R. I. Greenberg, Comp. Sci. Dept., Loyola U., 1052 W. Loyola Ave., Chicago, IL

Assignment #5

Issued 10/27 Due 11/10

Two options to submit your .c file:

- Easiest if you are working on a Loyola GNU/Linux machine: Copy your file to the directory `~rig/c264hw5sub` with a filename in the form `EMAIL-X.c`, where `EMAIL` is your email address, and `X` is a "random" string of at least 8 alphanumeric characters. The Unix command for this would look *similar* to:

  ```
  cp switch4.c ~rig/c264hw5sub/YOUREMAILADDRESS-RANDOM.c
  ```

  where you must put your own things in place of the all caps. (Don't cut and paste from the PDF, or the tilde might not come out right.) *Remember that if you submit this way the file must be readable by all, though you will want to have used* `chmod` *to protect the directory containing the file. Protections show with the* `ls -l` *command illustrated below.* You can verify successful submision by using the "ls" command with the same file name you just copied to, specifically you can use a command *similar* to:

  ```
  ls -l ~rig/c264hw5sub/YOUREMAILADDRESS-RANDOM.c
  ```

- Or if you prefer: Submit the file through the online submission mechanism on my course web page. Submit it as `switch4.c` or `5.c`.

HW5-1 (56 points)

The code that follows shows an example of branching on an enumerated type value in a switch statement. Recall that enumerated types in C are simply a way to introduce a set of names having associated integer values. By default, the values assigned to the names go from 0 upward. In our code, the actions associated with the different case labels have been omitted.

```
/*Enumerated Type creates set of constants numbered 0 and upward*/
typedef enum {ACASE, BCASE, CCASE, DCASE, ECASE} modetype;
long switch4 (long *p1, long *p2, modetype action) {
  long result;
  switch (action){
  case ACASE:
  case BCASE:
  case CCASE:
  case DCASE:
  case ECASE:
  default:
  }
  return result;
}
```

Shown below is a possible version (not necessarily most compact or efficient) of the generated assembly code implementing the different actions.

```
.file "switch4-soln.c"
.text
.globl switch4
.type switch4, @function
switch4:
.LFB0:
.cfi_startproc
cmpl $4, %edx
ja .L2
movl %edx, %edx
jmp *.L4(,%rdx,8)
.section .rodata
.align 8
.align 4
.L4:
.quad .L3
.quad .L5
.quad .L6
.quad .L7
.quad .L9
.text
.L2:
movl $2, %eax
ret
.L3:
movq (%rdi), %rax
subq (%rsi), %rax
movq %rax, (%rsi)
ret
.L5:
movq $31, (%rdi)
movq (%rsi), %rax
ret
.L6:
movq (%rdi), %rax
movq %rax, %rdx
addq (%rsi), %rdx
movq %rdx, (%rdi)
ret
.L7:
movq (%rsi), %rax
movq %rax, (%rdi)
movl $24, %eax
ret
.L9:
movl $24, %eax
ret
.cfi_endproc
.LFE0:
.size switch4, .-switch4
.ident "GCC: (Ubuntu 4.8.4-2ubuntu1~14.04.4) 4.8.4"
.section .note.GNU-stack,"",@progbits
```

Fill in the missing parts of the C code. Watch out for cases that should be written in the C code with a fall-through. You may want to test your C code by compiling with the `-S` switch. Your compiler may not generate identical code, but it should be functionally equivalent (and it ought to at least compile). Compiling with optimization switch `-O1` is probably best to get close to the given assembly code.