CSE 333 26su

Midterm Exam

Do not turn the page until you are told to do so.

First Name	**Sample Solution**
Last Name	
UW NetID	

Instructions

- This exam contains 9 pages, including this cover page.
- During the exam we will hand out another 3 pages for the personalized portion of the exam.
- The exam is closed book .
- You are allowed one page of handwritten notes. Double -sided is ok.
- Silence and put away all cell phones and other mobile or noise -making devices.
- Remove all hats, headphones, smart glasses, watches, and other digital wearables.
- You will have 55 minutes to complete this exam.

Advice

- Read all questions first and start where you feel the most confident.
- Read questions carefully before starting.
- Skip questions that are taking a long time.
- Show scratch work for partial credit, but put your final answers in the boxes and blanks provided.
- Relax. If you've been practicing, you got this! If you haven't, you'll learn something new!

Question	1	2	3	4	5	6*	Total
Possible Points	16	12	10	10	12	8	68

* Question 6 will be personalized to each student. We will hand this out once this exam starts.

Question 1. Code Review (16 pts)

The program below compiles and runs correctly. Read it carefully, then answer the four parts on the next page. Refer to lines by the numbers in the left gutter.

```
1  using namespace std;
2
3  #include <iostream>
4  #include <cstdlib>
5  #include <string>
6
7
8  struct PokeMove {
9      string name;
10     int power;
11 };
12
13 class Pokemon {
14  public:
15     Pokemon(string n, int numMoves) {
16         name = n;
17         moves = new PokeMove[numMoves];
18     }
19
20     string GetName() const { return name; }
21
22     PokeMove GetMove(int i) const { return moves[i]; }
23
24     void SetMove(int i, PokeMove move) {
25         moves[i] = move;
26     }
27
28  private:
29     string name;
30     PokeMove* moves;
31 };
32
33 // Helper functions to print the Pokedex banner.
34 static void printBanner() {
35     cout << "=== Pokedex ===" << endl;
36 }
37 static void printDivider() {
38     cout << "----------------" << endl;
39 }
40
41 int main() {
42     printBanner();
43     printDivider();
44
45     Pokemon *pika = new Pokemon("Pikachu", 4);
46     PokeMove tbolt = {"Thunderbolt", 90};
47
48     pika->SetMove(0, tbolt);
49     cout << pika->GetName() << " is ready!\n";
50
51     return 0;
52 }
```

Part A. Find three distinct CSE 333 style-guide violation. Circle the offending line number(s) on the previous page and write the line number(s) below. Then, explain why each one is wrong. [6 pts]

Line 1: #includes should be at the top of the file. We would add using namespace afterward.

Line 3-5: #includes should be ordered alphabetically.

Line 4: #include <cstdlib> is never used. Only include headers you use.

Line 15: parameter numMoves should be snake_case, so num_moves.

Line 15: string n is passed by value. Take it as const string& to avoid a copy.

Lines 16-17: set the members in an initializer list, not by assignment in the body.

Lines 29-30: member variables need a trailing underscore, so name_ and moves_.

Lines 34, 37: function names must be CamelCase, so PrintBanner and PrintDivider.

Lines 36-37: functions should have a newline between them for proper spacing.

Line 45: put the star on the type, so Pokemon* pika, not Pokemon *pika.

Line 49: printing in C++ should use std::endl instead of the '\n' character

Line 51: return EXIT_SUCCESS

Part B. The `SetMove` method takes its `PokeMove` parameter by value. Rewrite the `SetMove` method signature below using a const reference. Explain why we use const. Explain why we use a reference. [4 pts]

void SetMove(int i, const PokeMove& move)

const: we promise not to change the caller's move.
reference: we skip copying the struct.

Part C. The program compiles and runs, but it leaks memory. What must be added to the `Pokemon` class to prevent the leak, and what should it do? [3 pts]

Add a destructor like: ~Pokemon() { delete[] moves; }.

It frees the array the constructor got with new[]. Memory leak without this.

Part D. Suppose you added the fix from part (c). The code below then compiles and runs.

