Date

CISC 205 – OOPS C++

Professor Larry Forman

Telephone: 619.388.3666

E-Mail: [LForman@sdccd.edu](mailto:lforman@sdccd.net)

Mail Box: Room A-8

Office: BT-210-G

**PROGRAM #1PP: POINTER POWER**

**!!! 6 GRADES !!!**

**=== DUE === !NO EXTENSIONS!**

**=============**

**TASKS:**

**0 –** First, read this Task Sheet!

Second, put a check mark by each Task number **and** letter when you complete it.

**1 –** **READ**: **HO#1.5, P#1TT, P#1.TT+, TDB**

**2 –** **TA OBJECTIVES**

* Develop a menu-based C++ program to produce interactive screen output
* Incorporate pointers, dynamic memory allocation and arrays/vectors of pointers
* Maintain effective Michelangelo documentation

**3 – BACKGROUND / SPECIFICATIONS**

We will explore the versatility of pointers through a program that implements a variation of “The Galton Board” (also called a “bean machine” or a “quincunx”). Attributed to the eminent English scientist Sir Francis Galton, the board is a physical device – like a vertical pinball/pachinko/bagatelle machine – for conducting statistical experiments to demonstrate the “central limit theorem”. Balls are dropped from the top and fall down a vertically positioned board with pegs (or nails) in a staggered, (and in this case) triangular and evenly spaced arrangement. When a ball hits a peg, the ball can bounce with a 50-50 chance (in our version) to the left and to the right. As the number of balls that are dropped increases, a surprising pattern emerges as balls fill the empty slots at the base of the board: We see the emergence of the famous “bell curve” of mathematics, formally called the “normal distribution” (or the “Gaussian”, named after the “prince of mathematics” and perhaps greatest mathematician of the 19th century, Carl Friedrich Gauss).

Visit this link to see the Galton Board in action: [www.youtube.com/watch?v=6YDHBFVIvIs](http://www.youtube.com/watch?v=6YDHBFVIvIs)

**STAR (1+): Research the history of the Galton Board and its connection with the normal distribution. Give a PowerPoint presentation highlighting origins, intriguing facts and tidbits about the Galton Board and people involved in its development.**

**STAR (1+): Extend previous STAR by adding “pizzazz” to the PowerPoint presentation with dynamic graphics, transitions, audio and possibly relevant video**

Once again, using a menu-based system, create a program, LarrysPP (no spaces or punctuation), that simulates a variety of Galton Board. Pointers, dynamic memory allocation and arrays/vectors of pointers will be used to implement the simulation. Your program will include each of the following as a separate function with self-descriptive names and appropriate modifications – in your main(), where you deploy ONLY custom-defined functions with any necessary local variables and constants.

**BTW: NO GLOBAL VARIABLES ALLOWED**:

1. The "Welcome" function with your name via a **const** to greet the user.
2. The “Menu” function with appropriate error-handling of “dirty data” from user. Assume replies are NOT case-sensitive so lower- and uppercase values are valid
3. A separate function for EACH menu item: Sign-in, Logo, Go Galton, Play Game Galton, Game History, Exit (etc. for some STARS)
4. “Nest” a "Hit ENTER to Continue" message function in any of the above items to enhance readability. But do NOT deploy your hitEnter function in the main.

**4** – **SCREEN OUTPUT**

Display the following with appropriate blank lines between each section:

Welcome to Larry’s Galton A-Go-Go Program

<<< Hit ENTER to continue >>>

MENU

S – Sign in

L – Display Logo

G – Go Galton

P – Play Game Galton

H – Game History

X – Exit Program

Please enter your selection and hit RETURN: z

Sorry, but “z” is not a valid entry.

Time to try again . . .

**(NOTE: Do complete “error-handling” of “dirty” data)**

MENU

S – Sign in

L – Display Logo

G – Go Galton

P – Play Game Galton

H – Game History

X – Exit Program

Please enter your selection and hit RETURN: s

Thank you for selecting “s” . . .

