CSCI 1321 (Principles of Algorithm Design II), Fall 2001

Homework 2[1]

Assigned: September 13, 2001.

Due: September 20, 2001, at 5pm.

Credit: 40 points.

> *Note:* The HTML version of this document may contain hyperlinks. In this version, hyperlinks are represented by showing both the link text, formatted like this, and the full URL as a footnote.

Contents

1 Tips of the week **1**
- 1.1 Software design tip . . . 1
- 1.2 UNIX text editors, revisited . . . 1
- 1.3 Compilation tips . . . 2

2 Problem statement **2**

3 Hints and tips for this assignment **3**
- 3.1 Using the template linked list class . . . 3
- 3.2 Dealing with multiple C++ files . . . 4

4 What to turn in **4**

1 Tips of the week

1.1 Software design tip

It is frequently easier to build programs by iteratively adding features than by first writing all code and then beginning the process of compiling and debugging. If you test after each adding each feature, any mistakes are likely to be in the newly added code.

1.2 UNIX text editors, revisited

If you tried `vi`, hated it, and went back to the Windows world, be advised that there are other editors available on the CS Linux machines. `emacs` is probably the most widely available; it is very powerful, though it also requires some learning. There are also "friendlier" GUI-oriented versions of both `vi` (`gvim`) and `emacs` (`xemacs`). See Some Useful Links (Mostly for CS 1320/1321)[2] for links to more information.

[1] © 2001 Jeffrey D. Oldham (`oldham@cs.stanford.edu`) and Berna L. Massingill (`bmassing@cs.trinity.edu`). All rights reserved. This document may not be redistributed in any form without the express permission of at least one of the authors.

[2] `http://www.cs.trinity.edu/~bmassing/Misc/class-links.html`

1.3 Compilation tips

Many compilers can be asked to provide warning messages (about what *might* be wrong with your program) in addition to the usual error messages. These warning messages may help find programming mistakes more easily than the default of producing only error messages.

To ask the g++ compiler to produce these messages, use the `-Wall` and `-pedantic` options, e.g., type `g++ -Wall -pedantic hello.cpp`. If you use a different compiler, find out whether it has similar options and how to turn them on.

To ask the g++ compiler to name the resulting output something other than `a.out`, use the `-o` option. For example, to compile `hello.cpp` and put the result in `hello` rather than `a.out`, type `g++ -Wall -pedantic hello.cpp -o hello`.

It is a pain to type these every compilation. Instead, you can store compilation commands in a `Makefile` and use the `make` command to compile your program, creating the executable.

For example, suppose you use the command `g++ -Wall -pedantic foo.cpp -o foo` to compile a C++ file named `foo.cpp` and create an executable called `foo`. Instead, you can create a file called `Makefile` containing these two lines of code:

```
foo:
tab g++ -Wall -pedantic foo.cpp -o foo
```

(*tab* denotes a tab character; using eight spaces will not work correctly.) To create the executable called `foo`, type `make foo`.

You could accomplish the same thing by creating a `Makefile` containing the single line

```
CXXFLAGS -Wall -pedantic
```

This defines a general rule for compiling a file named *whatever*`.cpp` and putting the resulting executable in *whatever*. With this `Makefile`, typing `make foo` will work as described above, as will `make hello`, etc.

The `make` programming language can automate many tasks. If you want to learn more, read the online GNU make documentation[3], particularly the first chapter.

2 Problem statement

According to a former Stanford graduate student now teaching at Columbia University, the database program running on the main Stanford mainframe computer had no built-in arithmetic operations. Even though numbers such as tuition and housing charges were computed by this program, the program had no integer or floating point number type.

Our task is to implement arithmetic on natural numbers, i.e., nonnegative integers, to use with this program. Let's use unary number notation. For example, the decimal number 7 would be represented by "0000000" and 3 would be represented by "000". Zero is represented by no digits: "".

We can recursively define any natural number as one of the following:

[3] `http://www.gnu.org/manual/make-3.79.1/make.html`

- zero
- one more than another natural number

For example, "000" is one more than one more than one more than zero. More succinctly, "000" is add1(add1(add1(zero))).

