

SPL 5: Graphical Priority Queue
Out: Thursday, March 9th; Due: Thursday, March 16th

I. Objectives

Primary Objective:

Improve your understanding of pointers through the implementation of a graphical priority queue.

Secondary Objectives:

- Practice file parsing – File I/O. Once data is written to a file, how is it retrieved and understood? If a program reads “2.3+2.5” from a file, it should determine that the characters ‘2’, ‘.’, and ‘3’ represent the floating point value 2.3 and that ‘+’ represents addition. One way a computer can identify and interpret the useful pieces of information is by parsing.
- Practice code reuse – An important real life skill involving creating and reusing modular, reliable, pieces of code. A certain popular operating system consists of 50 million+ lines of code. Can you imagine how long it would take and how many more bugs would exist if a certain 50-million-lines-of-code operating system had to be rewritten from scratch each time?
- Introduce a basic Graphics Library – this will add a new dimension to interfacing with the user outside of the text window that you have been using.

II. Files

Downloads - Please go to the Unified C&P Website and download the two files under “Systems Problem Support” (If you haven’t done it last term, you may also need UnifiedC&PInstallation):

“UnifiedC&PInstallation-Graphics.exe”

“C&P_SPL_5.zip”

Install “UnifiedC&PInstallation-Graphics.exe”. The folder created by unzipping “C&P_SPL_5.zip” is probably where you will want to develop your code.

Included files:

“ada_graphics.ads & ada_graphics.adb”

“double_buffer.ads & double_buffer.adb”

“priority_queue.ads & priority_queue.adb”

“priority_queue_graphics.ads & priority_queue_graphics.adb”

“test_queue.adb”

“Input.txt”

“input_generator.exe”

“test_queue.exe”

“test_ada_graphics.exe”

“gnat.ago”

Files you will *have* to modify (you *are* allowed to modify any of them):

“priority_queue.adb”

“test_queue.adb”

III. Overview

This System Problem is a departure from the C&P work you have done in the past. While you are typically asked to create the programs, this one will ask you to modify and add sections to existing code. For this SPL you will be pulling together many of the C&P lessons. Your program will parse/read a file for information, store this information in a priority queue, and then dynamically draw how the pointers are being manipulated in the queue to make this all possible. The final result should be similar to “test_queue.exe”.

First, what is a priority queue? As you should know, a queue is a method of dynamically storing information using pointers. By definition, a queue is supposed to be FIFO (First In, First Out). A priority queue is a little different. You give the element a value when it is Enqueued. Upon Dequeue, the element in the queue with the smallest or largest associated value is returned. Our priority queue will return the element with **minimum** associated value current in the queue.

In the two files you are expected to modify: “priority_queue.adb” and “test_queue.adb” you will find comments “--Fill in here as needed.” These are locations that *may* need code. Since one of the skills tested in this SPL is code reuse, **read the comments in the given code or associated .ads file!** You will find that most of the .adb files are poorly commented, while the .ads files are richly commented. This is typical of most programming languages, the specification file or header file is has more explanations.

The procedures in “priority_queue.adb” that require modification are:

1. MakeEmpty
2. Enqueue – *Sort* the element into the queue using whatever sort you like.
3. Dequeue – *No Sorting*, just return the head element

A few words on “priority_queue.adb”

The record types that will form the nodes/elements of the queue have already been defined and can be found in “priority_queue.ads.” The general structure of the priority queue is depicted in Figure 1. The Queue Type holds the basic information you need to know about the Queue, the size and where its Head and Tail are. The Nodes form the backbone of the Queue. Each Node has a pointer linking it to the next Node (which should have a greater Value). Each Node also has a data Element which gives the (X, Y) coordinates at which each Node should be drawn in the window. The Tail pointer points to the last Node in the Queue, which has no pointer to any other Node.

ToDo:

- Modify these procedures so that they act as described in “priority_queue.ads”. (Hint: Look at the “Linked_List” package on the Unified C&P Website Extra Material, Lecture C5 for implementation ideas).

