COMP 2404/2004 -- Assignment #3

The Tortoise Chronicles: Locked Up

Due: Friday, March 16, at 12:00 PM (noon)

Background

Fortune has smiled upon our heroes, the Tortoise and the Hare. Not only did they emerge victorious from their battle with the Dread Pirate Harold's gang of miscreants, but they managed to capture the pirate ship and its entire crew. Curiously, neither the Dread Pirate Harold, nor his equally terrifying Dread First Mate Timmy, were found on board! However, never keen on borrowing trouble, our dynamic duo is now turning its attention to the problem of what to do with their captive pirate gang. King Humpernickel is planning to come aboard the pirate ship tomorrow to survey his new bounty, and all the pirates must be safely stowed in the ship's brig before then. Unfortunately, this presents a number of problems.

There are actually three different kinds of pirates: dwarf orcs (also known as "dorcs"), vicious bear orcs ("borcs"), and tasty pig orcs ("porcs"). Each type of pirate is of a different size, and so must occupy a certain amount of space inside a cell in the brig. Because there are very few of these prisoner cells, and each one is quite small, the Tortoise and the Hare are having difficulty figuring out how to cram the entire pirate crew into the brig. Not only that, but there are also restrictions! Borcs are known to have a weakness for porc meat. Housing borcs and porcs in the same cell is guaranteed to result in a bloodbath and so must be avoided.

Your assignment is to help the Tortoise in coding a simulation of how dorcs, borcs and porcs can be assigned to the cells in the pirate ship's brig. That way, our heroes can ensure that their captives are alive and well for King Humpernickel's visit.

Learning Objectives

In completing this assignment, you will:

- design a larger C++ program using:
  - basic software engineering principles, including encapsulation, extensibility and reusability
- write a C++ program on Linux, featuring:
  - dynamically allocated objects
  - basic inheritance
  - atomic classes and container classes, both supporting a variety of overloaded operators
  - cascading member functions

Submission

You will submit on WebCT, before the due date and time, the following:

- a UML class diagram (as a PDF file) that corresponds to your program design
- a C++ program that meets all the functional and non-functional requirements described in the next pages; all the program source code must be archived together into one `tar` file, including the corresponding Makefile and a readme file describing how the user can compile, execute and operate your program

Program Requirements:

➢ Functional requirements:

- Your program must simulate a brig (basically a prison), containing several cells, each containing a number of pirates. This must be implemented with:
  - a Brig class containing:
    - a static integer indicating the id number to be assigned to the next pirate admitted to a cell
    - an array of a maximum of ten cells comprising the brig
    - the default amount of space in each cell of the brig
    - a default constructor that takes the default amount of space in each cell
    - a copy constructor
  - a Cell class containing:
    - the unique cell number
    - a linked list, implemented as seen in the course notes, of *pointers* to Pirate objects, representing the list of pirates contained in the cell
    - an integer keeping track of the amount of space remaining unoccupied in the cell
    - a default constructor that takes the unique cell number and the amount of space contained in the cell
  - a Pirate class containing:
    - the cell number that the pirate is occupying; before the pirate is assigned to a cell in the brig, this number is zero
    - the amount of space in a cell that this pirate requires
    - a unique prisoner id, assigned when the pirate is assigned to a cell in the brig
    - a string indicating the type of pirate, either "Dorc", "Borc" or "Porc"
    - a default constructor
    - a constructor that takes the required amount of space and the type of pirate
  - a Dorc class, a Borc class and a Porc class, each inheriting from the Pirate class and containing **only** a default constructor with default values
    - a dorc occupies 4 units of space in a cell
    - a borc takes up 5 units of space
    - a porc occupies 2 units of space
  - a List class, implementing a linked list of pointers to Pirate objects, containing:
    - a default constructor
    - a constructor that takes a reference to a Pirate object
    - a copy constructor
    - a destructor

- Your classes must implement the following overloaded operators:
  - the Brig class:

