

Programming Problems

14th Annual Computer Science Programming Contest

Department of Mathematics and Computer Science
Western Carolina University
April 8, 2003

Criteria for Determining Team Scores

Each program should:

1. execute without error, solve the appropriate problem efficiently, and satisfy all the conditions specified in the problem statement
2. follow good programming style conventions such as avoiding confusing and unnecessary code,
3. be documented with comments and easily understandable variable names, and
4. start with an initial comment that includes a) the name of the team's school, b) the team's number, and c) the names of the team's members. An example such comment is

// Somewhere High School, Team 2, Susan Smith, Shelley Jones, and Sandy Hart

The score assigned to each program will be based upon the extent that the program satisfies the properties listed above. The total accumulated score for all four programs represents the team score. All the programs have the same weight in the scoring.

Notes

- Each team may use one C++ reference book.
- You can use up to two computers if you wish.
- **Inappropriate code comments will result in a lower score and is grounds for disqualification for an award.**

Problem 1: Game of Nim

You are to program the computer to play a modified version of the game of Nim, sometimes known as 21, with a user. The game proceeds in rounds. Consider that there are 21 markers on the floor. Players take turns picking up 1,2, or 3 markers. The object is to force your opponent to pick up the last marker.

At the beginning of each round

1. The computer announces the number of markers left.
2. The computer allows the user to specify how many markers (between 1 and 3) he or she would like to pick up for this turn and reduces the number left by that amount.
3. The computer announces the number of markers left.
4. The computer then chooses how many markers (between 1 and 3) it would like to pick up for this turn and reduces the number left by that amount.

If at any time, there is only one marker left, the computer ends the game and announces a winner. Note that you do not need to program a winning strategy for the computer; it may make a random choice for its turn.

Here is a sample run of the program (computer output in italics, user's input in bold):
Welcome to the game of Nim.

There are 21 markers left. Enter your choice: **3**

There are 18 markers left. Computer chooses 1.

There are 17 markers left. Enter your choice: **2**

There are 15 markers left. Computer chooses 2.

There are 13 markers left. Enter your choice: **3**

There are 10 markers left. Computer chooses 2.

There are 8 markers left. Enter your choice: **1**

There are 7 markers left. Computer chooses 2.

There are 5 markers left. Enter your choice: **1**

There are 4 markers left. Computer chooses 2.

There are 2 markers left. Enter your choice: **1**

There is 1 marker left. Computer loses.

Other constraints:

- You must do input verification, that is, you must ensure that the user can only enter 1 or 2. If the user enters anything else, you must provide an error message and have the user choose again.
- You must also ensure that neither the user nor the computer chooses all remaining markers. If the user makes such a choice, you must provide an error message and have the user choose again.

As an example, here is a game in progress:

There are 7 markers left. Computer chooses 2.

There are 5 markers left. Enter your choice: 4

You must choose an integer between 1 and 3. Please try again.

There are 5 markers left. Enter your choice: 1

There are 4 markers left. Computer chooses 1.

There are 3 markers left. Enter your choice: 3

You may not remove all markers. Please try again.

There are 3 markers left. Enter your choice: 2

There is 1 marker left. Computer loses.

Problem 2: Calendar Creator

Write a program that creates a calendar for a given month and year. The first line of your output should be the month (in English) and year (4 digits), the next line should be a list of column headings for the days of the week (use the first two letters of each day; e.g., “Su”, “Mo”). The following lines should be the dates as they would be listed on a calendar for that month and year.

In order to achieve this, you’ll need a formula to calculate the first day of the month for a given month and year. Here is such a formula, adapted from <http://webexhibits.org/calendars/week.html>:

Let $a = [(14 - \text{month})/12]$
 $y = \text{year} - a$
 $m = \text{month} + (12 * a) - 2$

Compute

$6 + y + \text{integer part of } (y/4) - \text{integer part of } (y/100) + \text{integer part of } (y/400) + \text{integer part of } ((31 * m)/12)$

Divide that result by 7. The remainder indicates the day of the week on which the month begins; e.g., “0” for Sunday, “1” for Monday, “6” for Saturday.