- Interleave into your code, procedures that graphically draw (and erase) the pointers as they are being manipulated. Look at the helper functions in “priority_queue” and graphical functions in “priority_queue_graphics” to make drawing much easier. Sections of your code that have been modified to have graphics will typically look like:

```
EraseSomething(...) ;-- This is the added graphics  
Pointer1 := Pointer2 -- This is the important part  
DrawSomething(...) ; -- This is the added graphics
```

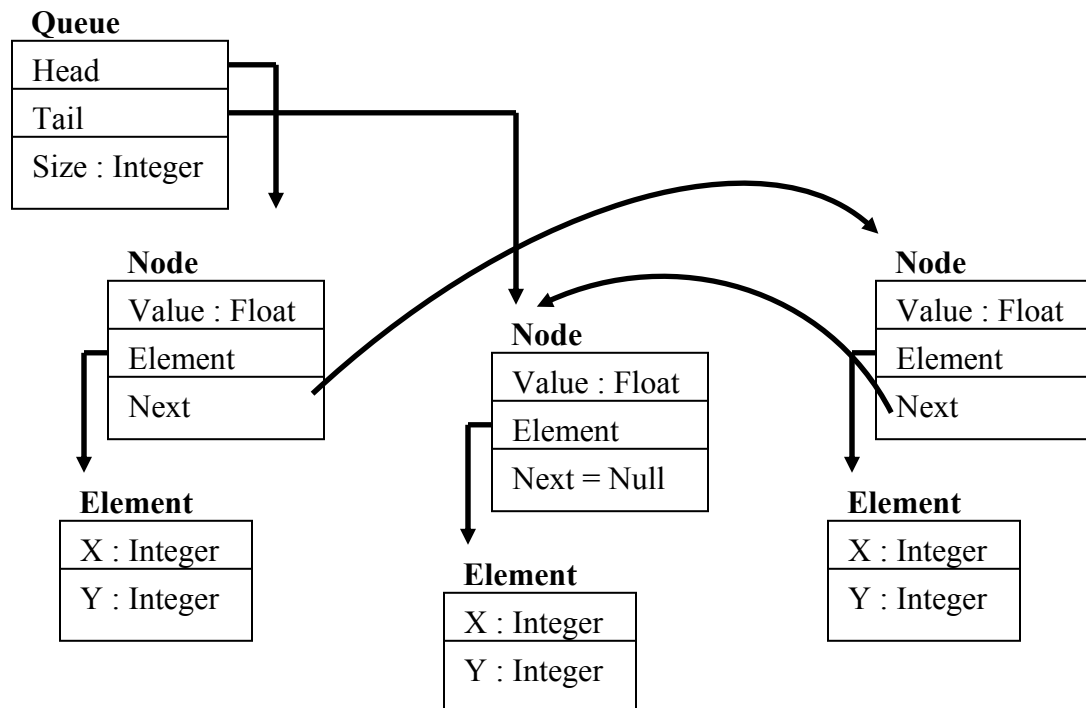


Figure 1: Structure of Priority Queue. Nodes should be linked from one to the Next starting with the minimum Value first.

The procedures in “test_queue.adb” that require modification are:

1. Body of Test_Priority_Queue
ToDo: You will need to look at “Test_Ada_Graphics.adb” to get an idea of what is missing. While most of the initialization needed to create a window is there, a few lines of file I/O and thread registering need to be done.
2. Body of Draw_Queue
ToDo: Add the code that allows this procedure to read from the file, create the element, and call Enqueue to put the element in the priority queue. The file you will be reading from is “Input.txt.” Note that Draw_Queue, when properly registered, will be called continuously in a loop this has an important impact on the code for File I/O.

A few words on “test_queue.adb”

The Queue that you will be operating on is called “Q” and has already been defined in the definitions portion of “priority_queue.adb.” The File being read from is “Input.txt” and is formatted as shown in Figure 2. The numbers are clustered in groups of 3, a float number used as the Node’s Value, and an X and Y coordinate for the Node, respectively.

Note: You will *ONLY* need to call procedure Enqueue from Draw_Queue. This is why it is important that the Enqueue procedure is the one that *sorts* the element into the Priority Queue.

Input.txt

```
20.3650012
  200
  400
91.6577965
  600
  400
36.2652487
  350
  400
```

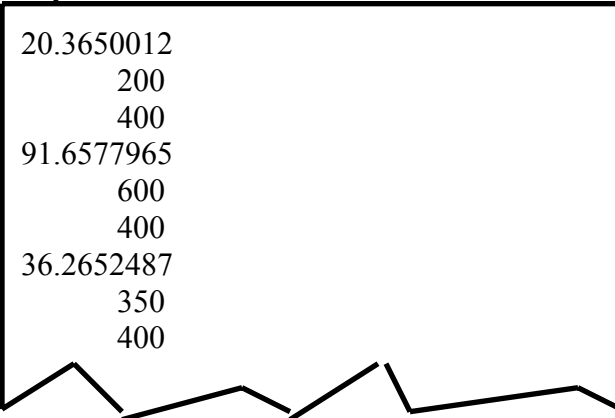


Figure 2: “Input.txt” can be randomly generated by “input_generator.exe” and has groups of 3 numbers.

