

## MIDTERM EXAM INSTRUCTIONS

1. Midterm Exam: **25 points w/ 0 E.C. points**
2. Due Date & Time: **10-18-2020 at 11:55 PM**

## 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	
<b>TOTAL</b>	<b>245</b>	

**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.

## 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 original answers, type in *italic* text and:
  - Give your original 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 original 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.
3. Think if the same topics will be tested again in our final exam, what questions we may get.

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

1. Section, Date and Time:

TIC, TAC, and TOE. Due 10-18-2020 at 11:55 PM

Full Name in CAPITAL LETTERS | SFSU ID

KAYVAUN KHOSHKHOU

920357344

2. Midterm Exam (2 exams, 0 dropped): 100 points

3. To prepare for this exam, please review all the related materials including WEEK 01-08 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 paper. No handwriting. No scanning. 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, and no communication with anyone except the course instructor. The course instructor will be available on Zoom or email during the exam time: zoom.ducta.net

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](http://csc340.ducta.net/00-README-StudentConduct_AcademicHonesty.pdf)

## EXAM START: SATURDAY 10/17/2020 4:00 PM

### PART A – 90 Points

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

Reference variables can not be reinitialized where as pointers can be reinitialized a number of times. Reference variables are used through '&' syntax and pointers are used through '\*'. A reference variable returns the address of a variable where as the pointer variable returns the value of that variable. One last note is that the reference variable is another name for the already existing variable and the pointer variable holds the address for a variable.

#### Pointer Example:

```
string* ptr = &color
```

Here's a pointer variable with the name ptr that stores the ADDRESS of color.

#### Reference Variable Example:

```
Int& x = y;
```

Here's a reference variable where you can see the & being used to distinguish the syntax difference between a standard variable and a reference variable.

.....

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

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

Here we have a reference being reassigned and as we've learned in class we know that one of the biggest rules to remember when working with references is that they cannot be reassigned because references share the address of the variable assigned.

.....

**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 memory or storage that is no longer allocated or already discharged. A stale pointer is essentially when a pointer clears a memory allocation, then another pointer references that same memory allocation that was cleared. The pointer that pointed to the cleared memory allocation is the stale 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.

What `delete` does is it deallocates memory from the heap so the pointer that points to the object is not deleted. However, the value or memory pointed to by the pointer is deleted and allocates the variables and memory to heap memory.

## 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?

**type** is a global variable initialized in the global scope. It is not stored in the runtime stack memory area or the free-store, and it cannot be in 4a because it is initialized already.

*What is its lifetime and why?*

Its lifetime is active from its declaration to the programs shutdown. It is declared in the global scope of the code so from the moment it is declared its life begins and continues to be active until the program shuts down.

### cc2

Which area: [1] [2] [3] [4a] [4b] [5]

Why is that area?

**cc2** is declared after static credit card\*. Hence it is a static uninitialized form of data belonging in 4a memory area.

*What is its lifetime and why?*

Its life begins there and for a very brief period it is used to be pointed to the value of cc1, and is then returned.

### cc3, the object

Which area: [1] [2] [3] [4a] [4b] [5]

Why is that area?

**cc3** is declared inside main, as an object. Here it is initialized and ran only through the runtime stack since it is not a global variable. It is not in the free-store memory area because in order to allocate memory to those areas you need to use "new" or "delete".

*What is its lifetime and why?*

Its lifetime begins in main and ends upon the completion of main when the return value is given. Cc3 is not used anywhere outside of main.

### cc4, the object

Which area: [1] [2] [3] [4a] [4b] [5]

Why is that area?

Similarly to cc3, cc4 is declared inside of main and nowhere else outside of it.

*What is its lifetime and why?*

Its lifetime is the same as cc3, however to be technical the usage of cc4 goes a little farther than cc3 because its info is displayed to the user using display\_info(). It's brought up a few times more than cc3 as well, to provide accurate updates since it is displayed twice.

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

```
Credit [Tesla]: 300
Credit [Zoom]: 300
Credit [Tesla]: 500
Credit [Zoom]: 300
```

**A.4 - 10 pts** – Please **explain each** parameter passing method, **when to use** it, and **code a function** prototype example.

