CS100 Mid-Term Exam Cover Sheet

Instructors: Laurent Kneip, Xiaopei Liu
November 8, 2019

PRINT your name: (last name)_____, (first name)_____

PRINT your email ID: _____@shanghaitech.edu.cn

PRINT your student ID:_____

PRINT your tutorial group number (or TA name, or tutorial time):_____

INSTRUCTIONS:

- You have 95 minutes (~10:20am-11:55am) to complete the exam.
- Your exam will not be graded unless you complete the above section and the cover sheet, and turn in both this exam book and the cover sheet.
- This exam is **closed-book and closed-notes**, and **no electronic devices are permitted**.
- Mark your answers on the exam itself. We will not grade answers written on scratch paper.
- Your performance is supposed to reflect your own level of understanding of the material. You are not allowed to talk with your neighbor or look at his exam sheet. Failure to obey this rule will simply result in a **zero score**.
- If you finish early, you can hand in the exam and leave early. However, this is only possible until at latest 15 minutes before the ending time of the exam. If less than 15 minutes are left, please stay seated and wait for the end.

STOP! Do not turn this page until the instructor tells you to do so.

Do NOT write in this section.

Problem	Max	Points
1	24	
2	24	
3	36	
4	16	
Total	100	

Important: Verify that your exam book has 18 pages.

Problem 1: Multiple-choice questions (24 points)

Please answer the following questions by ticking the choices that apply. Each question has 2 points, and is clearly marked as a C or C++ question. Note that multiple choices are possible for each question. Mark all possible choices that apply by ticking the corresponding box ☑ .

Question 1 (C): Given two integers **a** and **b**, and the expression **(((a++) * ++b) + 5)-(a&&b)**, which of the following statements is true?

- ☐ if **a**=1 and **b**=2, the expression evaluates to 6;
- ☑ if **a**=1 and **b**=2, the expression evaluates to 7;
- ☑ if **a**=2 and **b**=3, the expression evaluates to 12;
- ☐ if **a**=2 and **b**=3, the expression evaluates to 13;

Question 2 (C): You are given a floating point variable **x** which is used in the statement "**printf("%+10.5f", x);**". Mark all answers that apply! (we use "_" to indicate a spacing)

- ☐ if **x**=4.67, then the printed result is: **___4.67**
- ☑ if **x**=4.67, then the printed result is: **__+4.67000**
- ☐ if **x**=-4.67, then the printed result is: **_+-4.67000**
- ☑ if **x**=-4.67, then the printed result is: **__-4.67000**

Question 3 (C): Consider the below recursive function call. Mark all correct answers!

```
int output(char c){
    if (c == 'a'){
        printf("%c", c);
        return -1;
    }
    printf("%c", c);
    if (output(c-1)) printf("%c", c);
    return 1;
}
int main(){
    char c;
    scanf("%c", &c);
    output(c);
    return 0;
}
```

- ☐ if **c**='e', then the output is: **abcdedcba**
- ☑ if **c**='e', then the output is: **edcbabcde**
- ☑ if **c**='h', then the output is: **hgfedcbabcdefgh**
- ☐ if **c**='h', then the output is: **hgfedcbbcdefgh**

Question 4 (C): Suppose we define a structure for a 2D point:

```
typedef struct {
    float x;
    float y;
} point;
```

It is used in the following program. Mark all answers that apply!

```
int main(){
    int i=0, n=0;
    scanf("%d", &n);
    point* p = (point*)malloc(sizeof(point)*n);
    point* p_temp = p;
    for (i = 0; i < n; i++){
        p->x = (float)i;
        (p++)->y = (float)(i+1);
    }
    printf("x=%f, y=%f", (p-2)[0].x, (p-2)[1].y);
    free(p_temp);
    return 0;
}
```

- ☐ if **n**=9, the output will be: **x=8, y=9**
- ☐ if **n**=9, the output will be: **x=7, y=9**
- ☐ if **n**=10, the output will be: **x=8, y=10**
- ☐ if **n**=10, the output will be: **x=7, y=10**

Question 5 (C): Consider the following program:

```
int main() {
    int i = 0;
    char str1[] = "This is good!";
    char str2[] = "Yes!";
    char str[32];
    memset(str, 0, sizeof(str));
    for (i = 0; i < strlen(str1); i++)
        str[i] = str1[i];
    i++; //optional statement
    for (; i < strlen(str1)+strlen(str2); ++i)
        str[i] = str2[i-strlen(str1)];
    printf("%s", str);
    return 0;
}
```

Note that "**`i++;`**" is an optional statement. Which of the following statements is true?

