APS 105 — Computer Fundamentals
Lab #5: Functions and Arrays
Winter 2023

You must use `examify.ca` to electronically submit your solution by 11:59 pm on **Saturday, February 18, 2023**.

Objective

In this lab, you will be writing one C program. You will gradually develop the program by writing a few functions in each part. In each part, you will only be marked on the functions you submit in that part, not the entire program. Parts are independent of each other. This means even if you cannot do one part, you will gain marks of the parts you can do. This is the beauty of a modular program. You can divide the problem into sub-problems, and solve each problem independently. Input and output must be done using `scanf` and `printf`.

In the sample output examples that follow, the text that would be entered by the program user is displayed using a **bold** font. The text <**enter**> stands for the user pressing the `enter` key on the keyboard. On some keyboards, the `enter` key may be labeled `return`. When you execute your programs, the user's text will not be displayed in bold and <**enter**> will not be shown.

Once again, place your solution for each part in a separate directory, so that Visual Studio Code will not attempt to compile multiple files and link them together. Each of your directories should have one `.c` file corresponding to your solution for the individual part of the exercise. Please ensure you are selecting the folder with the `.c` you want to compile.

Game explanation

You are asked to develop a program for a game between two players. The game starts with 20 boxes, where each box may have a candy, bomb or nothing. In each round of the game, each player is asked to pick 5 boxes. It is **unknown** to the players what is inside the boxes, or which boxes the other player picked.

Score. If a player picks a box with a bomb, he/she loses 10 points. If a player selects a box with candy, they will earn 10 points if no other player has picked it, or receive 5 points if the candy happens to be already chosen by another player, i.e. **the 10 points will be shared amongst the two players**. The player gains no points if they pick an empty box. This applies to all the boxes the player choose.

Repetition. In the next round, the boxes are reset with different locations of candies and bombs, and again the players can pick 5 boxes. The game ends when the score difference between the players is more than 50 points. The player with the highest score wins the game.

Part 1 — Populate the boxes and validate the players' choices

In this part we will develop two functions with prototypes:

```
void populateBoxes(int boxes[], const BoxesNum);
```

```
bool validateChoices(int choices[], const int ChoicesNum, const int BoxesNum);
```

The function `populateBoxes` will populate the boxes with candies, bombs and nothing. The function receives an array of integers and the size of this array as inputs. It then randomly populates each element in the array with 0, 10 and -10, and prints the contents of each element. **10 corresponds to a candy, 0 corresponds to nothing/empty and -10 corresponds to a bomb.** Use the function `rand()` to generate random numbers. Do **not** use `srand` function to seed the random number generator. The function does not return any values, hence return type is `void`.

The function `validateChoices` takes in an array that has a player's choices of boxes and the size of this array. It then returns a boolean value. The function will return `true` if the choices are valid and `false` if the choices are not valid. The choices are valid if they are between 0 and 19 and if the choices are distinct, *i.e.* the same player cannot pick the same box more than once. You can assume that the array will have increasing number of indices. For example, `int choices[] = {0, 1, 2, 3, 4};` is valid, but `int choices[] = {0, 1, 2, 3, 3};` is not valid, because 3 is repeated. Also, `int choices[] = {0, 1, 5, 10, 20};` is invalid, because 20 is not valid index. Finally, `int choices[] = {0, 1, 5, 10, 2};` will not be entered by the user since 2 is smaller than 10. **We assume the user enters the indices in ascending order**.

Testing and submission. To test your functions in VS Code, we have provided a skeleton for your program on Quercus in `lab5part1-skeleton.c` and Examify. However, do not submit the `main` function, function prototypes and includes. Only submit the `populateBoxes` and `validateChoices` functions you developed. We will append the `main` function, function prototypes and includes from our end.