Here is a sample run of the program:

Enter the number for the month: **3**

Enter the year (4 digits): **2003**

March 2003

<i>Su</i>	<i>Mo</i>	<i>Tu</i>	<i>We</i>	<i>Th</i>	<i>Fr</i>	<i>Sa</i>
						<i>1</i>
<i>2</i>	<i>3</i>	<i>4</i>	<i>5</i>	<i>6</i>	<i>7</i>	<i>8</i>
<i>9</i>	<i>10</i>	<i>11</i>	<i>12</i>	<i>13</i>	<i>14</i>	<i>15</i>
<i>16</i>	<i>17</i>	<i>18</i>	<i>19</i>	<i>20</i>	<i>21</i>	<i>22</i>
<i>23</i>	<i>24</i>	<i>25</i>	<i>26</i>	<i>27</i>	<i>28</i>	<i>29</i>
<i>30</i>	<i>31</i>					

You will need to pay attention to the number of days to print for Feb, depending on whether it is a leap year or not. Feb has 29 days in a leap year. A leap year is defined as any year that is divisible by 4, except those years divisible by 100 but not 400 (thus 1900 was *not* a leap year, but 2000 *was* a leap year).

Your output should be nicely formatted and easily readable.

Problem 3: Balanced Parentheses

Write a program that determines whether a given line of input has a balanced number of parentheses or not. The input may contain more symbols than parentheses, but you are only concerned about the parentheses.

Here is the definition of balanced parentheses:

- As you move from left to right through the input string, you should never encounter more right parentheses than left parentheses.
- The total number of left parentheses must equal the total number of right parentheses.

Here are some examples of strings with balanced parentheses:

```
abcdfg  
(abc)(def)  
(abcdfghekjls)  
()  
((abc(ef)(gh)(ij)(hijklk)(jkljdk))((fdkjf)(kdjfk)))  
((())())
```

Here are some examples of strings that do not have balanced parentheses:

```
(abscd  
dkjdfd)  
(fdadf)(  
(ka)ad)(jk(dj))
```

Your program should accept one line of input, and state whether or not that line has a balanced set of parentheses or not.

Problem 4: Random “Planet” Walk

Write a program that conducts a “random walk” on a 2-dimensional grid.

- The size of the grid should be n by n , where n is a random number between 5 and 10.
- The walk should begin on a square of the grid that is randomly chosen.
- Each step consists of moving one square vertically, horizontally, or diagonally (thus, there are eight possible choices for a next move).
- The number of steps should be n^4 (thus, for an 8 by 8 grid, the number of steps will be 4096).
- The walker should be able to “wrap around” the square. For instance, if the walker is currently in a leftmost square, and the next step choice is directly to the left, the walker ends up in the corresponding rightmost square. If the walker is in a topmost square, and the next step choice is up and to the right, the walker ends up one square to the right of the corresponding bottommost square.
- Here is a visual example of the “wrap around” concept using a 5 by 5 grid:

Consider the following squares, numbered sequentially for sake of illustration

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25

If the walker currently occupies square 24 and is told to move down and to the left, the walker’s new location would be square 3. If the walker is in square 5 and is told to move up and to the right, the walker’s new location would be square 21.

- You should record how many times each square is visited and output the result, along with the size of the square, and the number of steps in the walk. Consider the beginning square to be step 1. This means that the total of the numbers for your outputted grid should be n^4 . Here is an example output:

The grid size is 6 by 6.

The number of steps is 1296.

Here is a record of the random walk:

37	38	35	35	29	30
32	40	41	42	32	31
37	40	35	34	35	32
33	35	33	38	39	37
38	34	40	39	37	36
38	35	36	38	38	37

- Your output should be nicely formatted and easily readable.