Time to sign in!

Please sign in with your first name: Albert

Ah, Albert, so nice you have you join us for

a little Galton A-Go-Go . . .

**STAR: Prompt user for full (first and last) name. Burp back full name and separately display just the first name and just the second name.**

**STAR: After user signs in, use the user name in all future prompts, like:**

**UserName, please enter your selection and hit RETURN:**

MENU

S – Sign in

L – Display Logo

G – Go Galton

P – Play Game Galton

H – Game History

X – Exit Program

Please enter your selection and hit RETURN: L Thank you for selecting “L” . . .

Time for Larry’s logo!

(Display your logo from TA #1.2Q)

<<< Hit ENTER to continue >>>

MENU

S – Sign in

L – Display Logo

G – Go Galton

P – Play Game Galton

H – Game History

X – Exit Program

Please enter your selection and hit RETURN: g

Thank you for selecting “g” . . .

Time to have a ball with the Galton Board!

**(Assume we start off with 1000 balls in row#0 and that as the balls “flow” down the rows there is a 50% chance that each ball will either stay in the same column or will move one column to the right. Record the number of balls that have “hit” each peg in each row. Use setw to align the columns.)**

0: 1000

1: 526 474

2: 273 501 226

3: 130 382 370 118

4: 74 233 393 247 53

5: 34 154 317 322 145 28

6: 17 92 263 290 234 94 10

7: 7 50 183 279 253 177 45 6

8: 5 30 114 224 276 221 97 30 3

9: 1 20 77 147 273 255 152 60 13 2

**STAR: Center-align the columns:**

**0: 1000**

**1: 526 474**

**2: 273 501 226**

**Etc.**

**STARS (2): Add a “histogram” after row #9 to visually show the values. Assume each \* represents a count of 20. Plan to take each count in row#9 and “round” to the nearest “20” to determine the number of \*s to display (Hint: Add 5, then “div” by 20 to get the number of \*s needed for each column):**

**9: 1 20 77 147 273 255 152 60 13 2**

**\* \* \* \* \* \* \***

**\* \* \* \* \* \***

**\* \* \* \* \* \***

**\* \* \* \* \***

**\* \* \* \***

**\* \* \* \***

**\* \* \* \***

**\* \* \***

**\* \***

**\* \***

**\* \***

**\* \***

**\* \***

**\***

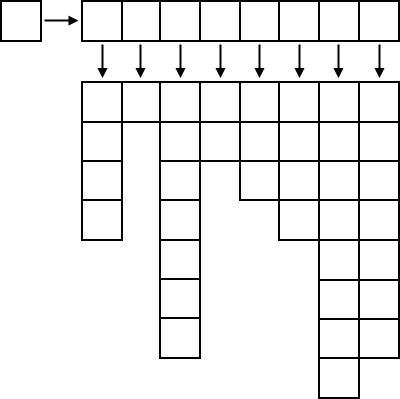
**STARS (2): Alternatively, do the previous STAR with the histogram rising “upward” instead of going “downward”**

**For the following 3 STARS, prompt user by name to select a value. Do “error-checking” via recursion with a “sorry” message for “dirty data” (= numbers not in the valid range):**

**STAR: The value is the number of balls that will be “dropped”. Accept only numbers > 0 and < 10000 (or whatever number you select). And smallest value>0 to crash program is?**

**STAR: The value is the number of rows that will be used. Accept only numbers > 0 and < 100 (or whatever number you select). What is smallest positive value to crash the program?**

**STAR: The value is a “bouncing” probability between >0 and <100, instead of using the 50-50 rule for moving the ball straight down or down to the right.**

**(Assume the Galton Board has 10 rows of pegs with 1 peg in row# 0, 2 pegs in row# 1, … , 10 pegs in row# 9. Use an array**  **of pointers with the first element of the array pointing to an array of length one, the second element of the array pointing to an array of length two, etc. The elements of the array count the number of balls that have “hit” the designated pegs as the balls “flow” from row#0 down to row#9)**

