

## **ECS-36B, Spring 2023, Final Exam Solution,**

version: 0x17195891A50FE437816962ED

Name: \_\_\_\_\_

Student ID: \_\_\_\_\_

E-mail: \_\_\_\_\_

### **Important Instructions:**

- Open book/notes, Internet, laptop, chat-GPT
- We have totally 10 questions (4% each, total 40%).
- 14 pages [version: 0x17195891A50FE437816962ED]
- Please write precise and clean answers. If you write down the steps regarding how you obtain the final answer, I might give you partial credits even the final answer itself is wrong.
- Please write clearly to let the instructor recognize/understand your hand-writing.
- Every page of this exam book needs to be returned.
- The first six questions, on top of the given credits, will be used to determine the possible compensation credits for your six homework assignments (as we announced earlier)
- If we suspect any cheating behavior, we will pass the case to the academic committee immediately.

Q-1 4% (for HW-1) Comparing two different files (are they the same? Like the “diff” tool) with the following piece of C code –

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

int
main
(int argc, char **argv)
{
    FILE *f1 = NULL;
    FILE *f2 = NULL;

    if (sizeof(unsigned long) != 8)
    {
        printf("this program assumes unsigned long is 8 bytes\n");
        return -1;
    }

    if (argc != 3)
    {
        printf("argc = %d => usage: mydiff <first> <second>\n", argc);
        return -2;
    }

    f1 = fopen(argv[1], "r");
    f2 = fopen(argv[2], "r");

    if (f1 == NULL)
    {
        printf("file [%s] fopen failed\n", argv[1]);
        return -3;
    }

    if (f2 == NULL)
    {
        printf("file [%s] fopen failed\n", argv[2]);
        return -3;
    }

    while(1)
    {
        unsigned char byte_f1[8];
        unsigned char byte_f2[8];
        unsigned long *ptr_byte_f1 = (unsigned long *) (&byte_f1);
        unsigned long *ptr_byte_f2 = (unsigned long *) (&byte_f2);

        int cr_f1;
        int cr_f2;

        cr_f1 = fread(byte_f1, 1, 8, f1);
        cr_f2 = fread(byte_f2, 1, 8, f2);

        if ((cr_f1 == 0) && (cr_f2 == 0))
        {
            fclose(f1);
```

```

        fclose(f2);
        printf("done... [%s] and [%s] are the same\n",
               argv[1], argv[2]);
        return 0;
    }
    else
    {
        /* the following LINE determines the difference. */
        if ((cr_f1 == 0) || (cr_f2 == 0) || (cr_f1 != cr_f2) ||
            ((*ptr_byte_f1) != (*ptr_byte_f2)))
        {
            fclose(f1);
            fclose(f2);
            printf("done... [%s] and [%s] are different\n",
                   argv[1], argv[2]);
            return 0;
        }
    }
}
printf("this should never be reached!\n");
return -4;
}

```

The above program is a simple implementation of “diff” (we call it “mydiff”). For this question, we will focus on the high-lighted line of code, an “if condition” statement,

```

    if ((cr_f1 == 0) || (cr_f2 == 0) || (cr_f1 != cr_f2) ||
        ((*ptr_byte_f1) != (*ptr_byte_f2)))

```

which determines whether these two input files are different. Furthermore, this line consists of FOUR logical comparisons, and if any of them is true, the input file 1 and file 2 are considered different.

For this question, please focus on the last/4<sup>th</sup> logical comparison:

```

    ((*ptr_byte_f1) != (*ptr_byte_f2))

```

and, answer the following sub-questions [2-5] based on the above mydiff.c program.

- (1) What is `ptr_byte_f1`? [follow the example answer of (1) for the rest]

[Given answer] `ptr_byte_f1` is an “unsigned long”-casted address/pointer for the unsigned char array, `byte_f1[8]`.

- (2) 0.5% What is `*ptr_byte_f1`?

`*ptr_byte_f1` is an “unsigned long”, which has 8 bytes and the content is equivalent of those 8 bytes under the unsigned char array, `byte_f1[8]`.

- (3) 0.5% What is `ptr_byte_f2`?

`ptr_byte_f2` is an “unsigned long”-casted address/pointer for the unsigned char array, `byte_f2[8]`.

- (4) 0.5% What is `*ptr_byte_f2`?

`*ptr_byte_f2` is an “unsigned long”, which has 8 bytes and the content is equivalent of those 8 bytes under the unsigned char array, `byte_f2[8]`.

- (5) 2% What is the purpose of `((*ptr_byte_f1) != (*ptr_byte_f2))`?

Comparing whether the content of `byte_f1[8]`, 8 bytes total, is different from `byte_f2[8]` or not.

Q-2 4% (for HW-2) The following is an implementation of the libc string function `strdup`, and it contains ONE mistake which might cause segmentation fault at run-time --

