CSC 340.03+.04 MIDTERM EXAM 01 SPRING 2022 TA

MIDTERM EXAM INSTRUCTIONS

1. Midterm Exam: **25 points w/ 0 E.C. points**
2. Due Date & Time: **03-21-2022 at 05:00 AM**

WHAT TO SUBMIT

1. Take-home Exam Report, 1 PDF

HOW TO SUBMIT AND THE RULES TO FOLLOW

- Submit via iLearn, the Assignment and Exam Submission section
- Please follow the exam instructions
- Please follow the Course Policy on Student Conduct and Academic Honesty

PERFORMANCE TRACKER		
ASMT	**GRADE**	**YOUR GRADE**
ZOOM	05	
01	15	
02	100	
03	100	
MIDTERM 01	25	
TOTAL	245	

A: 90-100% **B**: 80-89% **C**: 70-79% **D**: 60-69% **F**: 0-60%
The course grader provides feedback to your assignments on iLearn.

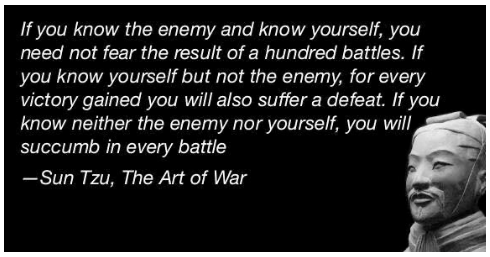
ABOUT

The goal of this take-home exam is for us to **know what we do not know**.

We are taking this exam as seriously as how we take an actual exam in class. Please,

1. Follow all the rules and the guidelines listed at the top of page 1 and page 2
2. Read each question carefully before answering

We will go through the answers to all the exam questions together in class.



If you know the enemy and know yourself, you need not fear the result of a hundred battles. If you know yourself but not the enemy, for every victory gained you will also suffer a defeat. If you know neither the enemy nor yourself, you will succumb in every battle

—Sun Tzu, The Art of War

STEP A – Take the Exam, **10 points**

1. Allocate 50 quiet minutes to take the exam on page 2 to the last page.
2. Record the date and time when you start.
3. Stop right at minute 50. Record the date and time when you stop the exam.

```
EXAM
                    Full Name | SFSU ID
- Each question
      - Answer, Step A
      - Corrected Answer, Step B
      - More practice, Step C
      - Other notes

- Next question
```

STEP B – Correct Your Answers, **10 points**

1. Review the related course materials and write code when necessary to find a correct answer for each question. We should be able to find all the answers using the packages, the in-class discussions, our assignments, and the other course materials.
2. At the end of each of your oringal answers, type in *italic* text and:
   - Give your orginal answer a score.
   - List all the mistakes then explain why, you think, you made the mistakes. Add the correct answer you found. Document how you found the correct answer. Document where you found the materials which support the answer.
   - If you did not make any mistakes, please document how you verified that your answer was accurate. Document how you found the correct answer. Document where you found the materials which support the answer. Outline how you could have done better.
   - Record your total score out of 100 points for all the orginal answers.

STEP C – Reflect and Retake the Exam, **5 points**

1. **Problem Solving**: Reflect if you managed the exam time efficiently and if you strategized your test-taking successfully.
2. Repeat steps A to C again if necessary. Please keep appending new contents as directed in Step B.2.
3. Think if the same topics will be tested again in our second midterm or final exam, what questions we may get.

It is a good idea to do every step of this exam thoroughly. We are creating a set of materials which we will use to review for the 2nd midterm exam and the final exam. And this is also the best way to prepare ourselves to succeed in the second half of the semester. Thank you.

Kullathon "Mos" Sitthisarnwattanachai
921425216

Step A

- See the following pages for the exam attempts.
- Started: 3/19 09:11 PT
- Finished: 3/19 10:01 PT

Step B

- Correction and evaluations are listed at the end of each questions in a box.
- Total score: 63/100

Question	Scored	Total
A.1	**8**	**10**
A.2	**10**	**10**
A.3	**15**	**20**
A.4	**0**	**0**
A.5	**0**	**0**
A.6	**30**	**30**
A.7	**0**	**30**
B.1	**0**	**0**
B.2	**0**	**0**
B.3	**0**	**0**
Total	**63**	**100**

Step C

- This exam was quote heavy on code tracing, which takes quite a lot of time. I think my time management on this exam has been very poor, as I was unable to complete one question despite skipping all of the non-credit ones.
- My prioritization of the questions were also not very efficient. I spent way too much time on questions A.1 and A.2 which only offered ten points each, one of which I didn't even answer correctly.