Example outputs if you test your code using the skeleton given in `lab5part1-skeleton.c`**:**
Please note random numbers generated were based on rand() on Examify. So you may not see the same random numbers on your computer. However, your outputs must be the same when you test against the test cases on Examify.
Example 1:

```
0: 0, 1: 0, 2: -10, 3: 0, 4: 10, 5: 0, 6: 0, 7: -10, 8: -10, 9: 0, 10: 10,
11: 0, 12: 10, 13: 0, 14: 10, 15: 0, 16: -10, 17: -10, 18: 0, 19: 0,
Player 1, please enter distinct box choices between 0 and 19: 4 5 6 7 20<enter>
Player 1, please enter distinct box choices between 0 and 19: 4 5 6 7 8<enter>
Player 2, please enter distinct box choices between 0 and 19: 4 5 5 7 8<enter>
Player 2, please enter distinct box choices between 0 and 19: 4 5 12 16 19<enter>
```

Example 2:

```
0: 0, 1: 0, 2: -10, 3: 0, 4: 10, 5: 0, 6: 0, 7: -10, 8: -10, 9: 0, 10: 10,
11: 0, 12: 10, 13: 0, 14: 10, 15: 0, 16: -10, 17: -10, 18: 0, 19: 0,
Player 1, please enter distinct box choices between 0 and 19: 7 18 19 20 21<enter>
Player 1, please enter distinct box choices between 0 and 19: 6 7 12 17 19<enter>
Player 2, please enter distinct box choices between 0 and 19: 0 0 1 1 3<enter>
Player 2, please enter distinct box choices between 0 and 19: 0 7 9 10 11<enter>
```

Example 3:

```
0: 0, 1: 0, 2: -10, 3: 0, 4: 10, 5: 0, 6: 0, 7: -10, 8: -10, 9: 0, 10: 10,
11: 0, 12: 10, 13: 0, 14: 10, 15: 0, 16: -10, 17: -10, 18: 0, 19: 0,
Player 1, please enter distinct box choices between 0 and 19: 6 9 10 15 19<enter>
Player 2, please enter distinct box choices between 0 and 19: 9 11 15 16 17<enter>
```

Part 2 — Get statistics of the game

In this part, we will develop two functions to find the most frequent box chosen by a player. The skeleton that you can use to test your code on VS Code is on Quercus in `lab5part2-skeleton.c` and on Examify. You are only required to submit the functions you developed on Exmaify. We will append the `main` function, function prototypes and includes from our end. The function prototypes are:

```
void appendStatistics(int userChoice[], const int ChoicesNum, int histogram[]);.
int frequentBox(int histogram[], const int BoxesNum);
```

The function `appendStatistics` takes in an array having a player's choice of boxes, size of the array, and an array having a histogram. Each index in the histogram corresponds to a box. The value at each index is the number of times the player picked that box. For example, if the player picked box 0 twice, the value at index 0 in the histogram will be 2. If the player picked box 0 once and box 1 once, the value at index 0 in the histogram will be 1 and the value at index 1 in the histogram will be 1. The function does not return anything. You can print the histogram in the `main` function in the skeleton to test your code.

The function `frequentBox` takes in a histogram and returns the most frequently chosen box. If there are multiple boxes that are chosen the same number of times, the function will return the smallest box number. For example, if the player picked box 0 twice and box 1 once, the function will return 0. If the player picked box 0 once and box 1 once, the function will return 0. If the player picked box 0 once and box 1 twice, the function will return 1.

