COMP 348: Principles of Programming Languages
Assignment 1 on Java and C

Winter 2022, sections E

January 19, 2022

Contents

1 General Information

Date posted: Wednesday January 19th, 2022.

Date due: Wednesday February 9th, 2022, by 23:59[1].

Weight: 5% of the overall grade.

2 Introduction

This assignment targets two programming paradigms:

1) Object-Oriented using Java, 2) Procedural Programming using C.

3 Ground Rules

You are allowed to work on a team of **3 students at most (including yourself).** Each team should designate a leader who will submit the assignment electronically. See Submission Notes for the details.

ONLY one copy of the assignment is to be submitted by the team leader. Upon submission, you must book an appointment with the marker team and demo the assignment. All members of the team must be present during the demo to receive the credit. Failure to do so may result in zero credit.

This is an assessment exercise. You may not seek any assistance from others while expecting to receive credit. You must work strictly within your team). Failure to do so will result in penalties or no credit.

4 Your Assignment

Your assignment is given in two parts, as follows. 1) Object-Oriented Programming using Java, 2) Procedural Programming using C.

[1] see Submission Notes

4.1 Object-Oriented Programming using Java

Q 1. Define the following classes and interfaces:

i. NamedObject (interface)

- `getName()`: a method to return object's name. Provide a default implementation that returns the object's Runtime class name[2].

ii. Shape (interface, extends NamedObject)

- `getPerimeter()`: a method to return shape's perimeter.
- `getArea()`: a method to return shape's area.

iii. Printable (interface)

- `print()`: a void method to print object's info to the console.
- A static `print()` method that receives a list of printables and calls their `print()` methods[3].

iv. PrintableObject (abstract class, implements NamedObject and Printable interfaces)

- `toString()`: overridden, returns the object's name by calling `getName()` method.
- `print()`: prints the text returned by `toString()` method. The information is printed in single line.

v. Rectangle (class, implements Shape and extends PrintableObject)

- Two sides as double: define the attributes and the corresponding accessors and mutators.
- Implement standard constructors: no-arg as well as specialized constructor that receives the sides.
- `toString()`: override and return the shape name (by calling the method in the base class), followed by the two values for the sides (separated by comma).
- A static `parse()` method that receives an input string and returns an instantiated `Rectangle` whose sides are initialized with the values in the input string. The input

[2]Use may use `this.getClass.getSimpleName()`
[3]See 4.1 Implementation requirements

string is in comma separated format,i.e.: "`Rectangle,2,3.5`". The method returns the object as `Rectangle`.

- Implement `getPerimeter` and `getArea` methods inherited from the `Shape` interface.

vi. Circle (class, implements Shape and extends PrintableObject)

- : radius as double; define the attribute and the standard accessor and mutator.
- Implement standard constructors: no-arg as well as specialized constructor that receives the radius.
- : `toString()`: override and return the shape name (by calling the method in the base class), followed by the value of the radius (separated by comma).
- : A static `parse()` method that receives an input string and instantiates and returns a `Circle` object whose radius is initialized with the value in the input string. The input string is in comma separated format,i.e.:
  "`Circle,1`". The method returns the object as `Circle`.
- Implement `getPerimeter` and `getArea` methods inherited from the `Shape` interface.
- Implement/override the `getName()` method and make sure the returned name is in ALL-CAPS.

Q 2. Using the above, write a java program that does the following:

- Read a file containing at least 7 shapes (each shape is provided in a single line, in comma separated format);
- Parse and store the shapes in an array (or collection) of `Shape`s.
- Sort the shapes by name and area.
- Print the sorted elements; Use `Printable.print()` method.

Implementation Requirements

- Use try-with-resources to open the file. The input file must be given by the user.
- Use `Arrays.sort()` or `Collections.sort()` to sort the shapes. You need to implement a `Comparator` class. Use anonymous class implementation.
- No additional classes / interfaces are allowed. The Shape and its sub-classes may not implement the `comparable` interface.

- You should strictly use the classes defined in 1. You may not define any additional classes nor change classes hierarchy, except for the anonymous comparator class
- Make sure all exceptions are caught. In case of format error or any particular runtime error, the program prints the details of the error and simply terminates.
- To implement static `Printable.print()` method, use "vararg" for the argument type and use Java "foreach" to loop through the elements.

Question: Despite the fact that all shapes are either `Rectangle` or `Circle`, you cannot pass an a `Shape[]` array as `Printable[]` to the `print` method. Why?

