# CS210 Fall 2023: PS6A

## Instructions

For all multiple choice questions fill **ONE AND ONLY ONE circle**. Be sure to fill the circle in completely.
For all the questions we encourage you to login into the provided UNIX environment and explore your answers. For some questions, you must use the UNIX environment to answer them.

**If you use checkmarks or other symbols the auto-grader may not be able to process your answer and will assign you a grade of zero.**

**All pages must have your name and id written on them. Unidentified pages will not be graded**

First Name: ______________________ Last Name: ______________________

BU ID: ____________

## Multiple Choice

1. (1 point) The C programming language

   ○ is composed of two components: the Core Language and a standard Library of functions.

   ○ source is directly executable on the processor

   ○ allows one to do things not possible in assembly language

   ○ was developed to support direct manipulation of HTML

   ○ all of the above

   ○ none of the above

2. (1 point) The first stage of the C toolchain is

   ○ linking

   ○ preprocessing

   ○ compilation

   ○ assembling

   ○ none of the above

3. (1 point) Each location of a process's address space

   ○ is valid and has an associated value

   ○ represents a single byte of memory

   ○ has a fixed contents that never changes

   ○ is a variable sized array

   ○ all of the above

   ○ none of the above

4. (1 point) The indirection operator, in C,

   ○ provides access to the value that a pointer is pointing to

   ○ provides the address of a value

   ○ provides the type of a value

   ○ provides the size of a value

   ○ all of the above

   ○ none of the above

5. (1 point) By the C calling conventions, if more than 6 arguments are passed into a C function, the remaining ones, that could not be assigned to registers, are pushed onto the stack

   ○ True

   ○ False

6. (1 point) Structures in C

   ○ always requires at least 8 bytes
   ○ may not contain pointers to structures of other types
   ○ must be placed in the data section of the binary
   ○ must be allocated on the heap
   ○ all of the above
   ○ none of the above

7. (1 point) A function can be defined in many files

   ○ True
   ○ False

8. (1 point) What must be passed to `printf` to send bytes to standard output

   ○ a format specifier
   ○ a char pointer to a format string
   ○ a file descriptor of standard output
   ○ the size of bytes you want to send
   ○ all of the above
   ○ none of the above

9. (1 point) A local variable of a C function

   ○ must be placed on the stack
   ○ must be assigned to a register
   ○ will be placed on the stack or assigned to a register
   ○ will be placed in the heap if there are no available registers or space on the stack
   ○ none of the above

10. (1 point) How many times will the following loop execute?

```
int i=5, j=0;
while (i-j){}
```

   ○ 0
   ○ 1
   ○ 4
   ○ 5
   ○ 6
   ○ none of the above

11. Provide the following value of the C expressions as a 32-bit hexadecimal value.
**DO NOT SKIP LEADING ZEROS:**
Eg. a value of 0 should be written as `00000000` and 1 as `00000001`.

Assume a 64 bit computer that uses 2's complement representation, INT_MAX and INT_MIN are defined as the computer's signed 32-bit integer representation maximum and minimum value respectively, and:

**int** x = −1, y = 0x7eedface, z = INT_MIN, i = **sizeof** ( **int** );

(a) (1 point) x : 0x______________________________

(b) (1 point) y : 0x______________________________

(c) (1 point) z : 0x______________________________

(d) (1 point) i : 0x______________________________

(e) (1 point) z<<12 : 0x______________________________

(f) (1 point) z<<((i>>1)−1) : 0x______________________________

(g) (1 point) ~0 == (z + INT_MAX) : 0x______________________________

(h) (1 point) y & 0xffff : 0x______________________________

(i) (1 point) y>>16 : 0x______________________________

(j) (1 point) (y>>16) | 0xffff : 0x______________________________

(k) (1 point) (~(0x10>>2)+1) == (x*i) : 0x______________________________

(l) (1 point) (x+1) + −1 : 0x______________________________

(m) (1 point) (~((~ x)<<1)) & y : 0x______________________________

(n) (1 point) ((z+INT_MIN)<<i) ^ ((z+INT_MIN)<<i) : 0x______________________________

