Introduction to C++

|  |  |
| --- | --- |
| **Assessment Task Number:** Part 1 – Activities on Arrays | |
| **Unit Code(s):** | **Unit Title(s):** |
| ICTPRG443 | Apply intermediate programming skills in different languages |
| ICTICT449 | Use version control systems in development environments |
| **Instructions to Learners:** | |

Download the provided Visual Studio solution *IntroToCpp\_Arrays*

Complete the following C++ programming problems on arrays.

**Problem 1: Looping**

Open the Project AIE\_01 and AIE\_01.cpp

You will see we have defined an array for you

|  |
| --- |
| const int NUM\_NUMBERS = 5;  int numbers[NUM\_NUMBERS] = { 10, 20, 30, 40, 50 }; |

Part 1: Print each value of the array in order.

Expected console Output:

|  |
| --- |
| 10, 20, 30, 40, 50, |

Part 2: Print each value of the array in reverse order.

Expected Output

|  |
| --- |
| 50, 40, 30, 20, 10, |

**Problem 2: Passing arrays to functions**

Open the Project **AIE\_02** and **AIE\_02.cpp**You will see we have defined the following functions

|  |
| --- |
| int MinValue(const int\* arr, int count);  int MaxValue(const int\* arr, int count);  int CountOccurencesOfValue(const int\* arr, int count, int search); |

An array in C++ is simply a pointer to the first value within an array, when we pass an array into a function, we also need to pass the number of values the array contains. So, the above methods also include a count parameter.

These methods should not modify the passed in array, therefore we have defined the array as const so the compiler will prevent us from modifying the array we pass into the method.

We have defined an array in the main method.

|  |
| --- |
| const int NUM\_NUMBERS = 10;  int numbers[NUM\_NUMBERS] = { 10, 20, 30, 40, 50, 10, 20, 30, 40, 50 }; |

In **AIE\_02.cpp**, You will see that we have invoked the above methods and expect these methods to return an expected value.

Your task is to implement the methods, so that the expected values are correctly returned. When you have implemented these methods, you should see the following output:

|  |
| --- |
| Expected: 10 - Got: 10  Expected: 50 - Got: 50  Expected: 2 - Got: 2  Expected: 0 - Got: 0 |

**Problem 3: Binary Search**

Open the Project **AIE\_03** and **AIE\_03.cpp**You will see we have defined an array that is sorted in ascending order.

|  |
| --- |
| const int NUM\_ITEMS = 8;  int arr1[NUM\_ITEMS] = {1, 3, 5, 7, 9, 11, 13, 15}; |

We have also defined a “BinarySearch” function that you are required to implement.

|  |
| --- |
| const int\* BinarySearch(const int\* arr, int count, int searchVal); |

This method should:

* Return a pointer the found value if it exists, or
* Return nullptr if the value does not exist.

We have provided a few tests to verify if your implementation works as expected, below we are searching for the values 9, 1, 15, 0 and 16 for the array defined above.

|  |
| --- |
| TestResult( BinarySearch(arr1, NUM\_ITEMS, 9), &arr1[4] );  TestResult( BinarySearch(arr1, NUM\_ITEMS, 1), &arr1[0] );  TestResult( BinarySearch(arr1, NUM\_ITEMS, 15), &arr1[7] );  TestResult( BinarySearch(arr1, NUM\_ITEMS, 0), nullptr );  TestResult( BinarySearch(arr1, NUM\_ITEMS, 16), nullptr ); |

Expected output:

|  |
| --- |
| Pass: result(9)  Pass: result(1)  Pass: result(15)  Pass: result(nullptr)  Pass: result(nullptr) |

**Problem 4: Sort Array**

Open the Project **AIE\_04** and **AIE\_04.cpp**You will see we have defined an array that is not sorted.

|  |
| --- |
| const int NUM\_ITEMS = 8;  int numbers[NUM\_ITEMS] = { 13, 1, 7, 5, 11, 9, 15, 3 }; |

You will see we have defined a SortArray function that you are to implement.

|  |
| --- |
| void SortArray(int\* arr, int count); |

This method should modify the passed in array, so that by the time the method returns, the array has been re-ordered so that it is sorted in ascending order.

**Problem 5: 2D Array Indexing**

Open the Project **AIE\_05** and **AIE\_05.cpp**You will see we have defined a 2D array of strings, here we have defined 3 rows and 5 columns.

|  |
| --- |
| const int ROWS = 3;  const int COLS = 5;  const char\* grid2D[ROWS][COLS] =  {  { "a0", "a1", "a2", "a3", "a4" },  { "b0", "b1", "b2", "b3", "b4" },  { "c0", "c1", "c2", "c3", "c4" },  }; |

We have also defined a second 1D array with the same values.

|  |
| --- |
| const char\* grid1D[ROWS \* COLS] =  {  "a0", "a1", "a2", "a3", "a4",  "b0", "b1", "b2", "b3", "b4",  "c0", "c1", "c2", "c3", "c4"  }; |

For this problem, we have 3 parts for you to complete. These will require you to loop through grid2D.

Part 1:

Print all values in grid2D. You will need 2 loops to iterate over the rows and columns.