- ☐ if "**`i++;`**" is not commented out, the program will output "**`This is good!Yes!`**".
- ☐ if "**`i++;`**" is not commented out, the program will output "**`This is good!`**".
- ☐ if "**`i++;`**" is commented out, the program will output "**`This is good!`**".
- ☐ if "**`i++;`**" is commented out, the program will output "**`This is good!Yes!`**".

Question 6 (C): Consider the following program:

```
int main() {
    char* str = (char*)malloc(sizeof(char) * 32);

    *str = 'y'; str = str+1;
    *str = 'e'; str += 1;
    *str = 's'; str += 1;
    *str = '\0';

    printf("%s", str-3);
    free(str);

    return 0;
}
```

Which of the following answers is true?

- ☐ The program will output "**`yes`**" and terminate correctly.
- ☐ The program will output "**`yes`**" correctly but crash when before exiting.
- ☐ If "**`*str = '\0';`**" is not written, the program will produce an unpredictable output.
- ☐ The program will not print any alphanumeric characters if "**`printf("%s", str-3);`**" is changed into "**`printf("%s", str);`**"

Question 7 (C): Consider the following program:

```
union data{
    int x;
    struct
    {
        char x1, x2, x3, x4;
    };
};
```

```
int main() {

    data d;
    d.x3 = 0;
    d.x4 = 0;

    int x1, x2;
    scanf("%d", &x1);
    scanf("%d", &x2);
    d.x1 = x1; d.x2 = x2;
    printf("%d", d.x);

    return 0;
}
```

Which of the following statements is true?

- ☐ If **x1**=4, **x2**=3, the printed output is: **34**
- ☐ If **x1**=4, **x2**=3, the printed output is: **772**
- ☐ If **x1**=1, **x2**=4, the printed output is: **14**
- ☐ If **x1**=1, **x2**=4, the printed output is: **1025**

Question 8 (C): Consider the following program:

```
void process(int* data, int num, int delta){
    if (num <= 0)
        return;

    for (int i = 0; i < num; i++)
        *(data++) = i + 1;
    process(data - (num - delta), num - delta, delta);
}

int main() {
    int i, delta, num=9;
    scanf("%d", &delta);
    int* data = (int*)malloc(sizeof(int) * num);

    process(data, num, delta);
    for (i = 0; i < num; i++)
        printf("%d", data[i]);
    free(data);
    return 0;
}
```

Which one of the following statements is true?

- [x] If **`delta`**=1, the program will output: **`111111111`**
- [ ] If **`delta`**=1, the program will output: **`123456789`**
- [x] If **`delta`**=2, the program will output: **`121212121`**
- [ ] If **`delta`**=2, the program will output: **`212121212`**

Question 9 (C++): This question is about include-guards. Mark all statements that are true!

- [x] An include-guard prevents double definition if a header is included more than once.
- [ ] An include-guard goes into a .cpp-file and not a header file.
- [x] Include-guards are processed by the pre-processor, and not the compiler.
- [ ] Include-guards are regular instructions that are executed at run-time.

Question 10 (C++): This question is about the "aggregation" object relationship. Mark all statements that are true!

- [x] An aggregation of an object is typically done via a pointer.
- [x] An aggregated object may be aggregated by multiple different objects of different classes.
- [ ] An aggregated object's life-time is limited by the life-time of the object that owns it.
- [ ] An object of a derived class may not be aggregated.

Question 11 (C++): Abstract base classes. Mark all statements that are true!

- [ ] An abstract base class has no virtual methods.
- [x] An abstract base class is one that has purely virtual functions.
- [x] An abstract base class has at least one method for which the declaration ends with … **`= 0;`**
- [ ] An abstract base class may be instantiated.

Question 12 (C++): STL containers/algorithms. Mark all statements that are true.