Operator	*Parameter*	*Functionality*
`<<`		Print out the `Brig` information
`+=`	`Pirate& pirate`	Add `pirate` to the first cell in the cell array that has enough space and can handle the `pirate` (i.e. you can't mix Borcs and Porcs); return the resulting `Brig&`
`+=`	`const List& pirates`	Add each Pirate element of `pirates` to the first cell in the cell array that has enough space and can handle the `pirate` (i.e. you can't mix Borcs and Porcs); return the resulting `Brig&`
`-=`	`Pirate& pirate`	Remove `pirate` from its cell in the cell array; return the resulting `Brig&`
`-=`	`const List& pirates`	Remove every Pirate element of `pirates` from its cell in the cell array; return the resulting `Brig&`
`[]`	`int cellNumber`	Return the `List` of pirates contained in the cell that is at position `cellNumber` within the cell array

  - the Cell class:

Operator	*Parameter*	*Functionality*
`<<`		Print out the `Cell` information
`+=`	`Pirate& pirate`	Add `pirate` to the list of pirates, and return the resulting `Cell&`
`-=`	`Pirate& pirate`	Remove every occurrence of `pirate` from the list of pirates, and return the resulting `Cell&`

  - the List class:

Operator	*Parameter*	*Functionality*
`=`	`const List& list`	Assign the contents of `list` to `this`, and return the resulting `List&`
`+`	`Pirate& data`	Return `List` that is the union of `this` and `data`
`+=`	`Pirate& data`	Add `data` to `this` and return the resulting `List&`
`-`	`Pirate& data`	Return `List` that is `this` with every occurrence of `data` removed
`-=`	`Pirate& data`	Remove every occurrence of `data` from `this`, and return the resulting `List&`
`==`	`const List& list`	Determine if every data element of `this` is the same (in same order) as in `list`
`!=`	`const List& list`	Determine if any data element of `this` is different from those in `list`
`[]`	`int pos`	Return a reference to the Pirate object stored in position `pos` of `this`

  - the Pirate class:

Operator	*Parameter*	*Functionality*
`<<`		Print out the `Pirate` information

- Your program must test all these classes using the main function found here: a3main.cpp
- The output should be formatted exactly as found here: a3output.txt

➢ Non-functional requirements:

- Design: the program must follow proper object-oriented design principles, including but not limited to:
  - encapsulation of data and functionality within classes, i.e. the principle of least privilege
  - easy modifiability and extensibility
  - Hint: you do not need any global variables; you only need one global function (main) -- all other functionality must be encapsulated within classes
- Implementation:
  - the program must not use classes from any library other than the C++ Standard Library
  - the program **must not** use any classes, containers or algorithms from the Standard Template Library (STL)
  - the program must explicitly deallocate all dynamically allocated memory – use `valgrind` to check for memory leaks
- Organization: the program must be correctly separated into source and header files
- Platform: the program must be written in C++, and it must compile and execute on the SCS lambda servers

Grading

➢ Assignment grade:

- Your original grade will be computed based on how much of the functional requirements your program implements
- Your final assignment grade will be the total of your original grade and bonus marks, minus deductions

➢ Deductions:

- Any submission instruction or non-functional requirement that you do not follow will result in severe deductions, including but not limited to:

  - **100 marks** if:
    - the assignment is marked **Missed** in WebCT
    - any submitted files (including the Makefile) are missing, or corrupt, or in the wrong format
    - the code does not compile using g++ on the SCS lambda servers
  - **50 marks** if:
    - the design does not generally follow the principles of encapsulation and least privilege
    - global variables or global functions (other than main and overloaded stream insertion operators) are used
    - supplied main function is not used
  - **20 marks** if:
    - the submission consists of anything other than exactly one diagram file and exactly one `tar` file
    - the program consistently crashes
    - the code is not correctly separated into header and source files
    - the readme file is missing or incomplete
  - **10 marks** for missing comments or other bad style (non-standard indentation, improper identifier names, etc)

➢ Bonus marks:

- Up to **10 extra marks** are available for fun and creative additional features