  It would have been much more efficient to spend time on answer the thirty-point questions, which, if I had answered both of them correctly, would have already made up as many points that I've scored in total. ßßß

1. Section, Date and Time:
   TIC and TAC, due 03-21-2022 05:00 AM
2. Midterm Exam (2 exams, 0 dropped): 100 points

Full Name in CAPITAL LETTERS	SFSU ID
KULLATHON SITTHISARNWATTANACHAI	921425216

3. To prepare for this exam, please review all the related materials including WEEK 01-05 packages, slides, mock-up exam(s), reading assignments, in-class practices, sample programs posted in the File Manager, and assignments.
4. You do not need to print this exam. No papers. No handwriting. No scanned images. No screenshots. Please type up all your answers in the answer space available in the exam. The provided exam will be in Microsoft Word format. Please submit a single PDF via iLearn.
5. All the rules of an actual exam apply to this exam such as: closed books, closed notes, closed IDEs, and no communication with anyone except the course instructor. The course instructor will be available on Zoom (zoom.ducta.net) or via email during the exam time. You cannot use any other materials or tools but only the provided exam which will be in Microsoft Word format.
6. Please ask all your questions, if any, during the review sessions. Thank you.

HONOR CODE:

- Please follow the CS Department's policies: https://cs.sfsu.edu/student-policies

- Please follow the course's policies: http://csc340.ducta.net/00-README-StudentConduct_AcademicHonesty.pdf

---

PART A – 100 Points

A.1 - 10 pts - How is a **pointer** different from a **reference variable**? And please give a code example.

A pointer is a variable that holds the address of another variable and has an address of its own. Whereas a reference variable is a *reference* to another variable and shares the address of the variable it is referencing.

```
int a = 1;
int *pA = &a  // pA is a pointer to a
int &rA = a    // rA is a reference to a
```

A pointer may be intialized to a null value (nullptr) – i.e., to point to nothing, or declared without being initialized at all. In addition, a pointer may also be reassigned afterwards. Reference variables, on the other hand, must reference another variable upon initialization, and cannot be reassigned.

```
int b = 1;
int *pB;        // pointer declaration only
int &rB;        // illegal declaration
```

The value of the pointer itself is always an integer, irrespective of the type that it is pointing to. The integer represents the memory address of the variable it is pointing. To obtain the value of the variable that the pointer is pointing to, one must deference the pointer. Reference variables, however, behaves the same way as the variable it is referencing. It can be referenced directly, just as one would to the variable it is referencing.

```
int c = 1;
int *pC = &c;
int &rC = c;

std::cout << *pC   // to print value of c, using its pointer it must be deferenced first
std::cout << rC     // with references, the value of c can be accssed directly
```

Would these statements cause an error? Why or why not?

```
int year = 2019;
int yearNext = 2020;
int & ref = year;
ref = yearNext;
```

Yes. Reference variables cannot be reassigned. The forth line attempts to reassign the reference that the third line defined.

Step B	
Original score:	*8/10*
Reasoning and corrections:	*No deductions for the first part.* *Reference: Package 03.* • *This part was discussed at length during the beginning of the Java-to-C++ transition.* *Deducted 2 points from the last part of the question.* *Reference: Tested the provided code snippet.* • *The statement would not cause an error. The fourth line redefines the value of* ref*, not its reference. Therefore, the statements are valid.*
Room for improvement:	• *Read the question properly.* • *Spend less time on this question. This question was only worth 10 points. A significant amont of time was spent on this question, which may have lead to loss of opportunity to score on other high-valued questions.*

A.2 - 10 pts - What is a **dangling/stale pointer**? And please give a code example.

A dangling pointer is a pointer that points to a value that no longer exists on the memory.

```
int a = 5;
int* pA = &a;       // pA is a pointer to A
delete pA;          // this deletes the value pA points to; pA is now a dangling pointer
```

Does **`delete`** delete a pointer? [Yes] **[No]**
Please explain why. Then please give instructions how to properly deallocate an object allocated on Free-Store. Please use a code example to demonstrate.