**STAR: Use vectors of pointers, not arrays of pointers, throughout this assignment**

**(Create four separate functions that use arrays of pointers as parameters and/or for return values, as is appropriate:**

1. **Set up and initialize the “ragged” array of pointers that will be used to count how many “balls” have hit each “peg” in each of the 10 rows**
2. **Run the simulation to fill up the values of the array of pointers in the previous function. Namely, count the number of times the balls have hit each peg in each row.**

**Assume if a “ball” is at row, r, and column, c, then it will move to row r+1 and half the time will stay in column c and the half the time will go into column c+1:**

**Column c Column c+1**

**Row r: ball**

**Row r+1: ball OR ball**

1. **Display the results of the simulation (as shown above)**
2. **Delete all the pointers dynamically created)**

<<< Hit ENTER to continue >>>

MENU

S – Sign in

L – Display Logo

G – Go Galton

P – Play Game Galton

H – Game History

X – Exit Program

Please enter your selection and hit RETURN: p

Thank you for selecting “p” . . .

Time to play Game Galton!

0: 1000

1: 526 474

2: 273 501 226

3: 130 382 370 118

4: 74 233 393 247 53

5: 34 154 317 322 145 28

6: 17 92 263 290 234 94 10

7: 7 50 183 279 253 177 45 6

8: 5 30 114 224 276 221 97 30 3

**9: 1 20 77 147 273 255 152 60 13 2**

**Weight 126 14 3.5 1.5 1 1 1.5 3.5 14 126**

**SCORE: 126+ 280+ 269.5+220.5+273+ 255+ 228+ 210+ 182+ 252**

Congratulations, Albert! Your score in Game #1 is 2296

**STAR: Perfectly align the Weight and SCORE values**

**(This menu selection deploys the “Go Galton” function AND then computes a game score as follows based on the “bottom” row, i.e., row #9:**

**Score = Sum of (number count in column #c)\*w, for all columns, c, from #0 through #9, where for:**

**Column #c = 0 1 2 3 4 5 6 7 8 9**

**we have:**

**w = 126 14 3.5 1.5 1 1 1.5 3.5 14 126**

**w represents the “weight” value of each of the 10 “slots” containing the balls. E.g., each ball falling into either column #0 or #9 is worth 126 points; in column #1 or #8 is worth 14 points; #2 or #7 is 3.5; #3 or #6 is 1.5; #4 or #5 is only 1**

**STAR: Figure out a (nearly simple) formula to compute w from c, namely identify a function, f, such that f( c ) = w for the values of c and w above. And, PROVE it!**

**STAR: What does f look like if there are more than 10 rows?**

**STARS (2): Prompt user by name to select the number of rows that will be used. Do “error-checking” using recursion – with a “sorry” message for “dirty data” – and accept only numbers > 0 and < 100 (or whatever number you select). Effectively modify the “weights” for scoring and explain how your scoring system works and why/how you selected it.**

<<< Hit ENTER to continue >>>

MENU

S – Sign in

L – Display Logo

G – Go Galton

P – Play Game Galton

H – Game History

X – Exit Program

Please enter your selection and hit RETURN: H

Thank you for selecting “H” . . .

Time to make history!

