College of Engineering, Construction and Living Sciences

Bachelor of Information Technology

IN628: Programming 4

Level 6, Credits 15

In Class Checkpoint 02: Component Review

Assessment Table

Assessment Activity	Weighting	Learning Outcomes	Assessment Grading Scheme	Completion Requirements
In Class Checkpoints	15%	1, 2, 3	CRA	Cumulative
Roguelike	45%	1, 2, 3	CRA	Cumulative
Programming Language Exploration	25%	1, 2, 3	CRA	Cumulative
Theory Exam	15%	3, 4, 5	CRA	Cumulative

Conditions of Assessment

This assessment will need to be completed by Friday, 12 June 2020.

Pass Criteria

This assessment is criterion-referenced with a cumulative pass mark of 50%.

Submission Details

You must submit your program files via **GitHub Classroom**. Here is the link to the repository you will be using for your submission – https://classroom.github.com/a/DigbcYgy. For ease of marking, please submit the marking sheet with your name & student id number via **Microsoft Teams** under the **Assignments** tab.

Authenticity

All parts of your submitted assessment must be completely your work and any references must be cited appropriately.

Policy on Submissions, Extensions, Resubmissions & Resits

The school's process concerning **Submissions, Extensions, Resubmissions and Resits** complies with Otago Polytechnic policies. Students can view policies on the Otago Polytechnic website located at https://www.op.ac.nz/about-us/governance-and-management/policies.

Extensions

Please familiarise yourself with the assessment due dates. If you need an extension, please contact your lecturer before the due date. If you require more than a week's extension, a medical certificate or support letter from your manager may be needed.

Resubmissions

Students may be requested to resubmit an assessment following a rework of part/s of the original assessment. Resubmissions are completed within a short time frame (usually no more than 5 working days) and usually must be completed within the timing of the course to which the assessment relates. Resubmissions will be available to students who have made a genuine attempt at the first assessment opportunity. The maximum grade awarded for resubmission will be C-.

Learning Outcomes

At the successful completion of this course, students will be able to:

1. Program effectively in an industrially relevant programming language.
2. Implement a wide range of intermediate data structures and algorithms to act as modules of larger programs.
3. Use an appropriate integrated development environment to create robust applications.
4. Demonstrate sound programming and software engineering practices independent of the environment or tools used.
5. Explain the theoretical issues surrounding programming language design and development.

Assessment Overview

In this practical, you will complete a series of tasks covering today's lecture. This practical is worth 0.5% of the final mark for the Programming 4 course.

Task 1

Create a Form with a Button. Each time the Button is clicked, increase the Button's width by 10 pixels. When the right edge of the Button's reaches the edge of the form, change the text of the button to "Too Big".

Task 2

Create a Form with a Button & TextBox. When the Button is clicked, its text changes to equal the contents of the TextBox.

Task 3

Create a Form with a Button & ListBox. Each time the Button is clicked, add a new line to the ListBox. The line should read "This is line n", where n is the line number.

Task 4

Create a Form with a Button. When the Button is clicked, its text toggles between "On" and "Off".

Task 5

Create a Form with a Button, TextBox & PictureBox. Using the Property Inspector, set the BackColor of the PictureBox to something other than grey, so that it is clearly visible. When the Button is clicked, the picture box moves n steps to the right, where n is the number entered in the TextBox. Each step should be 10 pixels. After each step, the program should pause for 100 milliseconds (otherwise the PictureBox moves so fast you can't see it). To pause for n milliseconds, use the command System::Threading::Thread::Sleep(n).

Task 6

In this problem, you will dynamically change the image contents of a PictureBox control. To do this, you modify the PictureBox− >Image property. You set it to an Image object, which is created by calling Image::FromFile("filename"). For example, you can display the picture file chicken.jpg as follows:

- pictureBox1− >Image = Image::FromFile("chicken.jpg");

You have been provided a set of 9 images called 02-dragon-1.bmp, 02-dragon-2.bmp, etc. Create a Form with a Button & a . When the Button is clicked use technique described above to load the PictureBox with each of the dragon pictures in order from 1 to 9 to produce a simple animation. Some technical points:

1. This problem can be solved by brute force, or more elegantly by using a for loop & the string concatenation operator "+". Strive for elegance whenever possible.
2. Use relative paths for the images; never hard-code a drive into your file paths. Relative paths in C++ CLI start from the project folder (i.e. where the Form's .h & .cpp files are found), not the Solution folder or the Debug folder.
3. Use Sleep(n) as above to control the timing, & use Application::DoEvents() to refresh the screen after sleeping.
4. Set the PictureBox's SizeMode property to allow it to dynamically size to its contents. This lets you change the image set without having to recompile.

Task 7

C++ CLI applications do not automatically provide a canvas. Instead, you must create a Graphics object to paint on. To create such an object, use the following code:

```
Graphics^ mainCanvas = CreateGraphics();
```

You must be using namespace System::Drawing to have access to the Graphics class.
Note that only some C++ CLI classes expose the CreateGraphics() method. The Form is one of them. So, you can create a Graphics object as shown above in any Form method (e.g. in the Form_Load or in any button click handler).

After you have created a Graphics object you have many methods for drawing on it. You can draw shapes, images, text, lines, etc. Today we will use only one of the many Graphics methods: FillEllipse.

The syntax for FillEllipse is: FillEllipse(Brush, x, y, width, height)

The parameters x & y are the coordinates of the upper left corner of the ellipse, & the parameters width & height are its horizontal & vertical dimensions. The Brush is another member of the System::Drawing namespace. Brushes determine the colour inside a drawn shape. (There is another object Pen that determines the colour of the outline of a shape.) Before calling FillEllipse, you must create a Brush. The full syntax to create a SolidBrush (there are other varieties we will see during the term) is:

```
Brush^ greenBrush = gcnew SolidBrush(Color::Green);
```

Given the above discussion, what do you think happens when the button shown below is clicked? Test it & see.

```
private: System::Void button7_Click(System::Object^  sender,
System::EventArgs^  e)
{
      Graphics^ mainCanvas = CreateGraphics();
      Brush^ greenBrush = gcnew SolidBrush(Color::Green);
      mainCanvas->FillEllipse(greenBrush, 100,100,10, 10);
}
```

Using the techniques just described, make the following: Create a Form with a Button & Timer. At each Timer tick, draw an ellipse at a random location on the screen. The ellipse should be of random width between 0 & 100 pixels, & random height between 0 & 100 pixels. Arrange to have approximately equal proportions of red, green, blue & yellow ellipses. Include a Button that turns the Timer on & off.

Submission

- Create a new branch named 02-checkpoint within your practicals GitHub repository
- Create a new pull request and assign Grayson-Orr to review your submission
- Deadline: Wednesday, 4 March at 5pm

Note: Please don't merge your own pull request.