12. (5 points) Given the assembly code on the left fill in the blanks in the C code on the right that it corresponds to.

```
        .intel_syntax noprefix
        .text
        .globl  func
func:
        xor     eax, eax
        xor     r8d, r8d
.L2:
        add     r8d, DWORD PTR A[0+rax*4]
        inc     rax
        cmp     rax, 10
        jne     .L2
        mov     eax, r8d
        ret
```

```

#define B _____

#define T _____

T A[B];

T func()
{
  int i;
  T s = 0;

  for (________ ;

       i < B;

       ________ ) {


    s = _________;
  }
  return s;
}
```

13. (6 points) Given the assembly code on the left, fill in the blanks in the C code on the right that it corresponds to.

```
        .intel_syntax noprefix
        .text
        .globl  func
func:
        xor     rax, rax
.L2:
        test    rdi, rdi
        je      .L7
        mov     rcx, QWORD PTR [rdi+16]
        mov     rdx, QWORD PTR [rdi+24]
        cmp     rsi, 12
        jle     .L3
        add     rax, QWORD PTR [rdi]
        mov     rdi, rcx
        jmp     .L2
.L3:
        add     rax, QWORD PTR [rdi+8]
        mov     rdi, rdx
        jmp     .L2
.L7:
        ret
```

```
struct S {
  long long x;
  long long y;
  struct S *f;
  struct S *b;
};

long long
func(struct S *h,
     long long v)
{
  long long r = 0;

  while (h != ________ ) {
    if (v > _________ ) {

      r = r + _________;

      h = ___________;
    } else {

      r = r + _________;

      h = __________;
    }
  }
  return r;
}
```

14. (4 points) Given the assembly code on the left, fill in the blanks in the C code on the right that it corresponds to.

```
        .intel_syntax noprefix
        .text
        .globl  f
f:
        imul    rdi, rdi, 7
        add     rdi, rsi
        mov     rax, QWORD PTR A[0+rdi*8]
        ret
        .comm   A,280,32
```

```

#define C __________

#define R __________

long long A[C][R];

long long f(long long c,
            long long r)
{
  return A[c][r];
}
```

15. Present the output for each line for the program below. Assume that it is compiled and executed on a 64-bit, little endian computer that uses 2's complement representation.

```c
#include <stdio.h>
typedef unsigned char *byte_pointer;

void show_bytes(byte_pointer start, int len) {
  int i;
  for(i=0; i<len; i++)
    printf(" %.2x", start[i]);
  printf("\n");
}

int main(void)
{
  unsigned int ux = 0x8000000;
  int x = ux;
  long long unsigned uy = ux;
  long long y = x;

  ux = ux >> 8;
  x  = x >> 8;

  printf("%lu\n", sizeof(ux));
  show_bytes((byte_pointer)&ux, sizeof(ux));
  printf("0x%x 0x%x\n", ux, x);
  printf("%d %lld\n", x>>19 , y>>19);

  return 0;
}
```

(a) (2 points) ______________________________________________

(b) (2 points) ______________________________________________

(c) (2 points) ______________________________________________

(d) (2 points) ______________________________________________

16. (40 points) Assume that the code below is compiled for and executed on a 64-bit little endian computer. Please provide the hex byte value and name of the variable that each address indicated corresponds to. **For addresses that correspond to arrays please indicate the array name and index the address belongs to, e.g.** `str[4]`**. If an address does not correspond to a variable, leave the name blank empty. You can assume that the padded bytes in memory between variables were initialized to 0s. For** `char` **variables the value should be provided as an ASCII character, e.g.** `'a'` **. For all other variables the values should be provided as a two digit hex value.**

```
char  bar[] = "To B or not to";
short sv = 0xBA0D;
int iv  = 0xCAFEF00D;

int * p1 = &iv;
char * p2 = &(bar[3]);

void func()
{
  *p2 = 0x43;
  p2 = (char *)&sv;
  p2 = p2 + 1;
  *p2 = 0x5e;
  *p1 = 0x12345678;
}
```

| Variable | Address of Variable |
|---|---|
| bar | 0x0000555555558010 |
| sv | 0x0000555555558020 |
| iv | 0x0000555555558024 |
| p1 | 0x0000555555558028 |
| p2 | 0x0000555555558030 |