- [x] For **`std::vector`**, element access has constant computational complexity.
- [ ] For **`std::list`**, insertion with an iterator has a (worst-case) complexity linear in its size.
- [x] STL-containers are templated by the type of the contained elements.
- [x] **`std::sort`** needs random access iterators.

Problem 2: Find and correct problems and bugs (24 points)

Question 1 (C, 12 points): There are several errors in the following C code. Find at least 4 of them. Briefly explain what the error is by mentioning the line, the problem, and a correct replacement for that entire line.

```
/* The following C program creates an array of records. For each record,
it stores the name, age, grade and score of a student. In the program, it
asks the user to input all the information from the students, and then
prints the student information. The functions input_record() and
print_record() are used for reading and printing the related information.
*/

1. typedef struct{
2.     char name[32];
3.     unsigned int age;
4.     unsigned int grade;
5.     float score;
6. } record;
7.
8. record* create_record(int num){
9.     record* r_ret = (record*)malloc(sizeof(record) * num);
10.    memset(r_ret, 0, sizeof(record)* num);
11.    return r_ret;
12.}
13.
14.void destroy_record(record* r){
15.    free(r);
16.}
17.
18.void input_record(record** r, int num){
19.    for (int i = 0; i < num; i++){
20.        printf("record %d:\n", i + 1);
21.        printf("name:");
22.        scanf("%s", (*r++)->name);
23.        printf("age:");
24.        scanf("%d", (*r++)->age);
25.        printf("grade:");
26.        scanf("%d", (*r++)->grade);
27.        printf("score:");
28.        scanf("%f", (*r++)->score);
29.        printf("\n\n");
30.    }
31.}
32.
```

```
33.void print_record(record* r, int num){
34.     for (int i = 0; i < num; i++){
35.         printf("record %d:\n", i + 1);
36.         printf("name: %s, ", r[i].name);
37.         printf("age: %d, ", r[i].age);
38.         printf("grade: %d, ", r[i].grade);
39.         printf("score: %f\n", r[i].score);
40.         printf("\n");
41.     }
42.     free(r);
43.}
44.
45.int main() {
46.     int record_num = 0;
47.     printf("input number of record:");
48.     scanf("%d", &record_num);
49.     record* r = create_record(record_num);
50.
51.     input_record(&r, record_num);
52.
53.     printf("the input records are:\n");
54.     print_record(r, record_num);
55.
56.     destroy_record(r);
57.     return 0;
58.}
```

Answer (only 4 need to be mentioned):

Line 1: Include missing. Correction `include <stdio.h>`

Line 22: Pointer should not be incremented. Correction: `scanf("%s", (*r)->name);`

Line 24: Pointer should not be incremented, and we should pass a pointer to the age variable. Correction: `scanf("%d", &(*r)->age);`

Line 26: Pointer should not be incremented, and we should pass a pointer to the grade variable. Correction: `scanf("%d", &(*r)->grade);`

Line 28: We should pass a pointer to the score variable. Correction: `scanf("%f", &((*r)++)->score);`

Lines 42 / 56: Double free of the array r. Correction: remove line 42.

Line 54: Pointer was incremented through function call, so the delete operates on wrong memory location. Correction: 1) decrement pointer, or 2) some may change the entire function signature to not pass `record** ptr`, but `record *ptr`

Question 2 (C++, 12 points): There are 4 errors in the following C++ code. Find all of them. Briefly explain what the error is by mentioning the line, the problem, and a correct replacement for that entire line.

```
1.  #include <list>
2.  #include <iostream>
3.  #include <algorithm>
4.
5.  namespace std;
6.
7.  int main() {
8.      int input;
9.      list<int> ilist;
10.
11.     // input
12.     int index = 0;
13.     while (cin >> input ) {
14.         ilist[index++] = input;
15.     }
16.
17.     // sorting
18.     sort(ilist.begin(),ilist.end());
19.
20.     // output
21.     list<int>::iterator it;
22.     for ( it = ilist.begin();
23.           it != ilist.end(); ++it ) {
24.         cout << it << " ";
25.     }
26.     cout << endl;
27.     return 0;
28. }
```

Answer:

Line 5: There should not be a namespace definition here. Correction: `using namespace std;`