Example outputs:
Example 1:

```
Enter choices: 7 8 9 10 11<enter>
0: 0, 1: 0, 2: 0, 3: 0, 4: 0, 5: 0, 6: 0, 7: 1, 8: 1, 9: 1, 10: 1, 11: 1,
12: 0, 13: 0, 14: 0, 15: 0, 16: 0, 17: 0, 18: 0, 19: 0,
Enter choices: 8 9 10 11 12<enter>
0: 0, 1: 0, 2: 0, 3: 0, 4: 0, 5: 0, 6: 0, 7: 1, 8: 2, 9: 2, 10: 2, 11: 2,
12: 1, 13: 0, 14: 0, 15: 0, 16: 0, 17: 0, 18: 0, 19: 0,
Enter choices: 7 13 14 16 17<enter>
0: 0, 1: 0, 2: 0, 3: 0, 4: 0, 5: 0, 6: 0, 7: 2, 8: 2, 9: 2, 10: 2, 11: 2,
12: 1, 13: 1, 14: 1, 15: 0, 16: 1, 17: 1, 18: 0, 19: 0,
Enter choices: 12 13 15 18 19<enter>
0: 0, 1: 0, 2: 0, 3: 0, 4: 0, 5: 0, 6: 0, 7: 2, 8: 2, 9: 2, 10: 2, 11: 2,
12: 2, 13: 2, 14: 1, 15: 1, 16: 1, 17: 1, 18: 1, 19: 1,
Enter choices: 0 11 10 14 18<enter>
0: 1, 1: 0, 2: 0, 3: 0, 4: 0, 5: 0, 6: 0, 7: 2, 8: 2, 9: 2, 10: 3, 11: 3,
12: 2, 13: 2, 14: 2, 15: 1, 16: 1, 17: 1, 18: 2, 19: 1,
Enter choices: 7 8 12 16 19<enter>
0: 1, 1: 0, 2: 0, 3: 0, 4: 0, 5: 0, 6: 0, 7: 3, 8: 3, 9: 2, 10: 3, 11: 3,
12: 3, 13: 2, 14: 2, 15: 1, 16: 2, 17: 1, 18: 2, 19: 2,
The smallest and most frequently used box is: 7
```

Example 2:

```
Enter choices: 1 12 17 18 19<enter>
0: 0, 1: 1, 2: 0, 3: 0, 4: 0, 5: 0, 6: 0, 7: 0, 8: 0, 9: 0, 10: 0, 11: 0,
12: 1, 13: 0, 14: 0, 15: 0, 16: 0, 17: 1, 18: 1, 19: 1,
Enter choices: 3 4 8 12 14<enter>
0: 0, 1: 1, 2: 0, 3: 1, 4: 1, 5: 0, 6: 0, 7: 0, 8: 1, 9: 0, 10: 0, 11: 0,
12: 2, 13: 0, 14: 1, 15: 0, 16: 0, 17: 1, 18: 1, 19: 1,
Enter choices: 1 2 4 6 8<enter>
0: 0, 1: 2, 2: 1, 3: 1, 4: 2, 5: 0, 6: 1, 7: 0, 8: 2, 9: 0, 10: 0, 11: 0,
12: 2, 13: 0, 14: 1, 15: 0, 16: 0, 17: 1, 18: 1, 19: 1,
Enter choices: 9 11 12 15 17<enter>
0: 0, 1: 2, 2: 1, 3: 1, 4: 2, 5: 0, 6: 1, 7: 0, 8: 2, 9: 1, 10: 0, 11: 1,
12: 3, 13: 0, 14: 1, 15: 1, 16: 0, 17: 2, 18: 1, 19: 1,
Enter choices: 5 11 10 14 18<enter>
0: 0, 1: 2, 2: 1, 3: 1, 4: 2, 5: 1, 6: 1, 7: 0, 8: 2, 9: 1, 10: 1, 11: 2,
12: 3, 13: 0, 14: 2, 15: 1, 16: 0, 17: 2, 18: 2, 19: 1,
Enter choices: 7 8 12 16 19<enter>
0: 0, 1: 2, 2: 1, 3: 1, 4: 2, 5: 1, 6: 1, 7: 1, 8: 3, 9: 1, 10: 1, 11: 2,
12: 4, 13: 0, 14: 2, 15: 1, 16: 1, 17: 2, 18: 2, 19: 2,
The smallest and most frequently used box is: 12
```

