Spring Semester 2018

KAIST EE209

Programming Structures for Electrical Engineering

Mid-term Exam

Name: ______________________________

Student ID: ______________________________

This exam is open book and notes, but closed electronic device. Read the questions carefully and focus your answers on what has been asked. You are allowed to ask the instructor/TAs for help only in understanding the questions, in case you find them not completely clear. Be concise and precise in your answers and state clearly any assumption you may have made. You have 165 minutes (1:00 PM – 3:45 PM) to complete your exam. Be wise in managing your time. Good luck.

Question 1	/ 10
Question 2	/ 20
Question 3	/ 15
Question 4	/ 20
Question 5	/ 20
Question 6	/ 15
Total	/ 100

Name: Student ID:

1. (10 points) Numbers

   (a) (6 points) What is the range of decimal numbers that can be represented using 7 bits for each format?

   2's complement:_____-64_______ to _____63_______

   1's complement:_____-63______ to _____63_______

   Unsigned binary:_____0________ to _____127______

   (a) (2 points) Convert the 8-bit signed 2's complement hex number 0xAB to decimal:

   Answer: -85

   (b) (2 points) Compute the decimal value of -12 ^ 23 using 8-bit 2's complement encoding:

   Answer: -29

Name: Student ID:

2. (20 points) Small programs

(a) (10 points) Identify all the bugs in the following program.

```
#include <stdio.h>

struct student {
  char name[3];
  int counter;
};

void increment(const struct student* s)
{
  s.counter++;
}

int main()
{
  struct student s;
  strcpy(s.name, "kim");
  s.counter = 0;
  increment(s);
  return 0;
}
```

Answer:

1. `#include <string.h>` is missing
2. The `name` array should contain more than 3 characters to include '\0'
3. In the `increment` function, the `const` should be removed to modify `s`
4. In the `increment` function, `s.counter` should be `s->counter` because `s` is a pointer
5. In the `main` function, `increment(s)` should be `increment(&s)` in order to pass a pointer

Name: Student ID:

(b) (10 points) What is the output of the following program?

```
#include <stdio.h>
#include <string.h>

int main()
{
  char s[] = "lbjtu\0abc", *p;
  for (p = s; *p; p++)
    --*p;
  printf("s: %s\n", s);
  printf("s + 6: %s\n", s + 6);
  printf("strlen(s): %zu\n", strlen(s));
  printf("strlen(s + 6): %zu\n", strlen(s + 6));
  printf("sizeof(s): %zu\n", sizeof(s));
  return 0;
}
```

Output:

s: _____kaist____________

s + 6: _____abc______________

strlen(s):_____5________________

strlen(s + 6):_____3________________

sizeof(s):_____10_______________

Name: Student ID:

3. (15 points) Functions

Write a function that finds the n^{th} fibonacci number. The Fibonacci series are the integers in the following sequence:

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, …

where the first two integers in the sequence are 0 and 1, and each subsequent integer is the sum of the previous two integers. For example, the 0^{th} fibonacci number is 0, and the 4^{th} fibonacci number is 3.

(a) (5 points) Implement the Fibonacci function using recursion. Assume input `n` is a non-negative integer.

```
int fibonacci_recursive(int n)
{
  if(n < 2)
    return n;
  return fibonacci_recursive(n-1) + fibonacci_recursive(n-2);
}
```

(b) (10 points) Implement the Fibonacci function without using recursion. Assume input `n` is a non-negative integer.

```
int fibonacci_iterative(int n)
{
  int n1 = 0, n2 = 1, r;
  for (int i = 0; i <= n; i++) {
    if (i < 2)
      r = i;
    else
    {
      r = n1 + n2;
      n1 = n2;
      n2 = r;
    }
  }
  return r;
}
```

Name: Student ID:

4. (20 points) Dynamic storage

Suppose we are dynamically allocating many blocks of memory. While we can free each allocated block individually, it may be more convenient to free all of them together through a single function. To implement this functionality, we will maintain a dynamically-allocated array `p` of pointers to all the allocations using these two functions:

- *mymalloc*: allocates a memory block and add its pointer to `p`
- *myfree*: frees all the allocated memory blocks using `p`

You may use the following functions and assume that memory is always available.

```
void *malloc(size_t size);
void *realloc(void *ptr, size_t size);
void free(void *ptr);
```

Fill in the lines below:

