Name:________________________________________

# CSE351 Winter 2016, Final Examination
# March 16, 2016

## Please do not turn the page until 2:30.

Rules:

- The exam is closed-book, closed-note, etc.
- **Please stop promptly at 4:20.**
- There are **125 (not 100) points**, distributed **unevenly** among **11** questions (most with multiple parts):
- **The exam is printed double-sided.**
- The last two pages of the exam have useful reference information.

Advice:

- Read questions carefully. Understand a question before you start writing.
- **Write down thoughts and intermediate steps so you can get partial credit. But clearly indicate what is your final answer.**
- The questions are not necessarily in order of difficulty and some in the middle can be done fairly quickly. **Skip around and make sure you get to all the questions.**
- If you have questions, ask.
- Relax. You are here to learn.

Name:________________________________________

1. *C data and locality* (**10** points) We consider two versions of C type definitions for a collection of student names and exam scores:

```
// Version A
struct Student {
  char * name;
  int exam_score;
};
struct ExamScores {
  int numStudents;
  struct Student * students; // array of length numStudents
};

// Version B: uses 2 arrays instead of 1, where name and score for
// each student is (implicitly) at the same array index
struct ExamScores {
  int numStudents;
  char ** names; // array (of strings) of length numStudents
  int * exam_scores; // array of length numStudents
};
```

Here is correct "version A" code for a function `get_score` that looks up a student's score by name. Note `strcmp` is defined in the C standard library for comparing strings (it returns 0 for equal strings).

```
int get_score(char * name, struct ExamScores * scores) {
  for(int i = 0; i < scores->numStudents; i++) {
     if(strcmp(name,scores->students[i].name)==0)
       return scores->students[i].exam_score;
  }
  return -1; // special value for name-not-found
}
```

(a) Rewrite `get_score` for "Version B."

(b) For a large number of students, would you expect "version A" or "version B" to get better performance for calls (with various names) to `get_score`? *Explain your answer.*

**Solution:**

(a)
```
int get_score(char * name, struct ExamScores * scores) {
  for(int i = 0; i < scores->numStudents; i++)
     if(strcmp(name,scores->names[i])==0)
       return scores->exam_scores[i];
  return -1;
}
```

(b) Version B performs better because it has better spatial locality. The pointers to name strings are in contiguous memory, so more fit it one cache line. We only need one score, so it is not very helpful to have the name/score for a student in contiguous memory as in Version A.

Name:________________________________________

2. *Caching* (**18** points) For this problem, assume:

   - 64-bit addresses
   - A single-level 128KB data cache
   - A 32-byte cache-line size
   - The cache "starts empty."
   - The local variables below are all in registers.

   Consider this C code:

```
void f(int * x, int * y, int n) {
  for(int i=0; i < n; i++) {
     x[i] = y[i] / 2;
  }
}
```

   (a) Suppose the cache is direct-mapped.

      i. Which bits in the addresses of `&x[0]` and `&y[0]` would have to be the same for these addresses to map to the same (1-element) set in the cache? Draw a picture if you need to make clear which bits you mean.

      ii. If `&x[0]` and `&y[0]` do map to the same (1-element) cache set (and, for simplicity, have an offset of 0 in the cache line), what would you expect the cache miss rate to be for an execution of `f`?

      iii. If `&x[0]` and `&y[0]` do not map to the same (1-element) cache set (and, for simplicity, have an offset of 0 in the cache line), what would you expect the cache miss rate to be for an execution of `f`?

   (b) Now suppose the cache is 4-way set-associative.

      i. Which bits in the addresses of `&x[0]` and `&y[0]` would have to be the same for these addresses to map to the same (4-element) set in the cache? Draw a picture if you need to make clear which bits you mean.

      ii. If `&x[0]` and `&y[0]` do map to the same (4-element) cache set (and, for simplicity, have an offset of 0 in the cache line), what would you expect the cache miss rate to be for an execution of `f`?

      iii. If `&x[0]` and `&y[0]` do not map to the same (4-element) cache set (and, for simplicity, have an offset of 0 in the cache line), what would you expect the cache miss rate to be for an execution of `f`?

   **Solution:**

   (a) i. We have room for $2^{17} - 2^5 = 2^{12}$ cache lines. Low 5 bits are offset. Next 12 bits are index. *So the 6th-17th least-significant bits.* (The remaining 42 bits are the tag.)

      ii. 100%: every access is either a compulsory miss or a conflict miss because the previous access must have evicted the line we needed.

      iii. 12.5%: now we have only compulsory misses and after 1 miss, we get 7 subsequent hits due to spatial locality.