Pass-by-lvalue-reference

In pass-by-lvalue-reference instead of variable names being passed we want to make sure the changes in the function affect the variable, so we pass pointers. You would use this reference if you're intention was to cut data out of a parameter rather than copying it. A good analogy to compare this to would be the "copy-paste" feature computers have. Instead of "copy-pasting" you would be "cut-pasting".

Pass-by-const-rvalue-reference

The changes in the function will not actually affect the variable here.

**A.5 - 10 pts**

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

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

```
int* const constPointer = &cs;
```

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

```
int& lvalueConst = *pointer;
```

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

```
int& *rvalueConst = *pointer;
```

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

```
*constPointer = 413;
```

A.6 - 15 Points

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

void t1() {
    y++;
    cout << "x: " << x << " | y: " << y << endl;
    y += 1;
    x -= -1;
}

void t2() {
    int* x = &y;
    cout << "x: " << x << " | y: " << y << endl;
}

void t3() {
    int y = x;
    static int x = 2;
    cout << "x: " << x + 1 << " | y: " << y + x << endl;
    x += y;
}

void t4() {
    int y = x + 1;
    int& z = y;
    z += -1;
    cout << "x: " << x + z << " | y: " << y << endl;
}

int main() {
    vector<int> vec1{ 1, 3, 5, 7, 9 };
    vector<int> vec2{ 2, 4, 6, 8, 10 };
    vec1.swap(vec2);
    int * ptr = &vec1[1];
    y = *(ptr + 2);

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

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

1. x: 1 | y: 9
2. x: ? | y: 10
3. x: 3 | y: 4
4. x: 5 | y: 6
5. x: 4 | y: 2

## A.7 - 15 Points

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

```
void swapplus1(int n1, int n2) {
    int temp = n1 + 1;
    n1 = n2 - 1;
    n2 = temp;
    x = x + n1;
}
void swapplus2(int& n1, int& n2) {
    int temp = n1 + 1;
    n1 = n2 - 1;
    n2 = temp;
}
void swapplus3(const int& n1, const int& n2) {
    int n1val, n2val, temp = n1 + 1;
    n1val = n2 - 1;
    n2val = temp;
    y -= n2;
}
void swapplus4(int* p1, int* p2) {
    int temp = *p1 + 1;
    *p1 = *p2 + 1;
    *p2 = temp;
    x = *p1 + y;
}
void swapplus5(int* &p1, int* &p2) {
    int* temp = p1 + 1;
    p1 = p2 - 1;
    p2 = temp;
}
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);

    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)--;
    (*py) -= -7;
    swapplus5(px, py);  print(x, y);
    return 0;
}
```

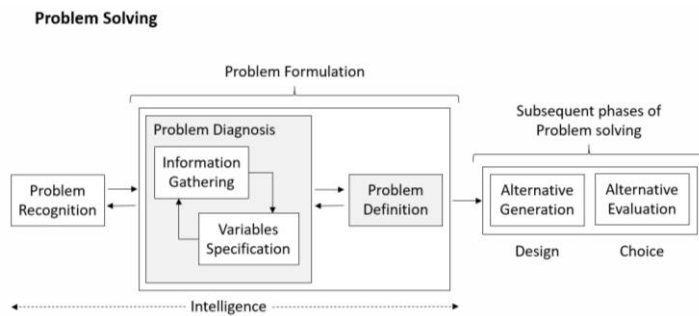
This program **outputs** 5 lines. What are they?

1. x: 1 |y: 1
2. x: 0 |y: 2
3. x: 0 |y: 0
4. x: 2 |y: 1
5. x: 1 |y: 8



## PART B – 10 Points

B.1 - 5 pts



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?

The IDEO team gathered information and specified variables during the problem formulation phase. They then defined their problem before going onward to the next phase, but they first had to gather info, and specify variables before defining the problem at hand in order to continue. I used the problem-solving steps in my previous assignments however I came across many hiccups when defining my problems at hand. I needed a better way of tackling the problems I defined with perhaps more specifications or information even though I had prepared a lot. It still wasn't enough and left me ill-prepared for my task at those times. The solution to this is to dedicate more time to gathering information and variable specification.

B.2 - 5 pts

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?