Given the above fill in the following

(1) 0x0000555555558010: Value:______________ Name:______________

(2) 0x0000555555558011: Value:______________ Name:______________

(3) 0x0000555555558012: Value:______________ Name:______________

(4) 0x0000555555558013: Value:______________ Name:______________

(5) 0x0000555555558014: Value:______________ Name:______________

(6) 0x0000555555558015: Value:______________ Name:______________

(7) 0x0000555555558016: Value:______________ Name:______________

(8) 0x0000555555558017: Value:______________ Name:______________

(9) 0x0000555555558018: Value:______________ Name:______________

(10) 0x0000555555558019: Value:______________ Name:______________

(11) 0x000055555555801a: Value:______________ Name:______________

(12) 0x000055555555801b: Value:______________ Name:______________

(13) 0x000055555555801c: Value:______________ Name:______________

(14) 0x000055555555801d: Value:______________ Name:______________

(15) 0x000055555555801e: Value:______________ Name:______________

(16) 0x000055555555801f: Value:______________ Name:______________

(17) 0x0000555555558020: Value:______________ Name:______________

(18) 0x0000555555558021: Value:______________ Name:______________

(19) 0x0000555555558022: Value:______________ Name:______________

(20) `0x0000555555558023`: Value:______________ Name:______________

(21) `0x0000555555558024`: Value:______________ Name:______________

(22) `0x0000555555558025`: Value:______________ Name:______________

(23) `0x0000555555558026`: Value:______________ Name:______________

(24) `0x0000555555558027`: Value:______________ Name:______________

(25) `0x0000555555558028`: Value:______________ Name:______________

(26) `0x0000555555558029`: Value:______________ Name:______________

(27) `0x000055555555802a`: Value:______________ Name:______________

(28) `0x000055555555802b`: Value:______________ Name:______________

(29) `0x000055555555802c`: Value:______________ Name:______________

(30) `0x000055555555802d`: Value:______________ Name:______________

(31) `0x000055555555802e`: Value:______________ Name:______________

(32) `0x000055555555802f`: Value:______________ Name:______________

(33) `0x0000555555558030`: Value:______________ Name:______________

(34) `0x0000555555558031`: Value:______________ Name:______________

(35) `0x0000555555558032`: Value:______________ Name:______________

(36) `0x0000555555558033`: Value:______________ Name:______________

(37) `0x0000555555558034`: Value:______________ Name:______________

(38) `0x0000555555558035`: Value:______________ Name:______________

(39) `0x0000555555558036`: Value:_____________ Name:_____________

(40) `0x0000555555558037`: Value:_____________ Name:_____________

17. (12 points) Given the following code, complete the diagram on the next page assuming main runs to completion. Fill in all blank boxes, either with an arrow or value as needed. A