(b) i. We now have 1/4 as many sets, so we have 10 bits for the index, *the 6th-15th least-significant bits.*

ii. 12.5%, no eviction of lines, so only compulsory misses

iii. 12.5%, no eviction of lines, so only compulsory misses

Name:________________________________________

3. *Caching and Locality* (**10** points) Consider this C code:

```
void f(int * a, int * b, int n) {
  for(int i=0; i < n; i++) {
     a[i] = b[i] / 2;
  }
  for(int i=0; i < n; i++) {
     a[i] += 7;
  }
}
```

(a) Rewrite the function so that it (1) performs the same computations but in a different order, (2) has better temporal locality than before, and (3) has no worse spatial locality than before.

(b) Suppose you did the same rewriting for a Java method (where in Java `a` and `b` would have type `int[]`). In Java would this provide:

- a much larger increase in locality,
- a much smaller increase in locality, or
- about the same increase in locality?

(c) In the C version, your rewriting is probably *wrong* for certain unusual argument values. Give an example where the code above and your answer to part (a) update memory differently. Notes:

- This is a tricky question not worth many points.
- Hint: In Java, there are no such examples.

**Solution:**

(a)
```
void f(int * a, int * b, int n) {
  for(int i=0; i < n; i++) {
    a[i] = b[i] / 2;
    a[i] += 7;
  }
}
```

(b) about the same increase

(c) The problem arises when `a` points ahead into the same array. In particular if `a-b` is greater than `0` and less than `n`, then the order of computations matters because `a` and `b` are overlapping so the writes to `a[i]` are affecting later reads of `b[i]`. For example, suppose `a-b==1`, `n==2`, and `b` points to an array holding `{1,2,3}`. Then the original code updates the array to `{1,7,7}` but the rewritten code produces `{1,7,10}`.

Name:__________________________________________

4. *Processes* (**12** points) In this problem, assume Linux.

   (a) Can the same program be executing in more than one process simultaneously?

   (b) Can a single process change what program it is executing?

   (c) When the operating system performs a context switch, what information does *NOT* need to be saved/maintained in order to resume the process being stopped later (circle all that apply):

   - The page-table base register
   - The value of the stack pointer
   - The time of day (i.e., value of the clock)
   - The contents of the TLB
   - The process-id
   - The values of the process' global variables

   (d) Give an example of an exception (asynchronous control flow) in which it makes sense to later re-execute the instruction that caused the exception.

   (e) Give an example of an exception (asynchronous control flow) in which it makes sense to abort the process.

   **Solution:**

   (a) Yes (the question is ambiguous as to what "simultaneous" means. We clarified during the exam, "Assume it *is* the case that multiple processes execute simultaneously. Then the question is whether more than one of these processes can be executing the same program." Under this interpretation, only "yes" is plausibly correct.)

   (b) Yes

   (c) The time of day and the contents of the TLB

   (d) Page fault for memory on disk (other answers possible; full credit given just for page-fault even though that's ambiguous)

   (e) Division by zero (other answers possible)

Name:________________________________________

5. *Fork* (**5** points) Consider this code using Linux's `fork`:

```
int x = 7;
if(fork()) {
   x++;
   printf(" %d ", x);
   fork();
   x++;
   printf(" %d ", x);
} else {
  printf(" %d ", x);
}
```

What are *all* the different possible outputs (order of things printed) for this code? (Hint: There are four of them.)

**Solution:**

```
7 8 9 9
8 7 9 9
8 9 7 9
8 9 9 7
```

Note: If you actually try this out, you may see 5 numbers printed, which is rather surprising. The issue is the implementation of `printf` may *buffer* output and the second `fork` call in the code then copies the not-empty output buffer into the child process. You can fix this in the code by putting `fflush(stdout);` after the first call to `printf`. In terms of the exam, the four outputs above *are* all still possible and of course we didn't expect you to also list the fifth possible output that arises from this buffered-output issue.