IV. How To Start

1. Install “UnifiedC&PInstallation-Graphics.exe”
2. Unzip “C&P_SPL_5.zip” You will probably want to develop your code in the folder which unzips.
3. Double Click on “test_queue.adb”. This is the main procedure/entry point for your program. In AdaGide go to “Tools->GNAT Options in Current Directory”. You should see:
Compiler Options: -IC:\GtkAda-2.4.0\include\gtkada
Gnatmake: -LC:\GtkAda-2.4.0\include\gtkada
If you do NOT see this or you installed “UnifiedC&PInstallation-Graphics.exe” in a directory other than “C:” please modify these lines accordingly.
4. Run “test_queue.exe.” This is what your finished product should look like.

Green Pointer – Pointer to the Head of the Queue
Yellow Pointer – Pointer to the Tail of the Queue
Orange Pointer – Pointer to the Newly Created Node/Element
Red Pointer – Pointer using for sorting
Purple Pointer – Pointer used for sorting
Blue Circles – Graphical representation of a Node/Element of the Queue
Green Links – The pointer from one Element to the Next in the Queue.

Note that every time a pointer is given a new value, the old pointer is erased and the new one drawn. (The .exe has been known to not display correctly on some machines)

5. Compare “`test_ada_graphics.adb`” and “`test_queue.adb`.” What are the similarities and differences? This should be enough to fill in the majority of the `Test_Priority_Queue` body.
6. Double click on “`input_generator.exe`” to create a new “`Input.txt`” file. Open this file and get an idea of its layout. You will see numbers in groups of 3, a floating point number, followed by two integers. Each element/node of the priority queue will need to store the float number and use it as the value by which to sort. The remaining two numbers are the X and Y coordinates at which the graphical representation of that element should be drawn.
7. Look at the Enqueue procedure in “`priority_queue.ads`,” what parameters does it require?
8. Look over your notes on File I/O. Remember that manipulating a file has three phases: open, read or write, and close. How do you open a file? How do you close a file? It is one thing to get text from a file, but how do you get numbers from a file (Hint: `Ada.Float_Text_IO`)? How do you know when there is nothing left in a file to be read?
9. We know that code is executed one line at a time. Where in the body of `Test_Priority_Queue` is procedure `Draw_Queue` called? You should now know enough to write the code that parses `Input.txt`.
10. Write the rest of the `Draw_Queue` procedure. Make sure you call `Priority_Queue_Enqueue`.
11. Look over “`Linked_List.ads`” and `.adb`. Do you understand how it works? Code the `Priority_Queue` procedures. Do NOT worry about the graphical part. Print text to the output to help you debug what your code is doing!
12. Copy ALL of your code over to a new folder for safe keeping!
13. Look over the ‘Helper Functions’ in “`priority_queue.ads`” and procedures in “`priority_queue_graphics.ads`.” Add the appropriate draw and erase functions to graphically draw the pointer manipulations.

V. Grading Scheme (Out of 100)

- 10 - Complete Window Creation in "Test_Priority_Queue."
- 10 - File Parsing in "Test_Priority_Queue" and "Draw_Queue."
- 10 - Correct pointer manipulation in "MakeEmpty" procedure
- 25 - Correct pointer manipulation in "Enqueue" procedure
- 15 - Correct pointer manipulation in "Dequeue" procedure
- 5 - Working Graphics for "MakeEmpty"
- 10 - Working Graphics for "Enqueue"
- 5 - Working Graphics for "Dequeue"
- 10 - Paragraph or Pseudo Code on how the pointers are manipulated by Enqueue and how your graphics window relates (what are all the colorful pointers doing)?

VI. Turn In Instructions

1. Please zip and turn in **ALL** the **.ads** and **.adb** files you used in developing this assignment via the on-line submission system (At least make sure "priority_queue.adb" and "test_queue.adb" are included).
2. Name the zip file: "2006SPL5_LastName_FirstName.zip"
3. Go to the web site to submit the file.
4. **Submit the paragraph you wrote in hardcopy.**

Reminder:

- READ THE COMMENTS!
- "Linked_List.ads" and "Test_Ada_Graphics.adb" are good resources.
- Remember to check for a null pointer before using it

Assistance:

Optional "Computer" Office Hours are tentatively scheduled for Monday, March 13th and Tuesday, March 14th from 4-6pm in the Computer Lab. Two TAs will be available during those time slots to help. Please read your e-mail as additional hours may be added.