Solution: Use the following solution:

```
Shape[] shapes = ...
Printable.print(java.util.Arrays.copyOf(shapes, shapes.length, Printable[].class));
```

4.2 Procedural Programming with C

IMPORTANT: Make sure the C++ compiler option is turned OFF.

Functions and Pointers

Q 3. Write the following five "aggregate" functions in C that receive an array of floats and return a single value.

i. $\min(\{a\}_i)$: that returns the minimum of the elements in a given array.
ii. $\max(\{a\}_i)$: that returns the maximum of the elements in a given array.
iii. $\text{sum}(\{a\}_i)$: that returns the sum of the elements in a given array.
iv. $\text{avg}(\{a\}_i)$: that returns the average of the elements in a given array.
v. $\text{pseudo\_avg}(\{a\}_i) = (\min(\{a\}_i) + \max(\{a\}_i))/2$.

Note: that the function receives the array size as a parameter. An example a *min* function, namely `minf` is given in the following:

```
float minf(float* arr, int size); // to be declared in a header file


float arr[] = {1, 4, 5, 6, -1};
float m = minf(arr, 5);
printf("%f", m); // -1.000000
```

Declare all five functions in an include file called `AGGREGATE.H` and implement them in `AGGREGATE.C`. Use proper include guards.
In case the input array is `NULL` or the size is non-positive, the function prints "`FATAL ERROR in line ...`" where "..." represents the line number and the program is aborted (see Ref 5).

Q 4. An array of pointers to functions may be defined in the same way that a regular array is defined, with the exception that the type of the elements is a pointer to a function. The following defines an array of two pointers to functions: `f` and `f2`:

```
// Given the following function prototype
void f(void);
void f2(void);

// The following declares an array of one pointer
void (*farray[2])(void) = { &f, &f2 };
```

Define an array of function pointers called `aggregates`, that contains 5 pointers to the 5 functions you implemented in the previous question. Define two arrays of float numbers that contain 5 and 10 random numbers, respectively. Write a short program using a **for** loop, that iterates through the array of functions and calls each function on the two sample arrays you defined. The loop prints the calculated values in 10 separate lines.
Note: You must include the `AGGREGATE.H` file that you created in previous question[4]. Do not redefine the function.

Q 5. A *typedef* in C lets the programmer create an additional name (alias) for another data type, without create a new type. Using typedef, i) define an alias for the aggregate function pointers in the above questions; ii) revise the declaration of the array of pointers in the previous question by using the new typedef.
Additionally, iii) rewrite the program in the previous question so that instead of reporting the final answers, it also prints the corresponding aggregate function names associated with the values. For example instead of printing `-1.000000` for the result of `fmin` on the sample array of 5, it prints `FMIN: -1.000000` on the line corresponding. To do so, create a constant array of C strings and use it in the for loop.

```
const char* funcnames[] = { "FMIN", "FMAX", ... };
```

[4]You need to add the `AGGREGATE.C` file in your project in order to compile the project into an executable.

Lists and Linked Lists Implementation

In this exercise, we implement a simple list, using "*singly linked-list*" data structure in C. The data structure is a specified as follows:

> A *list* contains zero or more *elements*. An element of a list can be either an *atom* or a list. A list may also be empty, in which case is represented by `NIL`.
>
> Two key operations are defined to access the elements of a list: **car** and **cdr** (see Ref 6).
>
> - Operation *car* (also: first) takes a list as an argument and returns the head of the list. The head of the list is the first element in the list.
> - Operation *cdr* (also: rest) takes a list as an argument and returns the tail of the list. The *tail* of the list, is a sub-list (list) that the head of the list is pointing to. The tail of a empty list is considered empty.

The following provides the necessary definitions to implement a list in C. In this exercise we assume atoms are of `char` data type, only.

```
typedef enum { ATOM, LIST } eltype;
typedef char atom;
struct _listnode;
typedef struct {
  eltype type;
  union {
    atom a;
    struct _listnode* l;
  };
} element;
typedef struct _listnode {
  element el;
  struct _listnode* next;
} * list;
const element NIL = { .type=LIST, .l=NULL };
```

Q 6. Using the above, implement the following functions:

i. `element aasel(atom a);` atom as element, that returns an element whose content is set to atom a.

ii. `element lasel(list l);` list as element, that returns an element whose content is set to the list, pointed by l.

iii. `element car(element e);` that returns the head of the list, represented by e; returns `NIL`, if e is not a list.

iv. `list cdr(element e);` that returns tail of the list, represented by e. The tail of a list with one or zero element is `NIL`. The tail of an element that is not a list is also `NIL`.

v. `list cddr(element e);` that returns the `cdr` of the `cdr` of the list, represented by e.

vi. `list cons(element e, list l);` that creates a new list whose *car* and *cdr* are the element e and the list l. The memory for the newly created list is to be allocated dynamically[5].