Name:________________________________________

6. *Protection* (**10** points) We saw several different ways to prevent a running program from doing something, including:

   A. A software check inserted by the compiler that is performed when the program executes (by having extra instructions/work in the code)
   B. A check performed by the compiler itself (failing compilation if the check fails)
   C. A hardware check performed directly by the hardware when the program executes (where the earlier decision of what is allowed may be set by the program and/or the operating system, but then the hardware does the checking)
   D. Simply providing no assembly instruction / operands that can perform the operation

   For each of the following, answer (A), (B), (C), or (D) for the most common way to prevent it:

   (a) Executing code that is on the call-stack
   (b) A process accessing the (private) memory of another process
   (c) A process accessing a portion of virtual memory it has not been given by the operating system
   (d) Branching (i.e., taking a conditional jump) based on whether a memory access hits or misses in the cache
   (e) A buffer overflow in Java

   **Solution:**

   (a) C
   (b) D
   (c) C
   (d) D
   (e) A

Name:________________________________________

7. *Virtual Memory* (**19** points)

   (a) Can a memory access ever be a TLB hit and cause a page fault?

   (b) Can a memory access ever be a TLB miss and cause a page fault?

   (c) Assume a page size of 64KB, a TLB with 128 entries, 4-way set associativity in the TLB, and a direct-mapped single-level cache of size 128KB. What are the maximum number of distinct addresses (in bytes) that a program could access in-between TLB misses? That is, during a sequence on TLB hits with no misses, how many different bytes of memory could possibly be accessed. (Hint: You do not need all the information given. Hint: We are definitely not assuming the TLB starts empty.)

   (d) Consider a memory location $\ell$ when accessing $\ell$ results in a TLB hit. For each of the following, answer "all", "some", or "none."

      i. How many of the bits for $\ell$'s virtual page number are in the TLB?

      ii. How many of the bits for $\ell$'s physical page number are in the TLB?

      iii. How many of the bits for $\ell$'s virtual page offset are in the TLB?

      iv. How many of the bits for $\ell$'s physical page offset are in the TLB?

   (e) Why does the operating system switch to running a different program after a page fault? (Answer in at most two sentences.)

   (f) Why is writing to the page-table base register something that only the operating system (not "regular" code in a process) should be allowed to do? (Answer in at most two sentences.)

**Solution:**

(a) No

(b) Yes

(c) $64\text{KB} \cdot 128 = 2^{23} = 8\text{MB}$

(d) i. some

   ii. all

iii. none

iv. none

(e) Because it will take so long in relative terms to bring the page from disk into main memory, that would be inefficient not to use the processor in the interim to run another process.

(f) If a process could write to this register it would break the protections of virtual memory by allowing the process to set up and use alternate tables that could access any physical memory.

Name:________________________________________

8. *Memory Allocation* (**11** points) Consider the (tiny) heap below using an implicit free-list allocator with coalescing, where each rectangle represents 4 bytes and a header or boundary tag of the form `x|y` indicates a block of size `x` (in base-10) where `y` being `1` means allocated. Addresses (in base-10) are written below the rectangles. Unlike your Lab 5 allocator, the allocator for this heap uses boundary tags for allocated blocks.

| | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 16\|1 | 12 | 27 | 16\|1 | 20\|0 | 9 | 10 | 15 | 20\|0 | 12\|1 | 42 | 12\|1 | 12\|1 | 43 | 12\|1 | 12\|0 | 42 | 12\|0 |
| 64 | 68 | 72 | 76 | 80 | 84 | 88 | 92 | 96 | 100 | 104 | 108 | 112 | 116 | 120 | 124 | 128 | 132 |

(a) Suppose the next call to the allocator is `free(104)`. Show the state of the heap after this call completes by indicating in the picture of the heap above what addresses have different contents and what the new contents are.

(b) Suppose the next allocator call (after part (a)) is `malloc(4)`. Under a *first-fit policy*, what address would the call to `malloc` return?

(c) Suppose the next allocator call (after part (a)) is `malloc(4)`. Under a *best-fit policy*, what address would the call to `malloc` return?

(d) Given your answer to part (a) (i.e., after doing this free), what is the smallest `z` such that `malloc(z)` would not succeed unless the allocator expanded the size of the heap?

**Solution:**

(a)

| | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 16\|1 | 12 | 27 | 16\|1 | 32\|0 | 9 | 10 | 15 | 20\|0 | 12\|1 | 42 | 32\|0 | 12\|1 | 43 | 12\|1 | 12\|0 | 42 | 12\|0 |
| 64 | 68 | 72 | 76 | 80 | 84 | 88 | 92 | 96 | 100 | 104 | 108 | 112 | 116 | 120 | 124 | 128 | 132 |

(b) 84

(c) 128

(d) 25 (partial credit for 28)

Name:____________________________________________

9. *Memory Bugs* (**13** points) Consider this buggy C code involving a singly-linked list of ints:

```
struct IntList {
  int i;
  struct IntList * next;
};

// insert x at beginning of list and return pointer to new head of list
struct IntList * add_at_front(int x, struct IntList * lst) {
  struct IntList * ans = (struct IntList*)malloc(sizeof(struct IntList*));
  ans->next = lst;
  ans->i = x;
  return lst;
}

// deallocate all list nodes (using recursion)
void free_entire_list(struct IntList * lst) {
  if(lst != NULL) {
    free(lst->i);
    free(lst);
    free_entire_list(lst->next);
  }
}
```

(a) `add_at_front` has two bugs. Rewrite the function to fix both.