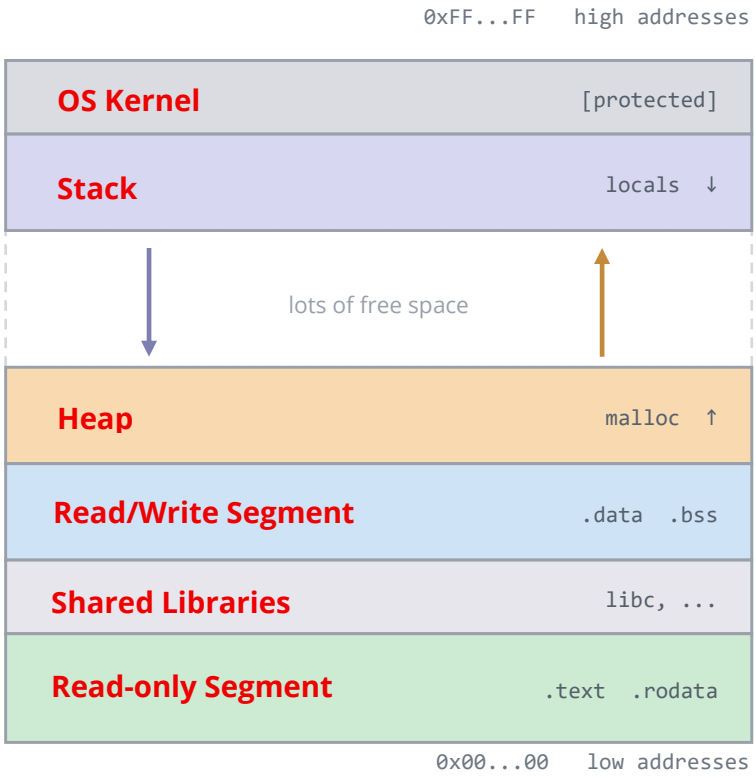
```
Pokemon a("Charmander", 4);
Pokemon b = a;   // b is a copy of a
```

Explain what goes wrong, and what else Pokemon needs (hint: Rule of Three). [3 pts]

Pokemon b = a; does a shallow copy, so a and b share one moves pointer.
The problem is that both destructors attempt to delete[] the pointer which is a double free.
The fix is to add a copy constructor and copy assignment that deep-copy the moves array.

Question 2. Virtual Address Space (12 pts)

The diagram shows a process's virtual address space from high addresses (top) to low addresses (bottom). The regions are drawn to scale, but the label for each region has been removed.



Part A. Fill in the name of each memory region on the diagram above. [6 pts]

Part B. List each region below and describe what type of data a program keeps there. [6 pts]

OS Kernel: kernel code for system calls
Stack: locals, parameters, return addresses, function stack frames
Heap: memory from new/malloc.
Read/Write Segment: global and static variables.
Shared Libraries: dynamically linked libraries
Read-Only Segment: code and string literals

Question 3. Constructors, Destructors, and Copy (10 pts)

Write the program's complete output, in order, exactly as it is printed.

```
1  class Box {
2   public:
3      Box() : data_(0) {
4          cout << "default 0" << endl;
5      }
6
7      Box(int v) : data_(v) {
8          cout << "ctor " << data_ << endl;
9      }
10
11     Box(const Box& other) : data_(other.data_ + 1) {   // don't forget the +1
12         cout << "copy ctor " << data_ << endl;
13     }
14
15     Box& operator=(const Box& other) {
16         data_ = other.data_ * 2;                    // don't forget the x2
17         cout << "assign " << data_ << endl;
18         return *this;
19     }
20
21     ~Box() { cout << "dtor " << data_ << endl; }
22
23  private:
24     int data_;
25 };
26
27 void CopyByValue(Box x) {
28     cout << "helper function" << endl;
29 }
30
31 int main() {
32     Box a(4);
33     Box b = a;
34     CopyByValue(b);
35     Box c;
36     c = b;
37     return 0;                                       // don't forget program ends
38 }
```

Program output:

```
ctor 4
copy ctor 5
copy ctor 6
helper function
dtor 6
default 0
assign 10
dtor 10
dtor 5
dtor 4
```

Question 4. Preprocessor (10 pts)

Read the program below carefully, then answer the four parts on the next page. You will be asked exactly what the C preprocessor emits for each of the files.

