Name: **SOLUTION** UID#: DirectoryID:

CMSC216: Practice Exam 2A SOLUTION

Fall 2025

University of Maryland

Exam period: 20 minutes *Points available:* 40 *Weight:* 0% of final grade

Problem 1 (20 pts): Nearby is a `main()` demonstrating the use of the function `setday()`. Below each call to `setday()`, its expected behavior and return are printed.

Implement this function according to the documentation given **in x86-64 assembly**. Comments below the `yearday_t` struct give information about how it lays out in memory and as a packed register argument.

```
.text   ######## SOLUTION ########
.global  setday
setday:
        ## rdi is packed {int day; int year}
        ## Extract fields from rdi
        movq %rdi, %rdx
        ## andl $0xFFFFFFFF, %edx  # edx now day
        andq $0xFFFFFFFF, %rdx  # edx now day
        movq %rdi, %rcx
        shrq $32, %rcx          # ecx now year
        ## andl $0xFFFFFFFF, %ecx  # optional mask
        andq $0xFFFFFFFF, %rcx  # optional mask

        ## do range checking
        cmpl $0, %edx
        jl .ERROR
        cmpl $1970, %ecx
        jl .ERROR

        subl $1970, %ecx      # offset from 1970
        imull $365, %ecx      # mult by days/year
        addl %edx, %ecx       # add days

        ## write back to global variable
        movl %ecx, DAYS_SINCE_1970(%rip)

        movl $0, %eax
        ret

.ERROR:
        ## error case, set global and ret
        movl $-1,DAYS_SINCE_1970(%rip)
        movl $1, %eax
        ret
```

```c
#include <stdio.h>
#include <stdlib.h>

// global: days since 1/1/1970
int DAYS_SINCE_1970 = 0;

// struct containing date info
typedef struct {
  int day;    // current day in this year
  int year;   // current year
} yearday_t;
// Layout of yearday_t in memory and
// as a packed register argument.
//
// |       | Byte |   Byte | Packed |
// | Field | Size | Offset |   Bits |
// |-------+------+--------+--------|
// | day   | 4    |     +0 |   0-31 |
// | year  | 4    |     +4 |  32-63 |

int setday(yearday_t yd);
// DEFINED IN ASSEMBLY
//
// Extracts the day and year fields
// from the provided struct yd. If day is
// negative or year is < 1970, sets the
// global variable DAYS_SINCE_1970 to
// be -1 and returns 1. Otherwise
// computes the number of days since
// 1970 based on these fields assuming
// NO LEAP YEARS and 365 days per
// year. Sets the global variable
// DAYS_SINCE_1970 to this value and
// returns 0.

int main(int argc, char *argv[]){
  // Demonstrate 3 examples of setday()
  int ret;
  yearday_t yd1 =
    { .day = 20, .year = 1970 };
  ret = setday(yd1);
  printf("%3d days since 1970 (ret: %d)\n",
         DAYS_SINCE_1970, ret);
  //  20 days since 1970 (ret: 0)

  yearday_t yd2 =
    { .day = 3, .year = 1972 };
  ret = setday(yd2);
  printf("%3d days since 1970 (ret: %d)\n",
         DAYS_SINCE_1970, ret);
  // 733 days since 1970 (ret: 0)

  yearday_t yd3 =
    { .day = 7, .year = 1955 };
  ret = setday(yd3);
  printf("%3d days since 1970 (ret: %d)\n",
         DAYS_SINCE_1970, ret);
  //  -1 days since 1970 (ret: 1)
  return 0;
}
```

Problem 2 (10 pts): Below is a main() function which uses the function setarray(). As the demo shows, compiling with a C version of this function works fine but the assembly version has some problems.

```
// setarray_main.c
#include <stdio.h>
int main(){
  long arr[3];
  setarray(arr,3,10);
  for(int i=0; i<3; i++){
    printf("%2d ",arr[i]);
  }
  printf("\n");
  return 0;
}
```

```
// setarray_c.c
void setarray(long *arr,
              long len,
              long val)
{
  for(long i=0; i<len; i++){
    arr[i] = val;
  }
  return;
}
```

```
## setarray_asm.s
 1 .text
 2 .globl  setarray
 3 setarray:
 4         movq    $0, %rax
 5 .LOOP:
 6         cmpq    %rsi, %rax
 7         jg      .DONE
 8         movq    %rdx,(%rdi,%rax,8)
 9         addq    $1, %rax
10         jmp     .LOOP
11 .DONE:
12         ret
```

```
>> gcc setarray_main.c setarray_c.c
>> ./a.out
10 10 10
```

```
>> gcc setarray_main.c setarray_asm.s
>> ./a.out
10 10 10
*** stack smashing detected ***: terminated
Aborted (core dumped)
```

Describe why the assembly version causes Stack Smashing and how to fix it.

SOLUTION: The assembly instruction at line 7 in the assembly code is what terminates the loop. Unfortunately, this jg *causes one extra loop iteration which goes out of bounds in the target array (the 4th element at index 3 in the* main *function). This changes data near the return address which is detected as a problem causing the program to terminate. The fix is to change* jg *to* jge *to stop going out of bounds in the array.*

Problem 3 (10 pts): While debugging a binary program, Nils Punters encounters an assembly instruction that baffles him: test %rax,%rax. Nils is struggling to understand what this could possibly accomplish. **Explain** what the testX instruction does AND what it is likely being used to do in the code Nils is examining which is shown nearby.

```
+=========================GDB=======================================+
|>>0x55154 <nodes_sorted+11>  test   %rax,%rax  # Nils: WTF?          |
|  0x55157 <nodes_sorted+14>  je     0x555555555167 <nodes_sorted+30> |
|  0x55159 <nodes_sorted+16>  mov    (%rax),%edx                      |
|  0x5515b <nodes_sorted+18>  cmp    %ecx,%edx                        |
|  0x5515d <nodes_sorted+20>  jl     0x555555555173 <nodes_sorted+42> |
|  0x5515f <nodes_sorted+22>  mov    0x8(%rax),%rax                   |
|  0x55163 <nodes_sorted+26>  mov    %edx,%ecx                        |
|  0x55165 <nodes_sorted+28>  jmp    0x555555555154 <nodes_sorted+11> |
|  0x55167 <nodes_sorted+30>  mov    $0x1,%eax                        |
|  0x5516c <nodes_sorted+35>  ret                                     |
+===================================================================+
```

SOLUTION: The testX *instruction is equivalent to a bitwise-And but the result is discarded. It is run solely to set the FLAGS register. Testing a register against itself yields information such as whether it is Negative (signed) or Zero and will set the flags register accordingly. A 64-bit* test *like the one Nils is looking at on* <node_sorted+11> *could be used to check a 64-bit number for being Zero or equivalently checking to see if a Pointer is* NULL *(encoded as 0 in binary). The use of "nodes" in the name of the function and the fact that that the* %rax *register is used to access main memory implies that it is a pointer and the instruction is checking whether it is* NULL*.*