Expected output:

|  |
| --- |
| a0, a1, a2, a3, a4,  b0, b1, b2, b3, b4,  c0, c1, c2, c3, c4, |

Part 2:

Unlike Part 1, we have defined a single loop. You are required to convert the loop index value into a row and column index value. To achieve this, you must know how many columns are defined for the array. The Formula is:

col index = index modulus num columns

row index = index divided by num columns

Expected output:

|  |
| --- |
| a0, a1, a2, a3, a4, b0, b1, b2, b3, b4, c0, c1, c2, c3, c4, |

Part 3:

For Task 3, we have defined a single 1D array called grid1D. we have also provided a nested loop that is used to iterate over rows and columns. Our task is to convert the row and column index into a single index.

Formula:

Index = rowIndex \* NUM\_COLS + colIndex

Expected output:

|  |
| --- |
| a0, a1, a2, a3, a4,  b0, b1, b2, b3, b4,  c0, c1, c2, c3, c4, |

**Problem 6: Simple Tile Map**

Open the Project **AIE\_06**, **Application.h** and **Application.cpp.**This project uses raylib to render a tile map to the screen, we have define a 1D array that will be used in a 2D context.

You’ll see we have defined the following member variables in Application.h

|  |
| --- |
| static const int ROWS = 40;  static const int COLS = 40;  int m\_tiles[ROWS \* COLS];  int m\_tileWidth = 20;  int m\_tileHeight = 20; |

Part 1:

Initialise all values in the m\_tiles array to a random value between 0 and 5 exclusive – these values will represent a colour for the tile.

* 0 = WHITE
* 1 = RED
* 2 = GREEN
* 3 = BLUE
* 4 = YELLOW
* Any other value = BLACK

Open Application.cpp navigate to the Load() method.

Iterate over all values in the array and assign them a random value between 0 and 5 exclusives.

|  |
| --- |
| void Application::Load()  {  // Task1:  // Initialise all values in m\_tiles array to a random  // value between 0 and 5 exclusive;  // -----------------------------------------------------    // write your code here  // -----------------------------------------------------  } |

Part 2:

We are going to use raylib to draw a coloured rectangle to the screen, here you will need to use a nested loop to iterate over rows and columns.

Open Application.cpp navigate to the Draw()

* Use the row and column value to calculate the position.
  + xPos = colId \* tileWidth
  + yPos = rowId \* tileHeight
* Use the tile value to find the colour required to render the tile.  
  To get the tile value – you will need to convert the row and col to an array index using the following formula:
  + index = rowId \* COLS + colId

Once you finish task 2, you should be able to run the project. You should see a window with random colours rendered.

|  |
| --- |
| void Application::Draw()  {  BeginDrawing();  ClearBackground(RAYWHITE);  // Task2:  // use a nested loop to iterate over rows and columns  // Use raylib's DrawRect method to draw each tile in the array.  // use the row and col index multipled by m\_tileHeight/m\_tileWidth  // to calculate the x and y position for each rectangle.  //  // change the color of the rect drawn based on the value of the tile.  // We have created a helper function you can use "GetTileColor"  // --------------------------------------------------------------------  // write your code here  float xPos = 0;  float yPos = 0;  Color color = GetTileColor(1); // pass in the tilevalue  DrawRectangle(xPos, yPos, m\_tileWidth, m\_tileHeight, color);  // --------------------------------------------------------------------  EndDrawing();  } |

Qr code

Description automatically generated

Part 3:

Now that we have rendered our tiles to the screen, we will modify values within the array when we click with the mouse.

* We need to convert the mouse position to a row / column. We can do this by dividing the mouse coordinates by the tileWidth and tileHeight.
* Once we have the row and column, we can calculate the array index through the following formula: index = rowId \* COLS + colId

Open Application.cpp navigate to the Update() method.

Change the code so that we calculate the rowIndex, colIndex and tileIndex values.

|  |
| --- |
| void Application::Update(float deltaTime)  {  if (IsMouseButtonPressed(MOUSE\_LEFT\_BUTTON))  {  Vector2 mousePos = GetMousePosition();  // Task 3:  // TODO: Calculate row and col index based on the mouse positon  int rowIndex = 0;  int colIndex = 0;  // TODO: calculate the index of the tile clicked on based on the  // row/col index  int tileIndex = 0;  m\_tiles[tileIndex] += 1;  if (m\_tiles[tileIndex] >= 5)  m\_tiles[tileIndex] = 0;  }  } |

When complete, you should be able click a tile in the window, and have it cycle through to the next colour.

|  |  |  |
| --- | --- | --- |
| **Task** | | **Evidence Criteria** |
| 1. | Looping | Write a C++ program that correctly generates the expected output for the following problems:  Part 1: Print values in order.  Part 2: Print values in reverse order. |
| 2. | Passing arrays to functions | Write a C++ program that meets the problem brief. |
| 3. | Binary Search | Write a C++ program that meets the problem brief. |
| 4. | Sort Array | Write a C++ program that meets the problem brief. |
| 5. | 2D Array Indexing | Write a C++ program that correctly generates the expected output for part 1-3 of the problem brief. |
| 6. | Simple Tile Map | Write a C++ program that correctly generates the expected output for the following problems:  Part 1: Initialise array.  Part 2: Draw Tile Map.  Part 3: Modify Tile Map on Mouse Click. |
| **Submission Requirements:** | | |
| You will need to submit the following:   * A Release build of each application that can execute as a stand-alone program * Your complete Visual Studio project   Be sure to remove any temporary build folders (i.e., the Debug and Release folders). Only project files, source code files, and any resource files used should be included in your submission.  Package all files in a single compressed archive file (.zip, .7z, or .rar) | | |