As noted in the previous part with the code example, the delete keyward deletes the object at which the pointer points to. This results in the pointer to become "stale" or "dangling." As such, one must deallocate the pointer by pointing it to a *nullptr*.

```
int b = 1;
int* pB = &b;
delete pB;          // delete the value it is pointing to
pB = nullptr;       // then, deallocate the pointer itself
```

Step B	
Original score:	*10/10*
Reasoning and corrections:	*No deductions.* • *This answer appears to be accurate according to package 03.*

Room for improvement:	• *N/A*

A.3 - 20 Points

```
#include <iostream>
#include <string>

using namespace std;

string type = "Credit";

class credit_card {
public:
    credit_card() = default;

    explicit credit_card(const double& balance, string com = "Disney") :
        com_(move(com)), balance_(balance) {}

    void display_info() const {
        cout << credit_card::type_ << " [" << this->com_ << "]: " << this->balance_ << endl;
    }

    void set_com(const string& com) {
        this->com_ = com;
    }

private:
    static string type_;
    string com_{ "N/A" };
    double balance_{ 0 };
};

string credit_card::type_ = type;

credit_card& update_credit_card(const double& balance) {
    credit_card cc1{ balance };
    static credit_card* cc2 = new credit_card{ 100 };
    cc1.set_com("Tesla");
    *cc2 = cc1;
    return *cc2;
}

int main()
{
    credit_card cc3 = update_credit_card(300);
    cc3.display_info();

    credit_card cc4 = credit_card{ cc3 };
    cc4.set_com("Zoom");
    cc4.display_info();

    credit_card* cc5 = new credit_card{ update_credit_card(500) };
    cc5->display_info();

    cc3.set_com("Google");
    cc4.display_info();

    return 0;
}
```

For each element listed below, please answer:

a. In which **memory area** is this element stored? Why?

b. The **lifetime**, begin & end, of this element? Why?

- ~~Memory Area 1: Environment~~ *(not tested in this exam)*
- Memory Area **2**: Runtime Stack
- Memory Area **3**: Free-store
- Memory Area **4A**: Uninitialized Data (global, static…)
- Memory Area **4B**: Initialized Data (global, static…)
- ~~*Memory Area 5: Binary Program*~~ *(not tested in this exam)*

type
Which area: [1] [2] [3] [4a] [✓ **4b**] [5]
Why is that area?

The variable is initiazlied and defined at the top of the file, placing it in the global scope.

What is its lifetime and why?

Until the program terminates. Since it is a global variable, it is available at all times, to all scopes below it until the program terminates.

cc2
Which area: [1] [2] [3] [4a] [✓ **4b**] [5]
Why is that area?

Although declared within a scope of a function, its declaration includes the "static" keyword. This means that the variable is placed in the data area.

What is its lifetime and why?

Until the program terminates. Since it is declared as a static variable, it is available within the scope when the function is invoked and retains its value after the function's activation record is removed from the stack.

cc3, the object
Which area: [1] [✓ **2**] [3] [4a] [4b] [5]
Why is that area?

The variable recieves an instance ofa credit_card object created by the update_credit_card function. The instance is stored on the runtime stack as it is a local variable of the main() function

What is its lifetime and why?

Until the execution of the function is complete. When a non-static, local variable is declared within a scope of a function, it is stored alongside the activation record of the function, which in turn exists on the runtime stack. Once the function reaches the end, the activation record is popped off the stack and all data within it are destroyed. In this case, the function is main(), therefore it will last until the execution of the program itself is complete.

cc4, the object
Which area: [1] [✓ **2**] [3] [4a] [4b] [5]
Why is that area?

It is a local variable defined within the scope of the main() function.

What is its lifetime and why?

Until the execution of the function is complete. When a non-static, local variable is declared within a scope of a function, it is stored alongside the activation record of the function, which in turn exists on the runtime stack. Once the function reaches the end, the activation record is popped off the stack and all data within it are destroyed. In this case, the function is main(), therefore it will last until the execution of the program itself is complete.

c. What is the **output** of the program?