Example 3:

```
Enter choices: 9 10 13 15 19<enter>
0: 0, 1: 0, 2: 0, 3: 0, 4: 0, 5: 0, 6: 0, 7: 0, 8: 0, 9: 1, 10: 1, 11: 0,
12: 0, 13: 1, 14: 0, 15: 1, 16: 0, 17: 0, 18: 0, 19: 1,
Enter choices: 13 14 15 16 17<enter>
0: 0, 1: 0, 2: 0, 3: 0, 4: 0, 5: 0, 6: 0, 7: 0, 8: 0, 9: 1, 10: 1, 11: 0,
12: 0, 13: 2, 14: 1, 15: 2, 16: 1, 17: 1, 18: 0, 19: 1,
Enter choices: 15 16 17 18 19<enter>
0: 0, 1: 0, 2: 0, 3: 0, 4: 0, 5: 0, 6: 0, 7: 0, 8: 0, 9: 1, 10: 1, 11: 0,
12: 0, 13: 2, 14: 1, 15: 3, 16: 2, 17: 2, 18: 1, 19: 2,
Enter choices: 0 10 11 12 13<enter>
0: 1, 1: 0, 2: 0, 3: 0, 4: 0, 5: 0, 6: 0, 7: 0, 8: 0, 9: 1, 10: 2, 11: 1,
12: 1, 13: 3, 14: 1, 15: 3, 16: 2, 17: 2, 18: 1, 19: 2,
Enter choices: 12 13 14 15 16<enter>
0: 1, 1: 0, 2: 0, 3: 0, 4: 0, 5: 0, 6: 0, 7: 0, 8: 0, 9: 1, 10: 2, 11: 1,
12: 2, 13: 4, 14: 2, 15: 4, 16: 3, 17: 2, 18: 1, 19: 2,
Enter choices: 5 9 10 13 14<enter>
0: 1, 1: 0, 2: 0, 3: 0, 4: 0, 5: 1, 6: 0, 7: 0, 8: 0, 9: 2, 10: 3, 11: 1,
12: 2, 13: 5, 14: 3, 15: 4, 16: 3, 17: 2, 18: 1, 19: 2,
The smallest and most frequently used box is: 13
```

Part 3 — Calculate the score of each player

In this part, we develop the function that updates the scores of the players. The skeleton that you can use to test your code on VS Code is on Quercus in `lab5part3-skeleton.c` and on Examify. You are only required to submit the function on Exmaify. We will append the `main` function, function prototypes and includes from our end. The function prototype is:

```
void calculateScore(int boxes[], const int BoxesNum, int userOne[], int userTwo[],
 const int ChoicesNum, int* score1, int* score2);.
```

The function takes as input:

- `boxes[]`: an array of integers that represents the content of the boxes.
- `BoxesNum`: the number of boxes, or size of `boxes` array.
- `userOne[]`: an array of integers that represents the choices of the first player. The elements of the array are valid.
- `userTwo[]`: an array of integers that represents the choices of the second player. The elements of the array are valid.
- `ChoicesNum`: the number of choices of each player, or size of `userOne` and `userTwo` arrays.
- `score1`: pointer to an `int` storing the score of the first player.
- `score2`: pointer to an `int` storing the score of the second player.

The function should update the score of each player by updating `*score1` and `*score2`. The function will check the contents of each box. If it has a bomb or candy, it will check which player picked this box. If a player (or players) picked it, update their score. Along the way, the function should print the following messages:

Example outputs. Example 1:

```
Enter boxes content: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 10<enter>
Enter player 1 choices: 0 1 2 3 4<enter>
Enter player 2 choices: 0 1 2 3 19<enter>
Found 10 in boxes[19].
Found index 19 in player 2.
+10 to player 2 score.
Player 1: 0.
Player 2: 10.
```

Example 2:

```
Enter boxes content: 10 -10 10 -10 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0<enter>
Enter player 1 choices: 0 1 2 3 4<enter>
Enter player 2 choices: 0 1 5 6 8<enter>
Found 10 in boxes[0].
Found index 0 in player 1.
Found index 0 in player 2.
+5 to players 1 and 2 scores.
Found -10 in boxes[1].
Found index 1 in player 1.
Found index 1 in player 2.
-10 from player 1 score.
-10 from player 2 score.
Found 10 in boxes[2].
Found index 2 in player 1.
+10 to player 1 score.
Found -10 in boxes[3].
Found index 3 in player 1.
-10 from player 1 score.
Player 1: -5.
Player 2: -5.
```

