0.) NAME: ____________________________


1.) Fill in the blanks to make true statements, for a 64-bit Linux machine (like NetRun's default).

• A function's return value is stored ___ in register rax ________________________________.

• A function's first parameter is stored _____ in register rdi _________________.

• A class with one long uses ___ 8 ___ bytes of memory.

• You would write a C++ prototype for an assembly function named “bar” with no parameters or return value as:

   _____ extern “C” void bar(void); ______

(Score: 10 points.)

2.) Convert each piece of C++ code to assembly, and vice versa:

<table>
<thead>
<tr>
<th>C++</th>
<th>Assembly</th>
</tr>
</thead>
<tbody>
<tr>
<td>long grabit() {</td>
<td>grabit:</td>
</tr>
<tr>
<td>return 5;</td>
<td>mov rax,5</td>
</tr>
<tr>
<td>}</td>
<td>ret</td>
</tr>
</tbody>
</table>

// The functions phase1 and phase2
// both take and return one long.
long pipeline(long x) {                   | pipeline:
    return phase2(phase1(x)+3);         | ; x stored in rdi
}                                       | call phase1
                                          | mov rdi,rax
                                          | add rdi,3
                                          | call phase2
                                          | ; return value in rax
                                          | ret                                       |

(Score: 30 points. Each piece of code is separate.)

3.) There are several memory allocation styles. Fill in the remaining table entries.

<table>
<thead>
<tr>
<th>What?</th>
<th>Why?</th>
<th>How? (allocate one long)</th>
<th>Static or Dynamic?</th>
</tr>
</thead>
<tbody>
<tr>
<td>section .data</td>
<td>Named static variables</td>
<td>dq 3</td>
<td>static</td>
</tr>
<tr>
<td>the stack</td>
<td>Fast function-local allocation</td>
<td>sub rsp,8</td>
<td>dynamic</td>
</tr>
<tr>
<td>malloc</td>
<td>Long-lived allocations</td>
<td>mov rdi, 8 call malloc</td>
<td>dynamic</td>
</tr>
</tbody>
</table>

(Score: 10 points. “Static” allocation happens once per program; “Dynamic” can happen again and again.)
4.) Fix the errors in this assembly translation of this C++:

```cpp
for (int i=0; i<n; i++) arr[i]=3;
```

; input: rsi == arr, rdi == n
mov rcx,0 ; i
jmp check ; need to verify i<n before starting loop

start:
mov DWORD[rsi + rcx *4], 3 ; need to scale for array access
(add rcx,1)
check:
cmp rcx, rdi
jle start ; just jl, jle is like i<=n

(Score: 15 points.)

5.) You see the compiler uses “push rbx” at the start of your function, and “pop rbx” at the end. Why?

rbx is a preserved register, and the compiler wants to store something in it.

(Score: 10 points.)

6.) If each piece of code returns a value, write the value. If it'll crash or hang, write that, and why.

<table>
<thead>
<tr>
<th>push rax</th>
<th>push 2</th>
<th>mov rax, 9</th>
<th>push 6</th>
<th>mov rax, 0x3F and rax, 0x55</th>
</tr>
</thead>
<tbody>
<tr>
<td>push 3</td>
<td>push 7</td>
<td>push rax</td>
<td>mov rax, QWORD[rsp]</td>
<td>pop rax</td>
</tr>
<tr>
<td>pop rax</td>
<td>pop rcx</td>
<td>sub rsp, 8</td>
<td>pop rcx</td>
<td>ret</td>
</tr>
<tr>
<td>ret</td>
<td></td>
<td>ret</td>
<td></td>
<td>ret</td>
</tr>
</tbody>
</table>

Crash
(not enough pops)
7
Crash
(moved stack pointer wrong way; add rsp,8 would work)
6
0x15

(Score: 15 points. Several of these are ... subtle. Be careful.)

7.) You had asked your newest employee to measure the speed of an empty function on Intel's new server chip, and his entire email in response was “503,148.7 per second”. What's wrong with his response?

1.) No units—should be “xxx function calls per second, as measured from C++.”
2.) No error bars—should be “xxx +-yy”, or even give a standard deviation or histogram of measurement variance.

(Score: 10 points.)