Step B	
Original score:	*15/20*
Reasoning and corrections:	*No deductions for parts A and B.* *Reference: Packages 03 and 05.* • *The packages demonstrates the lifetimes of different variables in various scopes.* • *I am actually not too sure about* cc3*, since it makes use of a function that has a return type of a reference type but then returns a deferenced pointer (which would make it return a instance of the* credit_card *type).* *However, as the variable declaration of* cc3 *itself is in the main() function, I assume that the object exists on the stack alongside the function's activation record as it would normally be.* • *Not quiet sure on the distinction between areas 4A and 4B either. I assume the only difference is if a variable has been allocated a value or not.* *Deducted 2 points for part C.* *Reference: Tested the provided code snippet* • *Did not answer the question. Skipped to attempt other questions.* • *The output would be as follows:* *Credit [Tesla]: 300* *Credit [Zoom]: 300* *Credit [Tesla]: 500* *Credit [Zoom]: 300*
Room for improvement:	• *Better time management and strategic planning.* • *I found this question to be harder than ones below that offer more points. This question should have been deprioritized in order to answer more higher-valued questions.*

A.4 - 0 pts – *Not covered in class yet. Please have this question as a future practice problem.*

Please **explain each** parameter passing method, **when to use** it, and **code a function** prototype example.

Pass-by-lvalue-reference

Pass-by-const-lvalue-reference

Step B	
Original score:	*0/0*
Reasoning and corrections:	*Did not attempt as it did not offer credit.*

	• *I believe this was partially discussed in class, but I have no idea what the question is referring to.*
Room for improvement:	• *N/A*

A.5 - 0 pts - *Not covered in class yet. Please have this question as a future practice problem.*

```
int cs = 340;
int *pointer = &cs;
```

Please code a constant pointer which points to **pointer**

Please code an **lvalue** reference to reference ***pointer**

Please code an **rvalue** reference to reference ***pointer**

Please use what you created, if legal, to change the value of **cs** to 413. **Provide your code** or **reasoning**:

Step B	
Original score:	*0/0*
Reasoning and corrections:	*Did not attempt as it did not offer credit.* • *I believe this was partially discussed in class, but I have no idea what the question is referring to ~~except maybe the first part~~.* • *I thought a constant pointer pointing to a pointer called "pointer" could be declared as: "const int** pp = pointer" but it didn't work as expected…*
Room for improvement:	• *N/A*

A.6 - 30 Points

```
…
static int x = 1;
int y = x * 2;

void t1() {
      y++;                                          // (y is 8 before t1 executes) therefore, y: 9
      cout << "x: " << x << " | y: " << y << endl;  // "x: 1 | y: 9"
      y += 1;                                       // y: 10
      x -= -1;                                      // static x: 2
}

void t2() {
      int* x = &y;                                  // local x: ptr to y
      cout << "x: " << x << " | y: " << y << endl;  // "x: [memaddr] | y: 10"
}

void t3() {                                             // first invocation          // second invocation
      int y = x;                                        // local y: global x = 2     // local y: global x = 2
      static int x = 2;                                 // static t3 x: 2            // (x now static local scope: 4)
      cout << "x: " << x + 1 << " | y: " << y + x << endl; // "x: 3 | y: 4"          // "x: 5 | y: 6"
      x += y;                                           // static t3 x: 4            // static t3 x: 6
}

void t4() {
      int y = x + 1;                                  // local y: global x + 1 = 3
      int& z = y;                                     // z -> local y: 3
      z += -1;                                        // z = y = 2
      cout << "x: " << x + z << " | y: " << y << endl;   // "x: 4 | y: 2"
}

int main() {
      vector<int> vec1{ 1, 3, 5, 7, 9 };
      vector<int> vec2{ 2, 4, 6, 8, 10 };
      vec1.swap(vec2);                                // vec1 is now even ints, vec2 is odd ints
      int * ptr = &vec1[1];                           // ptr to 4
      y = *(ptr + 2);                                 // shift ptr two slots and derefs; y is now 8

      t1();
      t2();
      t3();
      t3();
      t4();
      return 0;
}
```

This program **outputs** 5 lines. What are they?

1.

x: 1 | y: 9

2.

x: [memaddr] | y: 9

3.

x: 3 | y: 4

4.

x: 5 | y: 6

5.

x: 4 | y: 2

Step B	
Original score:	*30/30*
Reasoning and corrections:	*No deductions.* *Reference: Tested the provided code snippet.* • *To output matched the expected answer, with the excpetion where (2) prints the memory address of the variable.*
Room for improvement:	• *Faster analysis of the code. Did not have time to attempt the next high-value question.*

