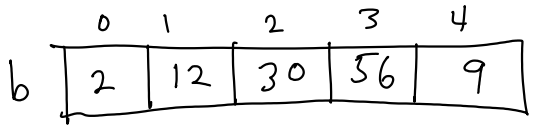
CS 101, Spring 2017 — Apr 11th — Exam 3 Name: Solution

Question 1. [5 points] What are the values in array `b` after the following code executes?

Assume the symbolic constant `SIZE` has been defined as `#define SIZE 5`

```
int a[SIZE] = {2,4,6,8,10};
int b[SIZE] = {1,3,5,7,9};

for (int i = 0; i < SIZE-1; i++) {
   b[i] = a[i] * b[i];
}
```

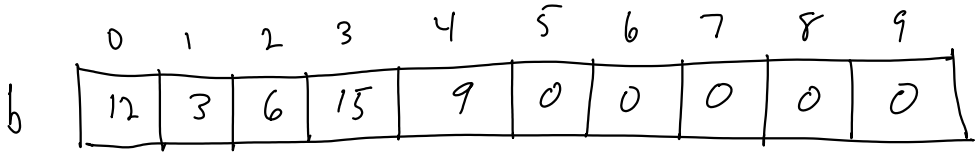


Question 2. [5 points] What are the values in array `b` after the following code executes?

Assume the symbolic constant `SIZE` has been defined as `#define SIZE 10`

```
int a[SIZE] = {12,3,5,6,15,23,7,8,4,9};
int b[SIZE] = {0,0,0,0,0,0,0,0,0,0};
int count = 0;

for (int i = 0; i < 10; i++) {
    if (a[i] % 3 == 0) {   // if a[i] is a multiple of 3
       b[count] = a[i];    // copy a[i] to b[count]
       count++;            // increment count
    }
}
```



Question 3. [5 points] Consider the following code:

```
int randNum = 0;

for (int i = 1; i <= 4; i++) {
    randNum += (rand() % 101);  // add a random value in range 0..100 to randNum
}
// randNum is in range 0..400
randNum -= 200;  //randNum now in range -200..200
printf("%i\n", randNum);
```

(a) What is the *smallest* possible value that can be printed by the `printf` statement?

−200

(b) What is the *largest* possible value that can be printed by the `printf` statement?

200

Question 4. [10 points] Identify and correct the error(s) in the following code:

```
int nums[10] = {9,2,14,5,6,3,18,11,7,10};
int count;
 count = 0;    — count must be initialized
printf("Entries > 10: ");

// find and print the numbers > 10 in the array
for (int i = 0; i <= 10; i++) {
  if (nums[i] > 10) {
     printf("%i ", nums);      — should be <
     count++;
  }                            — should be nums[i]
}
printf("\n");
printf("There are %i numbers > 10.\n",count);
```

Question 5. [10 points] Complete the following code so that the array `powersOf2[10]` contains the powers of 2 from 2^0 to 2^9. So, `powersOf2[0]` should equal 1, `powersOf2[1]` should equal 2, `powersOf2[2]` should equal 4, etc. Note that you *must* use a loop: don't just assign to individual array elements.

```
int powersOf2[10];
```

```
int prod = 1;
for (int i = 0; i < 10; i++) {
    powersOf2[i] = prod;
    prod = prod * 2;
}
```

Question 6. [5 points] What output is printed by the following program, which starts on the left and continues on the right?

```
#include <stdio.h>

void myfunc(int x);

int main(void) {
  int q = 12;
  myfunc(q);
  printf("q=%i\n", q);
  return 0;
}
```

```
void myfunc(int x) {
  x = x + 1;
  printf("x=%i\n", x);
}
```

x is a different variable than q: assigning to it doesn't change the value of q

x = 13
q = 12

Question 7. [5 points] Assume that the `compute_score` function has the following prototype:

```
int compute_score(double dist);
```

In the code below, add a call to `compute_score` so that a score is calculated for `arrow_dist` and then assigned to `arrow_score`. (Note that you do not need to *define* `compute_score`, just call it.)

```
double arrow_dist;
printf("Enter arrow distance: ");
scanf("%lf", &arrow_dist);

int arrow_score;
arrow_score = compute_score(arrow_dist);
printf("Arrow's score is %i\n", arrow_score);
```

Question 8. [10 points] Consider the following code:

```
printf("Enter an integer: ");
int n;
scanf("%i", &n);
int rounded = roundToNearest10(n);
printf("Rounded to the nearest 10, that is %i\n", rounded);
```