Line 14 (and 12): It is not possible to index a list as in the example, especially as the list is actually empty. Correction: `ilist.push_back(input);`

Line 18: The std::sort function can't be used with std::list. Correction: `ilist.sort();`

Line 24: The iterator is not dereferenced. Correction: `cout << *it << " ";`

Problem 3: Coding (C,36 points)

Note: Problem 3 consists of 3 parts. Please first read all the instructions until the end to also see an example use of the functions for which you have to provide the implementation, plus some useful tips.

a) Write two functions "**create_array(…)**" and "**destroy_array(…)**" to construct/destroy a dynamic array whose capacity is dynamically increased when needed. The array is represented by the following structure:

```
typedef struct{
    float* data;   //data pointer to store the array
    int size;      //effective number of elements in the array
    int capacity; //maximum array size (i.e. allocated space)
} array;
```

The array construction and destruction functions that you need to write have the signature:

```
bool create_array(array* a);
void destroy_array(array* a);
```

Note that **a** is created before the "**create_array**" function call. "**create_array**" only allocates the initial space. To specify the capacity of the array, the user sets "**a->capacity**" before passing **a** to the function "**create_array**". "**create_array**" needs to check if **a** is a valid pointer, if **a->capacity** has a proper value, and whether or not the memory allocation succeeded. It also needs to initialize **a->size**, and it should return **true** if the creation succeeded, and **false** otherwise. Similarly, "**destroy_array**" needs to check if **a** is a valid pointer, and it should reset all member variables of **a** after the memory is properly deallocated.

```
bool create_array(array* a){
    if (a == NULL || a->capacity <= 0)
        return false;
    a->data = (float*)malloc(sizeof(float) * a->capacity);
    if (a->data == NULL)
        return false;

    a->size = 0;
    return true;
}
```

```
void destroy_array(array* a){
    if (a != NULL){
        free(a->data);
        a->data = NULL;
        a->size = 0;
        a->capacity = 0;
    }
}
```

b) Once the dynamic array is created, it will be used in a function called "**input_data(…)**" that reads numbers inputted by the user one-by-one and stores them in the array. The function has the simple signature:

```
void input_data(array* a);
```

The function should return if the user inputs the character 'q' instead of a number. Note that when the array capacity is insufficient, the function should dynamically increase the array's capacity by 10 elements. The function should properly manage all member variables of **a**, and print an error message and return if an eventual increase of capacity does not succeed.

```
void input_data(array* a){
    printf("Please input the data below:\n");
    while (1){
        char data_str[32];
        scanf("%s", data_str);
        if (strcmp(data_str, "q") == 0)
            break;
        float data = (float)atof(data_str);
        int newSize = a->size + 1;
        if (newSize > a->capacity){
            a->data = (float*)realloc(a->data, sizeof(float) * (a->capacity + 10));
            if (a->data == NULL){
                printf("No sufficient memory!\n");
                break;
            }
            a->capacity += 10;
            a->data[newSize - 1] = data;
            a->size = newSize;
        }
        else{
            a->data[newSize - 1] = data;
            a->size = newSize;
        }
    }
}
```

c) Given an **array** struct, convert all the floating point numbers into an array of strings, and print them. The array of strings is represented by another structure as:

```
typedef struct{
    char** str;  //the array pointer of strings
    int size;    //the number of strings
} string_array;
```

string_array is dynamically created in the string conversion function, which has the signature

```
void convert_to_string(array* a, string_array* str_out);
```

The function needs to check if a is a valid pointer, if the size of **a** is a valid number, and if memory allocation succeeded. The string printing function has the signature:

```
void print_string(string_array* str);
```

Finally, write the following function that destroys a **string_array**:

```
void destroy_string(string_array* str);
```

```
void convert_to_string(array* a, string_array* str_out){
    if (a == NULL)
        return;
    if (a->size <= 0){
        str_out->str = NULL;
        str_out->size = 0;
        return;
    }
    str_out->str = (char**)malloc(sizeof(char*) * a->size);
    if (str_out->str == NULL)
        return;
    str_out->size = a->size;

    for (int i = 0; i < str_out->size; i++){
        str_out->str[i]= (char*)malloc(sizeof(char*) * 32);
        sprintf(str_out->str[i], "%f", a->data[i]);
    }

    return;
}
```

```
void print_string(string_array* str){
    if (str == NULL)
        return;

    for (int i = 0; i < str->size; i++)
        printf("%d. %s\n", i+1, str->str[i]);
}

void destroy_string(string_array* str){
    if (str == NULL)
        return;

    for (int i = 0; i < str->size; i++)
        free(str->str[i]);
    free(str->str);
}
```