list.h

```
ENTRY(ErrorCode_Ok,            0,  "No error")
ENTRY(ErrorCode_DivideByZero,  1,  "Divide by zero")
ENTRY(ErrorCode_Overflow,      2,  "Arithmetic Overflow")
```

error_codes.h

```
#define ENTRY(name, num, msg)  const int name = num;
#include "list.h"
```

error_messages.h

```
const char* error_message[3];

void InitMessages() {
#define ENTRY(name, num, msg)  error_message[num] = msg;
#include "list.h"
}
```

main.c

```
#include "error_codes.h"
#include "error_messages.h"

int main() {
    InitMessages();
    int status = ErrorCode_DivideByZero;
    printf("%d:%s\n", status, error_message[status]);
    return 0;
}
```

Above, our program defines a list of error codes in **list.h**. We reuse that list in two ways:

error_codes.h uses **list.h** to build a set of global integer constants

error_messages.h uses **list.h** to build a lookup that turns an error code into a printable message.

At first, this may seem daunting! But remember, the preprocessor applies text substitution blindly. It does not need to understand the program to replace macros. Similarly, we do not need to understand the program to perform the same text substitutions.

Part A. Write out `error_codes.h` exactly as the compiler sees it after the preprocessor runs. [4 pts]

```
const int ErrorCode_Ok = 0;
const int ErrorCode_DivideByZero = 1;
const int ErrorCode_Overflow = 2;
```

Part B. Write out `error_messages.h` exactly as the compiler sees it after the preprocessor runs. [4 pts]

```
const char* error_message[3];

void InitMessages() {
error_message[0] = "No error";
error_message[1] = "Divide by zero";
error_message[2] = "Arithmetic Overflow";
}
```

Part C. What does `main.c` program print when it runs? [2 pts]

Program output:

1:Divide by zero

Question 5. Inheritance Design (12 pts)

The Notification class below handles three kinds of notification: email, sms, and a default fallback. It compiles and works, but it would greatly benefit from polymorphism. Read the code

notification.cc

```
class Notification {
 public:
  Notification(string type, string to, string body)
      : type_(type),
        to_(to),
        body_(body) {}

  void Send() {
      if (type_ == "email") {
          cout << "Email to " << to_ << " | " << subject_ << " | " << body_;
      } else if (type_ == "sms") {
          cout << "SMS to " << to_ << ": " << body_;
      } else {
          cout << "Notify " << to_ << ": " << body_;
      }
  }

  int Cost() {
      if (type_ == "sms") {
          return 5;
      } else {
          return 0;
      }
  }

  void SetEmailSubject(string s) {
      if (type_ == "email") {
        subject_ = s;
      }
  }

 private:
    string type_;
    string to_;
    string body_;
    string subject_;
};


```

Part A. Pick two members in `notification.cc` (either a data member or a member function). For both, explain what's wrong with the member and how using inheritance / polymorphism would fix it. [6 pts]

subject_: only email uses it, yet every object stores it. Move it into EmailNotification.

Send(): the if/else-if/else on type_ is messy and trying to do dynamic dispatch manually. Make it virtual and both SmsNotification and EmailNotification can override it.

Cost(): the if/else-if/else on type_ is messy and trying to do dynamic dispatch manually. Make it virtual and only SmsNotification needs to override it.

SetEmailSubject(): Only used for emails. Move it out of the base class and place it directly in EmailNotification. Alternatively, add a constructor for EmailNotification that takes a subject

---

Part B. Redesign this as a base Notification with derived classes EmailNotification and SmsNotification.

For each class, sketch its header: the data members and the member functions it should declare. Skip any members that would be inherited from a base class. Write only method signatures (including virtual or override where appropriate). Think about which data and behavior belong in the base class versus a specific subclass. **You do NOT need to write any code.** [6 pts]

`class Notification {`

Data members	Member functions

to_, (protected)
body_ (protected)
	Notification(string to, string body);
	virtual void Send();
	virtual int Cost();

`class EmailNotification : public Notification {`

Data members	Member functions

subject_	EmailNotification(string to, string body);
	void Send() override;
	
	// must have at least one of the following two:
	void SetSubject(string s);
	EmailNotification(string to, string body, string subject);

`class SmsNotification : public Notification {`

Data members	Member functions

none (inherited)	SmsNotification(string to, string body);
	void Send() override;
	int Cost() override;