```
char *
strdup(const char *s)
{
    size_t len = strlen(s);
    void *new = malloc(len);
    if (new == NULL) return NULL;
    return (char *) memcpy(new, s, len);
}
```

- (1) 2% What is the mistake in the above code? Please provide the correct code (only ONE line needed).

```
size_t len = strlen(s) + 1;
```

- (2) 2% Please explain briefly, **less than three sentences**, why might the above original code cause segmentation fault?

The original code might allocate one byte short, which should have been reserved for the null terminator of a C string. Furthermore, the `memcpy` call didn't copy the terminator, the null character. Therefore, the termination of the string is not guaranteed.

Q-3 4% (for HW-3) The following question is related to the source code under <https://github.com/sfelixwu/Person-and-Thing-HW3-Reference>

```
Json::Value
GPS_DD::dump2JSON
(void)
{
    Json::Value result;
    if ((this->latitude < -90.0) || (this->latitude > 90.0))
    {
        this->latitude = 0.0;
    }
    result["latitude"] = this->latitude;
    if ((this->longitude < -180.0) || (this->longitude > 180.0))
    {
        this->longitude = 0.0;
    }
    result["longitude"] = this->longitude;
    return result;
}

Json::Value
Person::dump2JSON
(void)
{
    Json::Value result;
    result["name"] = this->name;
    result["location"] = (this->location).dump2JSON();
    return result;
}

int
main(void)
{
    GPS_DD gps_IKEA_Sacramento { 38.58681641563053, -121.55296296578501};
    Person aPersonObject { "John", gps_IKEA_Sacramento };
    Json::Value myv = aPersonObject.dump2JSON();
    std::cout << myv.toStyledString() << std::endl;
}
```

4% Given the above code segments, please write down **the output** of the above program.

```
{ "name" : "John",
  "location" : {
    "latitude" : 38.58,
    "longitude" : -121.55 }
}
```

Q-4 4% (for HW-4) Given the following program segment (which is an implementation of JSON2Object for class GPS),

```
bool
GPS_DD::JSON2Object
(Json::Value arg_jv)
{
    if ((arg_jv.isNull() == true) ||
        (arg_jv.isObject() == true))  ← != true
    {
        return false;
    }

    if (((arg_jv["latitude"]).isNull() == true) ||
        ((arg_jv["longitude"]).isNull() == true) ||
        ((arg_jv["latitude"]).isString() != true) ||  ← isDouble()
        ((arg_jv["longitude"]).isString() != true))  ← isDouble()
    {
        return false;
    }

    this->latitude = (arg_jv["latitude"]).asDouble();
    this->longitude = (arg_jv["longitude"]).asDouble();

    return true;
}
```

Please 3% circle exactly THREE mistakes in the above code (please circle clearly), and also 1% write down what the correct code should have been.

Q-5 4% (for HW-5) This question is related to the source code under [https://github.com/sfelixwu/memory\\_layout](https://github.com/sfelixwu/memory_layout)

```
class Core
{
private:
public:
    unsigned int class_id;
    Core(void) {}
};

class Thing : public Core
{
private:
protected:
public:
    unsigned int model;
    unsigned int sequence_num;
    Thing(void) {}
};

int
main(void)
{
    Thing myThingObject;
    return 0;
}
```

Given the above code, if we use

➔ `clang -cc1 -std=c++14 -fdump-record-layouts`

We might obtain a memory layout like –

\*\*\* Dumping AST Record Layout

```
0 | class Core
0 |   unsigned int class_id
  | [sizeof=4, dsize=4, align=4,
  |   nvsize=4, nvalign=4]
```

\*\*\* Dumping AST Record Layout

```
0 | class Thing
0 |   (Thing vtable pointer)
8 |   unsigned int model
12 |  unsigned int sequence_num
16 |   class Core (virtual base)
16 |     unsigned int class_id
  | [sizeof=24, dsize=20, align=8,
  |   nvsize=16, nvalign=8]
```

4% Is the above memory layout correct for the code given? If no, please explain why?

The memory layout is incorrect as it contains vTable pointers, while the original source code has neither virtual inheritance nor virtual function.

Q-6 4% (for HW-6) This question is related to the source code under <https://github.com/sfelixwu/exception>