If there's one thing my father taught that has rung true throughout my entire life it's that no matter what craft I choose to master there will always be someone who can do it better than I can. I took this with a grain of salt and twisted it to better inspire and motivate myself to shrink the size of the pool of those who can claim to be better than me at my mastered craft by truly working hard and putting my best effort into it. If I pull this off, master my craft, become the CEO and find someone who is better than me, I would not fire them. As a CEO, I would assume I have some management skills and would have to maintain the leadership role I assumed when I agreed to that position. A king is not necessarily the strongest duelist or combatant but he surrounds himself with the round table of knights who excel at what they do. The risk of being dethroned exists but in order to keep your position you must maintain the consensus that you are the greatest leader amongst them.

**EXAM END: SATURDAY 10/17/2020 4:50 PM**

# EXAM CORRECTION

## PART A – 90 Points

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

Reference variables can not be reinitialized where as pointers can be reinitialized a number of times. Reference variables are used through '&' syntax and pointers are used through '\*'. A reference variable returns the address of a variable where as the pointer variable returns the value of that variable. One last note is that the reference variable is another name for the already existing variable and the pointer variable holds the address for a variable.

**Pointer Example:**

```
string* ptr = &color
```

Here's a pointer variable with the name ptr that stores the ADDRESS of color.

**Reference Variable Example:**

```
Int& x = y;
```

Here's a reference variable where you can see the & being used to distinguish the syntax difference between a standard variable and a reference variable.

Answer is fine here, upon looking at examples online and in the packages, some more detail could have been used but overall the message was passed through and should be accepted.

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

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

Here we have a reference being reassigned and as we've learned in class we know that one of the biggest rules to remember when working with references is that they cannot be reassigned because references share the address of the variable assigned.

This statement would cause an error. The answer given that the references cannot be reassigned is correct.

**10/10**

**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 memory or storage that is no longer allocated or already discharged. A stale pointer is essentially when a pointer clears a memory allocation, then another pointer references that same memory allocation that was cleared. The pointer that pointed to the cleared memory allocation is the stale pointer.

All resources show that the answer given by student is correct. A good example of this is in package 3, on line 285 we see an example of a stale pointer. An example should have been given.

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.

What `delete` does is it deallocates memory from the heap so the pointer that points to the object is not deleted. However, the value or memory pointed to by the pointer is deleted and allocates the variables and memory to heap memory.

Similarly here, from package 3 you can see on line 289 that we are using `delete` to deallocate memory in `pStr2`. The given answer here is correct, but a code example was not shown. Only half credit will be given. (Time restraint on this exam was a bit rough with 50 minutes, I was not creative enough to find time to create examples for a couple of problems.)

5/10

### 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:

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

e. 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?

**type** is a global variable initialized in the global scope. It is not stored in the runtime stack memory area or the free-store, and it cannot be in 4a because it is initialized already.

*What is its lifetime and why?*

Its lifetime is active from its declaration to the programs shutdown. It is declared in the global scope of the code so from the moment it is declared its life begins and continues to be active until the program shuts down.

Student is correct, type is a global variable initialized in the global scope. Its lifetime is also correct as it is active from declaration all the way to shutdown. Package 07 code on line 42 can be referenced for confirmation.

#### cc2

Which area: [1] [2] [3] [4a] [4b] [5]

Why is that area?

**cc2** is declared after static credit card\*. Hence it is a static uninitialized form of data belonging in 4a memory area.

*What is its lifetime and why?*

Its life begins there and for a very brief period it is used to be pointed to the value of cc1, and is then returned.

Student's answer is correct, though it could use a little more explanation. Lifetime description is also correct. Answer located in Package 07, a good example of a reference for this would be starting on line 73.

#### cc3, the object

Which area: [1] [2] [3] [4a] [4b] [5]

Why is that area?

**cc3** is declared inside main, as an object. Here it is initialized and ran only through the runtime stack since it is not a global variable. It is not in the free-store memory area because in order to allocate memory to those areas you need to use "new" or "delete".

*What is its lifetime and why?*

Its lifetime begins in main and ends upon the completion of main when the return value is given. Cc3 is not used anywhere outside of main.

Good overall answer, reference to package 07 line 42 for a box with memory area allocation and in particular studentPtr2 as an object.

cc4, the object

Which area: [1] [2] [3] [4a] [4b] [5]

Why is that area?