Note: You only need to write the above six function bodies; The "**`main()`**" function is as follows:

```
int main(){
    array data;
    data.capacity = 20;
    if (!create_array(&data)){
        printf("Unable to create the array!\n");
        return -1;
    }

    input_data(&data);

    string_array data_str;
    convert_to_string(&data, &data_str);
    print_string(&data_str);

    destroy_array(&data);
    destroy_string(&data_str);

    return 0;
}
```

Tips: The following functions may be useful:

```
void* malloc (size_t size);
void* realloc (void* ptr, size_t size);
int strcmp (const char* str1, const char* str2 );
double atof (const char* str);
int sprintf (char* str, const char* format, ... );
```

Problem 4: Coding (C++,16 points)

Suppose you are given the below definition of a matrix class. Provide a new definition of this class which is templetized by the type of the contained values. Provide the implementation of all the member functions and the constructor/destructor.

Notes: Make sure the matrix is initialized to 0. The functions **do not** need to perform any size checks, you may assume that the user always passes correct/compatible function parameters.

```
class Matrix {
public:
    Matrix( int r, int c );
    virtual ~Matrix();

    int rows();
    int cols();

    double & operator()(int r, int c);
    void operator=( Matrix & rhs );
    Matrix operator+( Matrix & rhs );
    Matrix operator-( Matrix & rhs );
    Matrix operator*( Matrix & rhs );

private:
    std::vector< std::vector<double> > m_data;
};
```

```
template<typename T>
class Matrix {
public:
  Matrix( int r, int c );
  virtual ~Matrix();

  int rows();
  int cols();
  T & operator()(int r, int c);
  void operator=( Matrix<T> & rhs );
  Matrix<T> operator+( Matrix<T> & rhs );
  Matrix<T> operator-( Matrix<T> & rhs );
  Matrix<T> operator*( Matrix<T> & rhs );

private:
  std::vector< std::vector<T> > m_data;
};
```

```
template<typename T>
Matrix<T>::Matrix(int r, int c) {
  m_data.resize(r);
  for( int i = 0; i < r; i++ )
    m_data[i].resize(c,(T)0);
}

template<typename T>
Matrix<T>::~Matrix(){}

template<typename T>
int Matrix<T>::rows() {
  return m_data.size();
}

template<typename T>
int Matrix<T>::cols() {
  return m_data[0].size();
}

template<typename T>
T & Matrix<T>::operator()(int r, int c) {
  return m_data[r][c];
}

template<typename T>
void Matrix<T>::operator=( Matrix<T> & rhs ) {
  m_data = rhs.m_data;
}

template<typename T>
Matrix<T> Matrix<T>::operator+( Matrix<T> & rhs ) {
  Matrix<T> res(rows(),cols());
  for( int r = 0; r < rows(); r++ ) {
    for( int c = 0; c < cols(); c++ )
      res(r,c) = (*this)(r,c) + rhs(r,c);
  }
  return res;
}
```

```
template<typename T>
Matrix<T> Matrix<T>::operator-( Matrix<T> & rhs ) {
  Matrix<T> res(rows(),cols());
  for( int r = 0; r < rows(); r++ ) {
    for( int c = 0; c < cols(); c++ )
      res(r,c) = (*this)(r,c) - rhs(r,c);
  }
  return res;
}

template<typename T>
Matrix<T> Matrix<T>::operator*( Matrix<T> & rhs ) {
  Matrix<T> res(rows(),rhs.cols());
  for( int r = 0; r < rows(); r++ ) {
    for( int c = 0; c < rhs.cols(); c++ ) {
      res(r,c) = (T) 0;
      for( int k = 0; k < cols(); k++ )
        res(r,c) += (*this)(r,k) * rhs(k,c);
    }
  }
  return res;
}
```

(end of exam)