```
bool
GPS_DD::JSON2Object
(Json::Value input_json)
{
    ecs36b_Exception my_exception;
    my_exception.data["method"] = "JSON2Object";
    std::string sbuf;

    if (input_json.isNull() == true)
    {
        sbuf = "input is a NULL json";
        my_exception.data["message"] = sbuf;
        my_exception.data["error_code"] = EXC_ERR_INPUT_NULL;
        throw my_exception;
    }

    if (input_json.isObject() != true)
    {
        sbuf = "input is not a json object";
        my_exception.data["message"] = sbuf;
        my_exception.data["input"] = input_json;
        my_exception.data["error_code"] = EXC_ERR_INPUT_NOT_OBJECT;
        throw my_exception;
    }

    if ((input_json["latitude"]).isNull() == true)
    {
        sbuf = "missing required attribute \"latitude\"";
        my_exception.data["message"] = sbuf;
        my_exception.data["input"] = input_json;
        my_exception.data["error_code"] = EXC_ERR_MISSING_ATTRIBUTE;
        throw my_exception;
    }

    if ((input_json["longitude"]).isNull() == true)
    {
        sbuf = "missing required attribute \"longitude\"";
        my_exception.data["message"] = sbuf;
        my_exception.data["input"] = input_json;
        my_exception.data["error_code"] = EXC_ERR_MISSING_ATTRIBUTE;
        throw my_exception;
    }

    if ((input_json["latitude"]).isDouble() != true)
    {
        sbuf = "attribute \"latitude\" not double";
        my_exception.data["message"] = sbuf;
        my_exception.data["input"] = input_json;
        my_exception.data["error_code"] = EXC_ERR_DATA_TYPE;
        throw my_exception;
    }
}
```



```

    if ((input_json["longitude"]).isDouble() != true)
    {
        sbuf = "attribute \"longitude\" not double";
        my_exception.data["message"] = sbuf;
        my_exception.data["input"] = input_json;
        my_exception.data["error_code"] = EXC_ERR_DATA_TYPE;
        throw my_exception;
    }

    this->latitude = (input_json["latitude"]).asDouble();
    this->longitude = (input_json["longitude"]).asDouble();
    return true;
}

int
main(void)
{
    Json::Value myv;
    int rc = myFile2JSON("./myGPS.json", &myv);
    GPS_DD myLocation;
    bool b = myLocation.JSON2Object(myv);
    try {
        if (b == true) return 0;
        else return -1;
    } catch (ecs36b_Exception& eE) {
        char *what_ptr = (char *) eE.what();
        if (what_ptr == NULL)
        {
            std::cerr << "what_ptr NULL" << std::endl;
        }
        else
        {
            std::cerr << what_ptr << std::endl;
            free(what_ptr);
        }
    }
    return 0;
}

```

Given the above code segments, 2% what was definitely WRONG regarding the way exception handling was being implemented? Furthermore, 2% what kind of run-time errors might occur under what situation (maybe using an input GPS JSON example to illustrate)?

The line of code

```
bool b = myLocation.JSON2Object(myv);
```

(which was the only source code for triggering possible ecs36b\_exceptions)

Is NOT within the try/catch clause.

The runtime error due to this error will be “uncaught exceptions”

Q-7 4% Given the following segment of code under

<https://github.com/sfelixwu/exception>

```
class Thing : public virtual Core
{
private:
protected:
public:

    std::string model;
    std::string sequence_num;
    std::string description;
    SP_Person owner;
    JvTime purchased;
    JvTime sold;

    GPS_DD location;

    Thing(void);
    Thing(SP_Person);
    virtual void setClassName(void) { this->class_name = "Thing"; }
    virtual Json::Value dump2JSON(void);
    virtual bool JSON2Object(Json::Value);
};
```

And,

```
    Thing *my_thing_ptr;
    ...
    Json::Value myv = my_thing_ptr->dump2JSON(void);
    ...
```

4% Can the g++ compiler determine **which version/code** of **dump2JSON** to call **at the compiler time**? Why or why not? [maximum 5 sentences for the explanation]

No, the g++ compiler cannot determine the dump2JSON function version to call at the compiler time because dump2JSON is a virtual function.

Q-8 4% The following question of C++ Standard Template Library, is derived from the code under [https://github.com/sfelixwu/cpp\\_stl\\_map](https://github.com/sfelixwu/cpp_stl_map)