Similarly to cc3, cc4 is declared inside of main and nowhere else outside of it.

*What is its lifetime and why?*

Its lifetime is the same as cc3, however to be technical the usage of cc4 goes a little farther than cc3 because its info is displayed to the user using display\_info(). It's brought up a few times more than cc3 as well, to provide accurate updates since it is displayed twice.

Good overall answer, can also reference to package 07 line 42 for a box with memory area allocation and in particular studentPtr2 as an object. Cc4 and cc3 are very similar in terms of this program, they are in the same memory allocation and used in a similar context.

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

```
Credit [Tesla]: 300
Credit [Zoom]: 300
Credit [Tesla]: 500
Credit [Zoom]: 300
```

After running this through visual studio, the same output was received

```
Credit [Tesla]: 300
Credit [Zoom]: 300
Credit [Tesla]: 500
Credit [Zoom]: 300
```

**20/20**

**A.4 - 10 pts** – Please **explain each** parameter passing method, **when to use it**, and **code a function** prototype example.

Pass-by-lvalue-reference

In pass-by-lvalue-reference instead of variable names being passed we want to make sure the changes in the function affect the variable, so we pass pointers. You would use this reference if you're intention was to cut data out of a parameter rather than copying it. A good analogy to compare this to would be the "copy-paste" feature computers have. Instead of "copy-pasting" you would be "cut-pasting".

Pass-by-const-rvalue-reference

The changes in the function will not actually affect the variable here.

Student gives a good explanation for Pass-by-lvalue-reference but not as good for Pass-by-const-rvalue-reference. The answers to this problem could have been found in Package 05 on lines 301 and 302.

**5/10**

A.5 - 10 pts

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

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

```
int* const constPointer = &cs;
```

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

```
int& lvalueConst = *pointer;
```

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

```
int& *rvalueConst = *pointer;
```

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

```
*constPointer = 413;
```

Student's answer for rvalue is incorrect. After running the program in visual studio, I found that my mistake was in trying to use a pointer to a reference. I'm not sure why I didn't pick this up during the exam but after I reviewed it now I'm surprised it went over my head as I had hammered it in pretty well to remember that you cannot use a pointer to a reference. After some adjustment, the rest of the code was okay.

**7.5/10**

A.6 - 15 Points

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

void t1() {
    y++;
    cout << "x: " << x << " | y: " << y << endl;
    y += 1;
    x -= -1;
}

void t2() {
    int* x = &y;
    cout << "x: " << x << " | y: " << y << endl;
}

void t3() {
    int y = x;
    static int x = 2;
    cout << "x: " << x + 1 << " | y: " << y + x << endl;
    x += y;
}

void t4() {
    int y = x + 1;
    int& z = y;
```

```

    z += -1;
    cout << "x: " << x + z << " | y: " << y << endl;
}

int main() {
    vector<int> vec1{ 1, 3, 5, 7, 9 };
    vector<int> vec2{ 2, 4, 6, 8, 10 };
    vec1.swap(vec2);
    int * ptr = &vec1[1];
    y = *(ptr + 2);

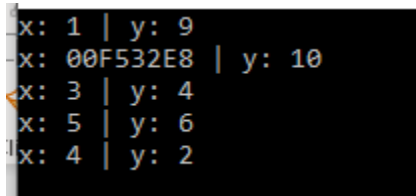
    t1();
    t2();
    t3();
    t3();
    t4();
    return 0;
}

```

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

1. x: 1 | y: 9
2. x: ? | y: 10
3. x: 3 | y: 4
4. x: 5 | y: 6
5. x: 4 | y: 2

After inputting the program into visual studio the output given my the student matched that of the computer except for line 2 x:. Student left this blank. (I was unsure what to get here, after correcting it through the computer I see now that it is a memory address.)