Example 3:

```
Enter boxes content: 0 10 0 -10 10 10 0 0 0 10 0 0 0 -10 0 0 10 0 0 0<enter>
Enter player 1 choices: 0 5 7 9 11<enter>
Enter player 2 choices: 7 9 13 17 18<enter>
Found 10 in boxes[1].
Found -10 in boxes[3].
Found 10 in boxes[4].
Found 10 in boxes[5].
Found index 5 in player 1.
+10 to player 1 score.
Found 10 in boxes[9].
Found index 9 in player 1.
Found index 9 in player 2.
+5 to players 1 and 2 scores.
Found -10 in boxes[13].
Found index 13 in player 2.
-10 from player 2 score.
Found 10 in boxes[16].
Player 1: 15.
Player 2: -5.
```

Part 4 — Add the loop

In this part you are asked to add a loop in the `main` function. The loop should continue to ask the players to enter their choices until the difference in score between the players is **more than 50**. The `main` function you are required to complete is shown below, and is on Quercus in `lab5part4-skeleton.c` and on Examify.

```
int main(void) {
const int BoxesNum = 20;
const int ChoicesNum = 5;
int boxes[BoxesNum], userOne[ChoicesNum], userTwo[ChoicesNum],
    histogramUserOne[BoxesNum] = {0}, histogramUserTwo[BoxesNum] = {0};
int userOneScore = 0, userTwoScore = 0;
```

```
  populateBoxes(boxes, BoxesNum);

  takeUserChoices(userOne, userTwo, ChoicesNum, BoxesNum);

  calculateScore(boxes, BoxesNum, userOne, userTwo, ChoicesNum, &userOneScore,
               &userTwoScore);


  appendStatistics(userOne, ChoicesNum, histogramUserOne);

  appendStatistics(userTwo, ChoicesNum, histogramUserTwo);

  printf("Player 1: %d.\nPlayer 2: %d.\n", userOneScore, userTwoScore);

  if (userOneScore > userTwoScore) {
    printf("User 1 wins!");
    printf("and the first box they chose most frequently is: %d\n",
           frequentBox(histogramUserOne, BoxesNum));
  } else {
    printf("User 2 wins!");
    printf("and the first box they chose most frequently is: %d\n",
           frequentBox(histogramUserTwo, BoxesNum));
  }
  return 0;
}
```

On Examify, submit only the main function. When you append the functions you implemented in Parts 1, 2 and 3 to the main function in the `lab5part4-skeleton.c` file, the output of the program should look as shown below. **In this part, if your previous functions were not correct, you will be able to pass the test cases if the main function you submit is correct on Examify!**

Example output.
Example 1:

```
0: 0, 1: 0, 2: -10, 3: 0, 4: 10, 5: 0, 6: 0, 7: -10, 8: -10, 9: 0, 10: 10, 11: 0,
12: 10, 13: 0, 14: 10, 15: 0, 16: -10, 17: -10, 18: 0, 19: 0,
Player 1, please enter distinct box choices between 0 and 19: 0 2 7 8 19<enter>
Player 2, please enter distinct box choices between 0 and 19: 4 10 13 14 15<enter>
Found -10 in boxes[2].
Found index 2 in player 1.
-10 from player 1 score.
Found 10 in boxes[4].
Found index 4 in player 2.
+10 to player 2 score.
Found -10 in boxes[7].
Found index 7 in player 1.
-10 from player 1 score.
Found -10 in boxes[8].
Found index 8 in player 1.
```

```
-10 from player 1 score.
Found 10 in boxes[10].
Found index 10 in player 2.
+10 to player 2 score.
Found 10 in boxes[12].
Found 10 in boxes[14].
Found index 14 in player 2.
+10 to player 2 score.
Found -10 in boxes[16].
Found -10 in boxes[17].
Player 1: -30.
Player 2: 30.
User 2 wins, and the first box they chose most frequently is: 4
```

Marking

This lab will be marked out of 10, 3 marks for Part 1, 3 marks for Part 2, 3 marks for Part 3, and 1 mark for Part 4. The automarker may use multiple test cases in each part, and if this is the case, the marks for this part will be evenly split between the test cases. The test cases used for marking may or may not be the same as the test cases that are made available to you. The deadline is strictly enforced **(11:59 pm on Saturday, February 18, 2023)**, so avoid last minute submissions.