So far, Albert, you have played 4 games with the following scores:

Game #1: 2296

Game #2: 2001

Game #3: 2020

Game #4: 2345

**(Use a pointer with dynamic memory allocation to store the Game History Scores)**

**STAR: Create two menu items, one to write the Game History to a disk file and another to read the Game History from the disk. When performing the disk writing and reading, be sure to “echo” to the screen the data being written and read.**

**STAR: Expand the previous STAR to give user the option to “append” data when disk writing.**

**NOTE: In the following 4 STARS, use pointers to do the STAR**

**STAR: Also display a “High Score” listing (using a sorting function you custom-create via any sorting algorithm):**

**From highest to lowest scores: 2345 2296 2020 2001**

**STAR: Expand the previous STAR to include the corresponding Game #:**

**High Score Table:**

**Game #4: 2345**

**Game #1: 2296**

**Game #3: 2020**

**Game #2: 2001**

**STAR: Expand the previous STAR by creating two menu items: one to write the High Score Table to a disk file and another to read the High Score Table from the disk. When performing the disk writing and reading, be sure to “echo” to the screen the data being written and read.**

**STARS (2): Allow multiple users to sign-in and keep separate High Score Tables for each user**

**STARS (2): Allow multiple users to sign-in. In the High Score Table include the user’s name with the high scores**

<<< Hit ENTER to continue >>>

MENU

S – Sign in

L – Display Logo

G – Go Galton

P – Play Game Galton

H – Game History

X – Exit Program

Please enter your selection and hit RETURN: x

Thank you for selecting “x” . . .

Time to x-cape!

Farewell, Albert, and thanks for visiting Larry’s Galton A-Go-Go Program

**Display the current date and time**

**Display your complete ID INFORMATION (nicely formatted), CREDITS, MEDIA and any STARS you did in the following format (BTW – STARS must be fully displayed here to get credit for them):**

**For example:**

**STARS**

1. **Fancy logo**
2. **Early demo**
3. **& 4. PowerPoint presentation**

**about Galton Board**

**TOTAL STARS = 4**

<<< Hit ENTER to continue >>>

**BTW: No “double-dipping” from previous assignments on the STARS**

**STAR: Allow only one sign-in. Alert user it can’t be done if user tries again – however, if you do the “multi-user” STARS above, you can’t do this STAR**

**STAR: Only proceed after sign-in is done, although it’s okay to select exit**

**STAR: Only proceed if sign-in is done first, although it’s okay to select exit**

**STAR: Allow only 3 invalid menu entries after which program “exits”**

**STAR: Expand previous STAR to warn user at 3rd invalid entry that program will exit if the 4th try is invalid.**

**STAR: Instead of re-writing the menu for “Sorry” message, be “ecological “and move the cursor back to the prompt point, but be sure to pause for a moment and then erase the now unnecessary “Sorry” part of the message**

**STAR: Expand previous STAR to be ecological for all appropriate menu selections**

**5** – **TEST** your file early and often -- like every 5 minutes -- How do you eat an apple?

**6** – **MICHELANGELO PROGRAM DOCUMENTATION**

1. First, include via comments your complete **ID INFORMATION, PROGRAM DESCRIPTION, CUSTOM-DEFINED FUNCTION LIST**, **CREDITS** (to those who helped you and whom you helped) and **MEDIA** (descriptions and specific URLs for sounds, graphics, etc. used).
2. **Include "inline credits" to acknowledge specifically where you were helped.**
3. Add comments immediately before each segment of your program to describe "highlights" of coming attractions. Insert at least one blank line before each of these comments. Make all function, variable and const names self-descriptive, clear and fully formed (no abbreviations or secret code names). **Use verbs for function names, like “displayXyz” or “getXyz” or “calculateXyz” or “doXyz” and nouns for constants and variables.**
4. Define a const for your name and use it whenever your name appears
5. Add “banners” for: your prototypes to include description of each function, all constants, variables, start of function definitions, end of function definitions and in the function definitions with a description of each function (you can use the same ones as in your prototypes). See Handout #1.2 for all the details.
6. **FUNCTION PROTOTYPES –** In your function prototype section just AFTER the function “signature”, insert a “mini-banner” with:

**//DESCRIPTION:** Generally describe what process(es) the function performs

**//PRECONDITIONS:** Describe the parameters and assumptions about the

parameters going INTO the function

**//POSTCONDITIONS:** Describe assumptions about the parameters going

OUT OF the function as well as about any return value

**See Handout on Pre- and Post-Conditions for more details**

1. **FUNCTION DEFINITIONS –** Before each custom-defined function definition include a big “banner” containing:

1. **NAME:** The **name** of the function

2. **DESCRIPTION:** A brief, but detailed, **description** of what the function does

(Just copy-and-paste description from Prototype section)

3. **CALLS:** A **list** that identifies the name of each custom-defined function that

the current function calls/uses/deploys

4. **CALLED BY:** A **list** that identifies the name of each function that calls/uses/deploys the current custom-defined function

1. IMPORTANT: Embrace the Michelangelo structure for **every** C++ program.

7 – **DEMO – 6 GRADES**

* **CLASS DEMO OF PROGRAM REQUIREMENTS (3 GRADES)**
* **ONE-ON-ONE PROGRAM DEMO (1 GRADE)**
* **MICHELANGELO DOCUMENTATION (1 GRADE)**
* **MANAGER’S REPORT (1 GRADE)**

**8 –** **HARDCOPY – MANAGER’S REPORT (**[**www.projectcartoon.com/cartoon/2**](http://www.projectcartoon.com/cartoon/2))

**Manager’s Status Report** – Word process at least one good-sized, **titled** paragraph for each answer, **along with relevant and effective clip art for each paragraph:**

1. **HURDLES**: Identify any relative hurdles, challenges and frustrations you encountered in C++ on this PROGRAM and describe how you dealt with them.
2. **PRIDE**: Identify a part of the PROGRAM that you were proud of completing, explain what you learned from it and why you were proud
3. **HELPED OTHERS**: Describe how you helped others on this PROGRAM. Give details. If you did not help anyone, then have someone demo their project and describe what you liked about it and your suggestions for enhancing it. Give the name(s) of the student(s) for either situation.
4. **HELPED BY OTHERS:** Describe how you were helped by others on this PROGRAM. Give details. If you were not helped by anyone, then have someone demo your project and describe what he/she liked about it and his/her suggestions for enhancing it. Give the name(s) of the student(s) for either situation
5. **STARS:** Briefly identify in a numbered list each **STAR** you did **and** display the **total number of** **STARS**
6. Include your **TASK TIMESHEET** with the accurate, completed information

**9** – **STARS**

1. Work in a 2-3 person team to write and demo one program together that contains all the required information. Additionally, include the names of all the team members (as separate constants) in each function. Add additional and different Galton Boards, one for each teammate. In the ID INFO, individually show hours and difficulty for each person. **Also, in each function, document the names of the contributors and the percentage each one contributed to the function (1 STAR per teammate).**
2. In a NEW engaging way, use >=4 windows.h colors throughout
3. In function Go Galton, also display the 10 rows using \333 in one color to replace any even numbers and another contrasting color to replace any odd numbers.
4. Effectively implement a pointer to a const item
5. Effectively implement a const pointer to a non-const item
6. Effectively implement a const pointer to a const item
7. Effectively implement a pointer to a pointer (NOTE: This might be useful to handle call-by-reference types of pointer parameters OR to act like a two-dimensional array vis a vis an array of an array to represent a 2D-array)
8. Effectively implement a “smart” pointer
9. **STARS (2)** Effectively implement a pointer to a function (PS: This is handy for menu-processing operations)
10. Implement this assignment by creating and using a custom header file to hold all your function prototypes and definitions. Include your Michelangelo documentation with ID INFO, etc. in the header file
11. Add extra pizzazz and briefly explain what you did:
12. Effectively use three new "Advanced" features (= not yet introduced). List them:
13. Demo before the due-date (N.B.: You still can do more STARS on due-date)

**Expect problems and eat them for breakfast.**

**Alfred A. Montapert**

**The secret of getting ahead is getting started.**

**Mark Twain**

**Start where you are. Use what you have. Do what you can.**

**Arthur Ashe**



**(A “bagatelle” pinball device)**