

# CS103: Introduction to Programming

## Spring 2023 – Midterm 1 Exam

02/23/23, 7 PM – 8:30 PM

[Complete all the information in the box below.]

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

Student ID: \_\_\_\_\_ Email: \_\_\_\_\_@usc.edu

Lecture section (Circle One):

Redekopp	Redekopp	
MW 2 p.m.	MW 12:30 p.m.	

Ques.	Max score	Time
1	14	12 min.
2	8	10 min.
3	6	8 min.
4	6	12 min.
5	9	22 min.
6	12	26 min.
<b>Total</b>	55	

**Only work on this exam can be graded (No work on scratch paper will be considered)!**

## 1. Short Answer/Multiple Choice (12 pts)

Either fill in the blanks in the space provided OR circle the correct option to make the statement true.

- 1.1. **True/False:** When only the name of an array is used in an expression, it evaluates to the array's starting address.
- 1.2. What type would be returned by the expression `new int*[10];`  
(a) `int`                      (b) `int*`                      (c) `int &`                      (d) `int **`
- 1.3. **True/False:** A `break` statement will break out of all nested loops in which it is located.
- 1.4. When a function need not return a value, its return type should be declared as (one word): \_\_\_\_\_
- 1.5. **True/False:** Given a program composed of `file1.cpp` and `app.cpp`, we can produce an executable named `app` with the command: `g++ -g -Wall app.cpp -o app`
- 1.6. **True/False:** An array of pointers is a common approach to representing a 2D array.
- 1.7. **True/False:** Assuming the background of an image has been initialized, drawing a diagonal line using our image processing BMP library requires 2 nested loops.

For 1.8 - 1.11, assume an array is declared in `main()` using the statement:

`double mat[2][5][3];`. Indicate whether each statement is **true** or **false**.

- 1.8. **True/False:** The array values will be initialized to 0.
- 1.9. **True/False:** The middle dimension (`[5]`) must indicate the number of columns.
- 1.10. **True/False:** `mat[1][3][1]` is stored in memory directly next to `mat[0][3][1]`.
- 1.11. **True/False:** The address of the last element of the array can be found using the C++ expression: `(mat + 1*5*3 + 4*3 + 2)`

For 1.12 – 1.14, which of the following is/are necessary reason(s) to use dynamic allocation.

- 1.12. **True/False:** When you need a pointer to a variable.
- 1.13. **True/False:** When the allocated memory needs to live beyond the scope of the function where it is declared.
- 1.14. **True/False:** When the size of an array is constant.

## 2. Expressions (8 pts)

Consider the code below. Suppose it was compiled and then run with the command line:

```
./prog1 456 wxyz
```

Trace the execution and determine the value that would be printed on each line. Any update to a variable is **carried through the rest of the program**.

Note: You **MUST** show doubles with **ONLY** the decimal places that are necessary (e.g. **2.5** but not **2.50**) but with **at least 1 digit after the decimal** (i.e. **2.0** not **2**).

```
#include <iostream>
#include <cstring>
using namespace std;
// The program is run at the command line as: ./prog1 456 wxyz
int main(int argc, char* argv[])
{
    double a = 5.0, b = 3.0;
    int f = 6, g = 12, h = 94;
    char d[] = "fedc";

    double z = g / 8;

    cout << z << endl;           // 2.1) _____
    cout << g++ % 5 << endl;      // 2.2) _____
    cout << h / 10 % 6 << endl;   // 2.3) _____
    cout << 5 + f / a << endl;    // 2.4) _____
    cout << strlen(d) << endl;    // 2.5) _____
    cout << --d[1] << endl;      // 2.6) _____

    // Hint: no addresses will be printed by the following line
    cout << argv[1][1] << endl;  // 2.7) _____
    cout << d+2 << endl;         // 2.8) _____
    return 0;
}
```

### 3. Pointer and Pointers to Pointers (6 pts).

Suppose a program is run as:

```
./prog We can excel in 103
```

The program updates the signature of main() to be: `int main(int argc, char* argv[])`

Finally, assume the expression `argv` will yield address **600** and the actual content of `argv[1]` is **200**.

3.1. What is the value of `argc`?

3.2. What kind of result would be shown on the screen when `cout << *argv << endl;` is executed?

- a) a pointer
- b) a string of characters
- c) a single char
- d) none of the above

3.3. What kind of result would be shown on the screen when `cout << argv[2] << endl;` is executed?

- a) a pointer
- b) a string of characters
- c) a single char
- d) none of the above

3.4. What kind of result would be shown on the screen when `cout << argv+1 << endl;` is executed?

- a) a pointer
- b) a string of characters
- c) a single char
- d) none of the above

3.5. Show EXACTLY what would be printed on the screen when

`cout << *(argv[3]+1) << endl;` is executed (do not show " or ' in your answer).

\_\_\_\_\_

3.6. Show the address (type in the exact number) that would result from the expression `argv[1] + 2`.

\_\_\_\_\_

4. **Functions, Arguments, and Tracing (6 pts)** – Study the program below which prints 6 lines of output. Show the 6 lines of output exactly as they would be printed to the screen in the commented blanks after each `cout` line in `main()`.