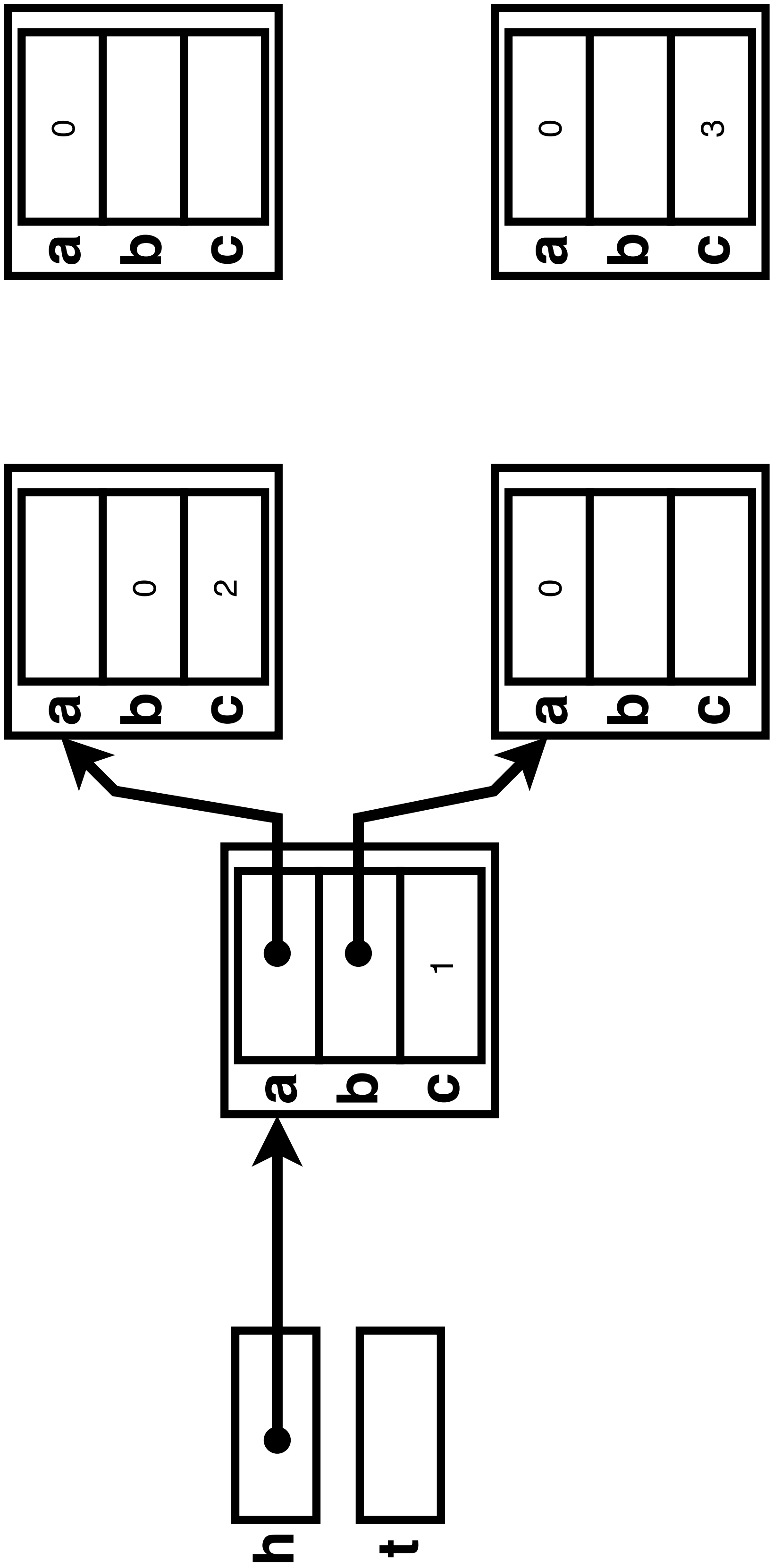
```
#include <stdlib.h>

struct Node {
  struct Node *a;
  struct Node *b;
  long long    c;
};

struct Node * new()
{
  struct Node *n = malloc(sizeof(struct Node));
  n->a=0; n->b=0; n->c=0;
  return n;
}

struct Node *h;
struct Node *t;

int main(int argc, char **argv)
{
  h = new();
  h->c = 1;
  t = new();
  t->c = 3;
  h->b = new();
  h->b->c = 4;
  t = new();
  h->a = t;
  t->c = 2;
  t = h;
  h->a->a = new();
  t->b->b = h->a->a;
  h->a->a->c = 5;
  t->a->c = 2;
  h->b->b->c = 8;
  return 0;
}
```


0
a
b
c
0
3
a
b
c
0
2
a
b
c
0
a
b
c
1
a
b
c
h
t