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

void* mymalloc(size_t size, void*** p, int* psize)
{

  *p = ___ realloc(*p, (*psize + 1) * sizeof(void*));__

  void *temp = _____malloc(size);________________

  _____*(*p + *psize) = temp;_______________

  _____(*psize)++;_____________________

  return temp;
}
```

```
void myfree(void **p, int psize)
{

  for (int i = 0; i < psize; ++i)
  {


    _____ free(*(p+i));_____________________
  }


  _____ free(p);_______________________
}


int main()
{
  void** p = NULL;
  int psize = 0;
  mymalloc(sizeof(int), &p, &psize);
  mymalloc(sizeof(char), &p, &psize);
  myfree(p, psize);
  return 0;
}
```

Name: Student ID:

5. (20 points) Linked list

In addition to the linked list functions covered in class (`Table_create`, `Table_add`, `Table_search`, and `Table_free`), implement functions for updating and deleting individual nodes. Duplicates keys may exist, and the following structs are used:

```
struct Node {
   const char *key;
   int value;
   struct Node *next;
};

struct Table {
   struct Node *first;
};
```

(a) (10 points) Implement the update function, which finds all the nodes with the given `key` and changes their values to the given `value`.

```
void Table_update(struct Table *t, const char *key, int value)
{
  struct Node *p;
  for (p = t->first; p != NULL; p = p->next)
    if (strcmp(p->key, key) == 0) {
      p->value = value;
    }
  return;

}
```

(b) (10 points) Implement the delete function, which finds all the nodes with the given `key` and deletes them. Make sure the remaining nodes still form a linked list.

```
void Table_delete(struct Table *t, const char *key)
{
  // Handle the case when t->first has the same key
  while (t->first && strcmp(t->first->key, key) == 0) {
    struct Node *q = t->first;
    t->first = t->first->next;
    free(q);
  }
  for (struct Node *p = t->first; p != NULL;) {
    if (p->next && strcmp(p->next->key, key) == 0) {
      struct Node *q = p->next;
      p->next = q->next;
      free(q);
    } else {
      // Only advance p if key was not found
      p = p->next;
    }
  }
  return;
}
```

Name: Student ID:

6. (15 points) C++

(a) (10 points) What is the output of the following program? Briefly explain why each line is printed.

```
#include <iostream>

using namespace std;

class B
{
 public:
  B() {
    cout << "B()" << endl;
  }
  ~B() {
    cout << "~B()" << endl;
  }
  void f() {
    cout << "B::f()" << endl;
  }
  virtual void vf() {
    cout << "B::vf()" << endl;
  }
};



class D : public B
{
 public:
  D() {
    cout << "D()" << endl;
  }
  ~D() {
    cout << "~D()" << endl;
  }
  void f() {
    cout << "D::f()" << endl;
  }
  void vf() {
    cout << "D::vf()" << endl;
  }
};
```

```
int main()
{
  B b;
  D d;
  D* e = (D*)&b;
  d.f();
  d.vf();
  e->f();
  e->vf();
}
```

Output:

B() ➔ When declaring b, B’s constructor is called
B() ➔ When declaring d, B’s constructor is called first
D() ➔ … then D’s constructor is called
D::f() ➔ In d.f(), D’s f function is called
D::vf() ➔ In d.vf(), D’s vf function is called
D::f() ➔ In e->f(), D’s f function is called (static binding)
B::vf() ➔ In e->vf(), B’s vf function is called (dynamic binding)
~D() ➔ When d goes out of scope, D’s destructor is called first
~B() ➔ … then B’s destructor is called
~B() ➔ When b goes out of scope, B’s destructor is called

Name: Student ID:

(b) (5 points) What is the output of this program? (Hint: look at the arguments of the swap function carefully.)

```
#include <iostream>
using namespace std;

void swap(int& a, int b)
{
  int temp = a;
  a = b;
  b = temp;
}

int main()
{
  int a = 1;
  int b = 2;
  int c = 3;

  swap(a,b);
  cout << "swap(a,b):" << a << " " << b << " " << c << endl;

  swap(b,c);
  cout << "swap(b,c):" << a << " " << b << " " << c << endl;

  swap(c,a);
  cout << "swap(c,a):" << a << " " << b << " " << c << endl;
}
```

Output:

swap(a,b): ______2 2 3__________

swap(b,c): ______2 3 3__________

swap(c,a): ______2 3 2__________