The `roundToNearest10` function rounds an integer up or down to the nearest multiple of 10. For example, if the input value were **47**, the code above would print the output

```
Rounded to the nearest 10, that is 50
```

(a) Write a prototype for the `roundToNearest10` function.

```
int roundToNearest10(int x);
```

(b) Write a definition for the `roundToNearest10` function.

```
int roundToNearest10(int x) {
    int rem = x % 10;
    if (rem > 5) {
        x = x + (10 - rem); //round up
    } else {
        x = x - rem;        //round down
    }
    return x;
}
```

Programming Questions

Note: For all of the programming questions, you should use `scanf` to read the input value(s) required by the program.

Note: Make sure your programs produce the output in **exactly** the format described, including capitalization and punctuation. You may not receive credit for programs that produce incorrectly-formatted output.

Getting started: Start **Cygwin Terminal** and **Notepad++** and make sure ALL TABS are closed. (Note: do *not* open any other programs.) Your instructor will give you the name of a zip file. In your terminal, run the following commands:

```
cd h:
mkdir -p CS101
cd CS101
curl -O http://faculty.ycp.edu/~dhovemey/spring2017/cs101/zipfile
unzip zipfile
cd CS101_Exam03
```

Note that in the `curl` command, the `-O` has the letter 'O', not the digit '0'.

Substitute the name of the zip file for *zipfile*.

Editing code: Use your text editor to open the source file (e.g., `question9.cpp`) referred to in the question. Do not open any files other than the ones for the exam.

Compiling: To compile the program for Question 9, run the following command in the terminal:

```
make question9.exe
```

Change the number as appropriate for the other questions (e.g., `question10.exe`).

Running: To run the program for Question 9, run the following command in the terminal:

```
./question9.exe
```

Change the number as appropriate for the other questions (e.g., `question10.exe`).

To submit: In Cygwin Terminal, run the command

```
make submit
```

Enter your Marmoset username and password when prompted.

Good luck!

Question 9. [25 points] Complete the program in `question9.cpp` as follows. The program should first prompt the user to specify a number of values and weights to be entered. Second, it should read exactly that many values into the `values` array. Third, it should read the same number of weights into the `weights` array. Finally, it should compute the *weighted average* of the values and weights.

A weighted average is defined in the following way. Given a sequence of n values

$$v_0, v_1, v_2, \ldots, v_{n-1}$$

and a sequence of n weights

$$w_0, w_1, w_2, \ldots, w_{n-1}$$

such that the sum of the weights is 1, the weighted average is the sum

$$w_0 v_0 + w_1 v_1 + w_2 v_2 + \ldots + w_{n-1} v_{n-1}$$

Example session (user input in **bold**):

```
How many values? 4
Enter values: 14.5 23.33 101.32 4.04
Enter weights: .05 .33 .41 .21
Weighted average is 50.81
```

Another example session (user input in **bold**):

```
How many values? 3
Enter values: 673.1 808.9 320.3
Enter weights: .65 .13 .22
Weighted average is 613.14
```

Hints/specifications:

- Store the number of values/weights in an `int` variable
- Use `for` loops to read the values and the weights (read all of the values first, then read all of the weights), storing them in the `values` and `weights` arrays
- Use a `for` loop to compute the product of each weight/value pair and incorporate the product into the weighted average
- Print the weighted average using two decimal places of precision after the decimal point
- Make sure your program produces the correct output for the example inputs shown above

Question 10. [20 points] Complete the program in `question10.cpp` to add a prototype and a definition for the `ipow` function.

The `ipow` function takes two parameters: an integer base b, and an integer exponent x. It should return the integer whose value is

$$b^x$$

For example, if $b = 3$ and $x = 4$, `ipow` should return 81, because $3 \times 3 \times 3 \times 3 = 81$.

A `main` function is provided which reads two integer values, calls `ipow`, and prints the result. **Important**: do *not* modify the `main` function in any way. Your only modification to the program is to add a prototype and definition for `ipow`.

Example run (user input in **bold**):

```
Enter base: 3
Enter exponent: 4
3^4 = 81
```

Another example run (user input in **bold**):

```
Enter base: 5
Enter exponent: 3
5^3 = 125
```

Hints/specifications:

- Do *not* call the built-in `pow` function; instead, use a loop to compute the result
- Products should start at 1 (unlike sums, which should start at 0)
- Make sure your program produces the correct output for the example inputs shown above