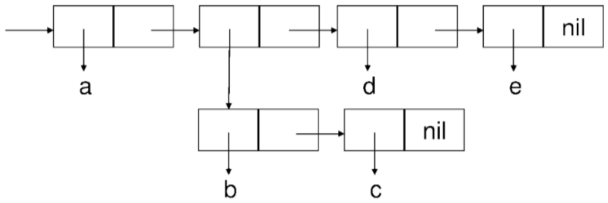
vii. `list append(list l1, list l2);` that creates a new list whose elements are shallow copies of elements in $l1$ and $l2$, appended.

viii. `void lfreer(list l);` that frees all the memory previously allocated by the whole list (including all its elements and its inner lists)

ix. `void print(element e);` that prints the content of the element e. If e is an atom, it prints the char symbol embraced in spaces, and if e it is a list, it (recursively) prints the elements of the list enclosed in parentheses ('(' and ')'). If e is `NIL`, the word "NIL" is printed (see the following example).

[5]Note that you only need to allocate one linked node for the first element. The tail of the newly created list points to the same list pointed by l

Q 7. Write a short code to create and print the following list:



Q 8. Print the car and the cdr of the above list, as well as the car of the car of the original list.

The output must look like the following:

```
( a ( b c ) d e )
 a
(( b c ) d e )
NIL
```

Make sure the list is freed before your program terminates.

5 What to Submit

The whole assignment is submitted by the due date under the corresponding assignment box. Your instructor will provide you with more details. It has to be completed by ALL members of the team in one submission file.

Submission Notes

Clearly include the names and student IDs of all members of the team in the submission. Indicate the team leader.

IMPORTANT: You are allowed to work on a team of 3 students at most (including yourself). Any teams of 4 or more students will result in 0 marks for all team members. If your work on a team, ONLY one copy of the assignment is to be submitted. You must make sure that you upload the assignment to the correct assignment box on Moodle. No email submissions are accepted. Assignments uploaded to the wrong system, wrong folder, or submitted via email will be discarded and no resubmission will be allowed. Make sure you can access Moodle prior to the submission deadline. The deadline will not be extended.

Naming convention for uploaded file: Create one zip file, containing all needed files for your assignment using the following naming convention. The zip file should be called a#_studids, where # is the number of the assignment, and studids is the list of student ids of all team members, separated by (_). For example, for the first assignment, student `12345678` would submit a zip file named `a1_12345678.zip`. If you work on a team of two and your IDs are `12345678` and `34567890`, you would submit a zip file named `a1_12345678_34567890.zip`.
Submit your assignment electronically on Moodle based on the instruction given by your instructor as indicated above: `https://moodle.concordia.ca`
Please see course outline for submission rules and format, as well as for the required demo of the assignment. A working copy of the code and a sample output should be submitted for the tasks that require them. A text file with answers to the different tasks should be provided. Put it all in a file layout as explained below, archive it with any archiving and compressing utility, such as WinZip, WinRAR, tar, gzip, bzip2, or others. You must keep

a record of your submission confirmation. This is your proof of submission, which you may need should a submission problem arises.

6 Grading Scheme

Q1 20 pts

Q2 15 pts

Q3 10 pts

Q4 10 pts

Q5 5 pts

Q6 30 pts

Q7 5 pts

Q8 5 pts

Total: 100 pts.

References

1. Java Comparator:
   `https://docs.oracle.com/javase/8/docs/api/java/util/Comparator.html`
2. Java Arrays:
   `https://docs.oracle.com/javase/8/docs/api/java/util/Arrays.html`
3. Java Class: `https://docs.oracle.com/javase/8/docs/api/java/lang/Class.html`
4. Java String Format: `https://www.javatpoint.com/java-string-format`
5. C Standard Predefined Macros:
   `https://gcc.gnu.org/onlinedocs/cpp/Standard-Predefined-Macros.html`
6. CAR and CDR: `https://en.wikipedia.org/wiki/CAR_and_CDR`
7. Aggregate Function: `https://en.wikipedia.org/wiki/Aggregate_function`
8. COMP 348 Course Pack, C. Constsantinides, Concordia University, 2018, Chapter 5: Lists.