**Note:** As shown below, when printing a `bool`, the value of `false` will be printed as 0 and `true` will be printed as 1. Just show **0** or **1** (not “false” or “true”). Assume the user inputs **5** for the first `cin` and **3** for the second.

```
#include <iostream> // SCRATCH WORK
using namespace std;

int f1(bool skiplist[], int c )
{
    int lc = -1;
    int p[] = {2, 5, 4, 1, 6, 0, 3};
    while( skiplist [p[c]] ){
        lc = c;
        c = p[c];
        skiplist[c] = false;
    }
    return lc;
}

int main()
{
    bool skip1[] = {true, false, true, true, false, false, false};
    bool skip2[] = {true, false, true, true, false, true, false};
    int choice, r;
    cin >> choice; // User enters 5
    r = f1(skip1, choice);

    cout << r << endl; // 4.1 _____

    cout << skip1[0] << endl; // 4.2 _____ (output 0 for false, 1 for true)

    cout << choice << endl; // 4.3 _____
    cin >> choice; // User enters 3
    r = f1(skip2, choice);

    cout << r << endl; // 4.4 _____

    cout << skip2[0] << endl; // 4.5 _____ (output 0 for false, 1 for true)

    cout << choice << endl; // 4.6 _____
    return 0;
}
```

5. **Coding 1 – Unique1 (9 pts)** – Consider the program below that takes an array of **non-negative** integers from the command line (of any length) and should find and output the number that has a unique **last 2 digits**. (To be clear, the last two digits of `103` are `03`).

More formally, for each number in the array consider **ONLY** its **last 2 digits**. We **guarantee** that only **1 number** in the array will have a **unique last 2 digits** (i.e. its last 2 digits will **NOT** match any other number), while the last 2 digits of **ALL OTHER NUMBERS** will match **1 or more** values in the array. **Find and output** that one number whose last 2 digits are unique.

For example, if we ran the program: `./unique 921 145 509 21 34509`

You should output **145**. This is because the last two digits of **921** match **21**, the last two digits of **509** match **34509**, but the last two digits of **145** has no match (i.e. the last two digits **45**, are unique). It is the number with the unique last 2 digits.

To implement this you must complete the function `unique()` and can determine its return value and other parameter types.

### Requirements

- You can **ONLY** write/modify 1 line in `main()` to call your function. You may not declare other variables, add more than 1 line, or alter any other part of `main()`.
- You may **NOT** use **C++ reference variables** which we have not learned nor covered yet this semester but that some students may know about.
- You may not change the return type of `unique()`.
- Bad (out-of-bounds) memory access will be penalized and code that is extremely inefficient may be assessed a small penalty.

```

#include <iostream>
#include <cstring>
using namespace std;

void unique(int dat[], int len, _____ ) {
    // Complete this function and its parameters above

}

int main(int argc, char* argv[]) {
    if(argc < 3) {
        cout << "list at least 2 integers on the cmd line" << endl;
        return 1;
    }
    int* dat = new int[argc-1];
    int len = argc-1;
    for(int i=0; i < len; i++){
        dat[i] = atoi(argv[i+1]);
    }
    int unique1;
    // **You may only update this ONE line below and no others.**
    // You may not declare other variables, add more lines, etc.

    unique(dat, len, _____);

    cout << "Unique number: " << unique1 << endl;
    delete [] dat;
    return 0;
}

```

6. **Coding 2 – str\_short\_long (12 pts)** – Consider the `str_short_long.cpp` program shown below to read in **two** C-strings (character arrays) of at most **20 alphanumeric characters**. Complete the function `short_long()` to determine which input string is **longer** (the first or second) and then produce an **output C-string of at most 40 alphanumeric characters** that contains:

- the shorter string first,
- followed by an underscore character ('\_'), and
- ends with the contents of the longer string UP TO BUT EXCLUDING the first digit character ('0'-'9') **if it exists (the longer string may or may not have a digit)**.

**Note: You MAY assume one string will ALWAYS be longer than the other.**

- **Example 1:** If the user enters `abc12345` and `wxyz`, the shorter would be placed first (i.e. `wxyz`) in the output string, followed by an **underscore**, and then everything in the longer string up to the first digit (i.e. `abc`). The final output string would, thus, be `wxyz_abc`.
- **Example 2:** If the user enters `hello` and `a123`, the output string should be `a123_hello`.
- **Example 3:** If the user enters `hi` and `123`, the output string should be `hi_` (i.e. "hi" and an underscore character).

### Procedure

1. Indicate the appropriate sizes of each character array declared in `main()`
2. Finish the code for the `short_long()` function. Note: You SHOULD NOT need an ASCII table (nor the decimal numbers associated with certain characters) to accomplish this task.

**Hint:** At a very high level, you can break this function into a sequence of 4 basic tasks:

- a. Find which is longer and which is shorter
- b. Copy the shorter string and then an underscore to the output string
- c. Find if and where a digit appears in the longer string
- d. Copy all the characters up until that point from the longer string to the output string

### Requirements and Assumptions

- **You may NOT use of C++ strings**, but may use the following `<cstring>` library functions if it helps:  
`int strlen(char* src);`  
`char* strcpy(char* dest, char* src);`  
`char* strcat(char* dest, char* src);`
- Bad (out-of-bounds) memory access will be penalized and code that is extremely inefficient may be assessed a small penalty.
- You may **NOT** modify `main()` [other than the character array sizes] **NOR** modify the signature of `str_short_long()`

```
// MODIFY THE ARRAY SIZES IN MAIN() BELOW
```

```
#include <iostream>
#include <cstring>
using namespace std;
```

```
void short_long(char s1[], char s2[], char ostr[]);
```

```
int main()
```

```
{
```

```
    char str1[_____];
```

```
    char str2[_____];
```

```
    char ostr[_____];
```

```
    // You may NOT modify anything in main() except for the array sizes above
```

```
    cout << "Enter two strings: " << endl;
```

```
    cin >> str1;
```

```
    cin >> str2;
```

```
    short_long(str1, str2, ostr);
```

```
    cout << ostr << endl;
```

```
    return 0;
```

```
}
```

```
// Write short_long() on the next page
```

```
void short_long(char s1[], char s2[], char ostr[]) {  
    // Your code here
```

```
}
```