```
#include <iostream>
#include <map>
#include <string>
#include <string_view>

void
print_map
(std::string_view comment, const std::map<std::string, int>& m)
{
    std::cout << comment;

    for (const auto& n : m)
        std::cout << n.first << " = " << n.second << "; ";

    std::cout << '\n';
}

int main()
{
    std::map<std::string, int> m{{"Q1", 10}, {"Q3", 7}, {"Q7", 23}};
    m["Q1"] = 25;
    m["Q4"] = 30;
    std::map<std::string, int>::iterator it = m.lower_bound("Q2");
    it->second = 12;
    print_map("result -- \n", m);
    return 0;
}
```

4% What should be the correct output of the above program (please write down the output)?

Result --

Q1 = 25, Q3 = 12, Q4 = 30, Q7 = 23

Q-9 4% The following code segment was from `ecs36b_JSON.cpp`. The function `myFile2JSON()` reads a JSON file, and then converts it into a `Json::Value` object.

```
int
myFile2JSON
(char *f_name, Json::Value * jv_ptr)
{
    char *json_str = myFile2String(f_name);
    if (json_str == NULL) return -1;

    int rc = myParseJSON(json_str, jv_ptr);
    free(json_str);
    return rc;
}
```

Now, the code is modified as the following –

```
int
myFile2JSON
(char *f_name, Json::Value jv)
{
    char *json_str = myFile2String(f_name);
    if (json_str == NULL) return -1;

    int rc = myParseJSON(json_str, &jv);
    free(json_str);
    return rc;
}
```

4% Please answer whether the above modification is correct/appropriate? If not, why? [Hint: please consider the difference between call by value and call by reference.] (max 5 sentences)

The modification is wrong. The `myFile2JSON` call in the original code is call-by-reference, which means that any changes within the call to the `Json::Value` object pointed by `jv_ptr` will be observable by the caller of `myFile2JSON`.

However, the modified version is call-by-value, which means that the data from the file (represented by `f_name`) will still update the local variable `jv`, BUT, this update is useless to the caller of `myFile2JSON` (as it cannot see such update).

Q-10 4% When our C++ program called a class member function such as –  
    myThing\_ptr->dump2JSON(void)  
it involved not only the member function itself (e.g., dump2JSON( )) but also the affiliated object (e.g., the de-reference of myThing\_ptr). Therefore, our program (or the run-time system) was dealing with **two** different time moments related to this call. First, it was the time when the Object, the de-reference of myThing\_ptr, was constructed/created. Then, the second was the moment when the function myThing\_ptr->dump2JSON( ) was actually called.

For C++, under what situation that the Object construction moment would be important to the run-time support (i.e., the software supporting the execution of a C++ program) to do something extra? In other words, there are some extra work for the run-time support to handle things a bit differently under this specific situation. And, under this case, what information must be recorded (and where) **at run-time** when the object (such as the de-reference of myThing\_ptr) was **created**? (max two short paragraphs, Hint: this question is related polymorphism and function pointers in C++.)

If the run-time support/system creates an object with either virtual base or virtual function, it needs to not only reserve memory for vtable pointer, but also update the function pointer for the corresponding vtable at the moment when the object was created. This extra work is necessary for the program to indirectly find the right function to call at run time.

For Question 10, I submitted the question to ChatGPT and I got the following –

The object construction moment is important for the run-time support in C++ in situations where memory management is involved. C++ uses constructors and destructors to allocate and deallocate memory for objects. The construction moment becomes crucial when objects are dynamically allocated using operators like 'new'. In such cases, it is important to ensure proper memory allocation and initialization during object construction to avoid memory leaks or undefined behavior.

At run-time, when an object is created in C++, it is important to record information related to memory management. This includes details such as the size of the object, its memory address, and any other relevant metadata that facilitates memory allocation and deallocation. Additionally, if the object relies on external resources, such as file handles or network connections, it may be necessary to record and manage those resources as well, ensuring proper cleanup and release when the object is destroyed.

**[BTW, the above is absolutely the wrong answer! The correct answer should be related to the implementation of Polymorphism and Function pointers in C++.]**