A.7 - 30 Points

```
int x = 1, y = -1;

void swapplus1(int n1, int n2) {                // note y=1 before any fn execution, therefore n1=1, n2=1
        int temp = n1 + 1;                      // blah blah, this is PBV; swap doesn't work.
        n1 = n2 - 1;                            // global x: 1, global y: 1
        n2 = temp;
        x = x + n1;
}
void swapplus2(int& n1, int& n2) {                // still n1=1, n2=1
       int temp = n1 + 1;                         // temp: 2
       n1 = n2 - 1;                               // n1: 0
       n2 = temp;                                 // n2: 2
}                                                 // (PBR) => global x: 0, global y: 2
void swapplus3(const int& n1, const int& n2) {    // n1=0, n2=2
       int n1val, n2val, temp = n1 + 1;           // temp: 1
       n1val = n2 - 1;                            // n1val: 1
       n2val = temp;                              // n2val: 1
       y -= n2;                                   // (direct access) y: 0
}                                                 // => x: 0, y: 0
void swapplus4(int* p1, int* p2) {                // p1 -> 0, p2 -> 0
       int temp = *p1 + 1;                        // temp: 1
       *p1 = *p2 + 1;                             // *p1: 1
       *p2 = temp;                                // *p2: 1
       x = *p1 + y;                               // (direct access) x: 2
}                                                 // => x: 2, y: 1
void swapplus5(int* &p1, int* &p2) {              // p1 = px -> 1, p2 = py -> 8
       int* temp = p1 + 1;                        // (move p1 slot by 1, presumably where p2 is) temp: (px+1) -> py
       p1 = p2 - 1;                               // (move p1 slot to the slot before p2, presumably where p1 already is) p1: (py-1) -> px
       p2 = temp;                                 // p2: py
}                                                 // => (pointers still points to the same places)
void print(const int& x, const int& y) {
       cout << "\n x: " << x << " |y: " << y;
}

int main() {
       int arr[]{ 2, 4, 6, 8, 10, 12, 14 };
       y = arr[3] / size(arr) ;                   // 8/7=1

       swapplus1(x, y);     print(x, y);
       swapplus2(x, y);     print(x, y);
       swapplus3(x, y);     print(x, y);
       swapplus4(&x, &y);  print(x, y);
       int *px = &x, *py = &y;                    // px -> 2, py -> 1
       (*px)--;                                   // (deref and decr) x: 1
       (*py) -= -7;                               // (deref and +7) y: 8
       swapplus5(px, py);   print(x, y);
       return 0;
}
```

This program **outputs** 5 lines. What are they?

1.

2.

3.

4.

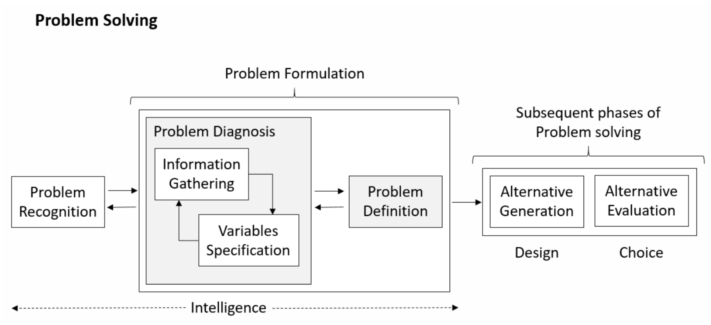
5.

Step B	
Original score:	*0/30*
Reasoning and corrections:	*Failed to attempt the question. Ran out of time.* *Reference: Tested the provided code snippet.* • *See the comments beside the code for the tracing attempt.*

	• *The program outputs the following:* *x: 1 |y: 1* *x: 0 |y: 2* *x: 0 |y: 0* *x: 2 |y: 1* *x: 1 |y: 8*
Room for improvement:	• *Should have planned more strategically in order to attempt this question, rather than doing the lesser-valued questions.*

PART B – 0 Points

B.1 - 0 pts – *Not covered in class yet. Please have this question as a future practice problem.*



What did the IDEO team do during the Problem Formulation phase? Did you use the problem-solving steps in ASMT 2 and ASMT 3? How or why not?

Step B	
Original score:	*0/0*
Reasoning and corrections:	*Did not attempt as it did not offer credit.*
Room for improvement:	• *N/A*

B.2 - 0 pts – *Not covered in class yet. Please have this question as a future practice problem.*

IDEO: In the future, if you will be a CEO of a software company, will you fire people who are better than you are? How will you feel sitting in a meeting with these people? What are the risks if any?

Step B	
Original score:	*0/0*
Reasoning and corrections:	*Did not attempt as it did not offer credit.*
Room for improvement:	• *N/A*