(b) `free_entire_list` has two bugs. Rewrite the function to fix both.

(c) Consider the memory allocator from Lab 5. Which of the four bugs you fixed, while still a bug in C, would happen not to cause problems when using this memory allocator? Explain your answer.

*NOTE: When we gave the exam,* `free_entire_list` *had three bugs because we accidentally and embarrassingly also left out the base case (testing for* `lst` *being* `NULL`*), so it would always go into an infinite loop. We clarified during the exam, "oops there are 3 bugs, please fix all three" and then in grading we gave full credit in part (b) for fixing any 2 of the 3 bugs.*

**Solution:**

(a)
```
struct IntList * add_at_front(int x, struct IntList * lst) {
  struct IntList * ans = (struct IntList*)malloc(sizeof(struct IntList));
  ans->next = lst;
  ans->i = x;
  return ans;
}
```

(b)
```
void free_entire_list(struct IntList * lst) {
  if(lst != NULL) {
    free_entire_list(lst->next);
    free(lst);
  }
}
```

(c) The extra `*` inside the `sizeof` would not be an issue because it causes the argument to `malloc` to be 8 instead of 16. But the allocator never makes blocks with fewer than 16 bytes for payload anyway, so even though we asked for too little memory, we will be given enough.

Name:________________________________________

10. *C vs. Java* (**11** points) Consider this Java code (left) and somewhat similar C code (right) running on x86-64:

```
public class Foo {
  private int[] x;
  private int y;
  private int z;
  private Bar b;
  public Foo() {
     x = null;
     b = null;
  }
}
```

```
struct Foo {
  int x[6];
  int y;
  int z;
  struct Bar * b;
};

struct Foo * make_foo() {
  struct Foo * f = (struct Foo *)malloc(sizeof(struct Foo));
  f->x = NULL;
  f->b = NULL;
  return f;
}
```

(a) In Java, `new Foo()` allocates a new object on the heap. How many bytes would you expect this object to contain for holding `Foo`'s fields? (Do *not* include space for any header information, vtable pointers, or allocator data.)

(b) In C, `malloc(sizeof(struct Foo))` allocates a new object on the heap. How many bytes would you expect this object to contain for holding `struct Foo`'s fields? (Do *not* include space for any header information or allocator data.)

(c) The function `make_foo` attempts to be a C variant of the `Foo` constructor in Java. One line fails to compile. Which one and why?

(d) What, if anything, do we know about the values of the `y` and `z` fields after Java creates an instance of `Foo`?

(e) What, if anything, do we know about the values of the `y` and `z` fields in the object returned by `make_foo`?

**Solution:**

(a) 24

(b) 40

(c) `f->x = NULL` does not compile. In C, the field declaration `int x[6]` creates an inline array, not a pointer, so it does not make any sense to "assign NULL to the array" — the struct itself has slots for six array elements.

(d) We know both fields hold 0.

(e) We know nothing. (We know something abou their size, but not their contents – it could be any bit-pattern.)

Name:_________________________________________

11. *vtables* (**6** points)

    (a) In an implementation of Java, can a vtable for a subclass ever have more entries than a vtable for a superclass? Explain in 1–2 English sentences.

    (b) In an implementation of Java, can a vtable for a subclass ever have the same number of entries as a vtable for a superclass? Explain in 1–2 English sentences.

    (c) In an implementation of Java, can a vtable for a subclass ever have fewer entries than a vtable for a superclass? Explain in 1–2 English sentences.

**Solution:**

(a) Yes. If the subclass defines methods not defined in the superclass, then it will have a larger vtable because there is one entry for each method defined in the class (including methods inherited).

(b) Yes. If a subclass does not define any new methods – only inherits or overrides methods – then its vtable will be the same size as the vtable for the superclass.

(c) No. A subclass always has defined at least all the methods defined in the superclass, so its vtable must be at least as large.