8]
        mov     DWORD PTR [esp], eax
        .cfi_offset 3, -12
        call    _ZN2Q2C1Ei
        mov     ebx, 0
        lea     eax, [esp+8]
        mov     DWORD PTR [esp], eax
        call    _ZN2Q2D1Ev
        mov     eax, ebx
        mov     ebx, DWORD PTR [ebp-4]
        leave
        .cfi_restore 5
        .cfi_def_cfa 4, 4
        .cfi_restore 3
        ret
```