```

x: 1 | y: 9
x: 00F532E8 | y: 10
x: 3 | y: 4
x: 5 | y: 6
x: 4 | y: 2

```

9/10

#### A.7 - 15 Points

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

```

void swapplus1(int n1, int n2) {
    int temp = n1 + 1;
    n1 = n2 - 1;
    n2 = temp;
    x = x + n1;
}

void swapplus2(int& n1, int& n2) {
    int temp = n1 + 1;
    n1 = n2 - 1;
    n2 = temp;
}

void swapplus3(const int& n1, const int& n2) {
    int n1val, n2val, temp = n1 + 1;
    n1val = n2 - 1;
    n2val = temp;
}

```

```
    y -= n2;
}
void swapplus4(int* p1, int* p2) {
    int temp = *p1 + 1;
    *p1 = *p2 + 1;
    *p2 = temp;
    x = *p1 + y;
}
void swapplus5(int* &p1, int* &p2) {
    int* temp = p1 + 1;
    p1 = p2 - 1;
    p2 = temp;
}
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);

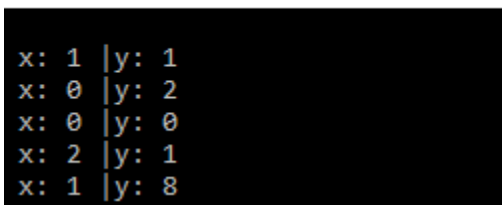
    swapplus1(x, y);    print(x, y);
    swapplus2(x, y);    print(x, y);
    swapplus3(x, y);    print(x, y);
    swapplus4(&x, &y);  print(x, y);
    int *px = &x, *py = &y;
    (*px)--;
    (*py) -= -7;
    swapplus5(px, py);  print(x, y);
    return 0;
}
```

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

1. x: 1 |y: 1
2. x: 0 |y: 2
3. x: 0 |y: 0
4. x: 2 |y: 1
5. x: 1 |y: 8

After inputting the program into visual studio the output given by the student matched that of the computer.

 Microsoft Visual Studio Debug Console



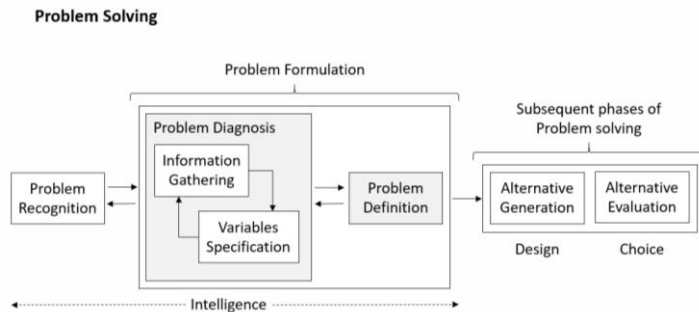
```
x: 1 |y: 1
x: 0 |y: 2
x: 0 |y: 0
x: 2 |y: 1
x: 1 |y: 8
```

10/10



## PART B – 10 Points

## B.1 - 5 pts



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?

The IDEO team gathered information and specified variables during the problem formulation phase. They then defined their problem before going onward to the next phase, but they first had to gather info, and specify variables before defining the problem at hand in order to continue. I used the problem-solving steps in my previous assignments however I came across many hiccups when defining my problems at hand. I needed a better way of tackling the problems I defined with perhaps more specifications or information even though I had prepared a lot. It still wasn't enough and left me ill-prepared for my task at those times. The solution to this is to dedicate more time to gathering information and variable specification.

Good answer, Package 03 first page was extremely useful in verifying the answer to the student's response here.

## B.2 - 5 pts

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?

If there's one thing my father taught that has rung true throughout my entire life it's that no matter what craft I choose to master there will always be someone who can do it better than I can. I took this with a grain of salt and twisted it to better inspire and motivate myself to shrink the size of the pool of those who can claim to be better than me at my mastered craft by truly working hard and putting my best effort into it. If I pull this off, master my craft, become the CEO and find someone who is better than me, I would not fire them. As a CEO, I would assume I have some management skills and would have to maintain the leadership role I assumed when I agreed to that position. A king is not necessarily the strongest duelist or combatant but he surrounds himself with the round table of knights who excel at what they do. The risk of being dethroned exists but in order to keep your position you must maintain the consensus that you are the greatest leader amongst them.

Great Answer! The king should surround himself with those who are confident in their abilities and excel at what they do in order to make the king successful. (I was very proud of this analogy when I came up with it on the spot, I think it works really well.)

**10/10**

.....  
TOTALS

Part A: 66.5 / 90

Part B: 10 / 10

**GRAND TOTAL: 76.5 / 100 = C**