Note this recursive definition for numbers is very similar to the recursive definition of lists.

We will assume that the database program supports lists of booleans and implement unary arithmetic using the template linked list class discussed in class. Our database program must compute tuition charges so it needs `+`, `-`, `*`, integer division `/`, and exponentiation `^` (to compute interest on overdue balances). Our database program will never use subtraction to produce negative numbers, but it needs to be able to compare numbers, i.e., `>`, `>=`, `<`, `<=`, `==`, and `!=`. You need not define arithmetic assignment operators such as `+=` and `-=`, but your code may not use them either.

To get you started, file `unary.h`[4] contains some starter source code for the class definition, and file `test-unary.cpp`[5] contains a simple `main()` function for preliminary testing of your code. (Of course, you will want to write more extensive testing code.)

The starter source code includes:

- a definition of the word `natural` as the type for our numbers
- a definition of the number zero
- input and output functions, i.e., definitions of the `<<` and `>>` operators for unary (natural) numbers. For example, `cin >> n` will read a base-10 number from the standard input, storing it into the (unary) natural number n. `cout << n` will print the base-10 representation of the (unary) natural number n.

These input and output functions assume the following functions are defined:

- `add1()`: creates a larger number.
- `sub1()`: given a nonzero number, extracts its subnumber.
- `iszero()`: returns true if the number is zero, false otherwise.

Your mission in this assignment is to fill in the missing functions in `unary.h`. (Of course, you may add other functions if you need them.)

3 Hints and tips for this assignment

3.1 Using the template linked list class

The only functions you should use from the template linked list class (`Seq`) are its two constructors and member functions `empty()`, `hd()`, and `tl()`. You do not need to read and understand the source code for this class; it should be possible to use the required functions by following the example in sample program `list-example.cpp`[6] and this short description of the class[7].

[4] `http://www.cs.trinity.edu/~bmassing/CS1321_2001fall/Homeworks/HW02/Problems/unary.h`
[5] `http://www.cs.trinity.edu/~bmassing/CS1321_2001fall/Homeworks/HW02/Problems/test-unary.cpp`
[6] `http://www.cs.trinity.edu/~bmassing/CS1321_2001fall/SamplePrograms/list-example.cpp`
[7] `http://www.cs.trinity.edu/~bmassing/CS1321_2001fall/Notes/jdo-lists/`

3.2 Dealing with multiple C++ files

Fitting all the source code for complicated programs into one file would be unrealistic, so C++ supports storing code in multiple files. Historically, files ending with ".cpp" stored definitions, e.g., function definitions, while files ending with ".h" stored the corresponding declarations. Unfortunately, recent C++ changes have blurred these distinctions, so sometimes we put function definitions in header files, as in `unary.h`.

For this assignment, I recommend the following:

- Make sure that the files `seq.cpp`[8], `seq.h`[9], `unary.h`[10], and `test-unary.cpp`[11] are in the same directory.
- Do not change the given `#include` header directives, but feel free to add to them as necessary.
- Compile using the command `g++ -Wall -pedantic test-unary.cpp`. This will produce an executable named `a.out`. You can use the `-o` *output-file-name* g++ option if you desire.

Note that the code, as distributed, should compile, albeit with numerous warnings. Also, it will not run until code for the necessary functions is written.

If you are using a compiler other than g++, it is your responsibility to determine how to deal with multiple files.

4 What to turn in

Submit your source code as described in the Guidelines for Programming Assignments[12]. For this assignment, use a subject header of "cs1321 hw2", and submit a single file containing your revised version of `unary.h`. You do not need to send `seq.h`, `seq.cpp`, or `test-unary.cpp`; I will provide these files when I test your code.

[8] `http://www.cs.trinity.edu/~bmassing/CS1321_2001fall/SamplePrograms/seq.cpp`
[9] `http://www.cs.trinity.edu/~bmassing/CS1321_2001fall/SamplePrograms/seq.h`
[10] `http://www.cs.trinity.edu/~bmassing/CS1321_2001fall/Homeworks/HW02/Problems/unary.h`
[11] `http://www.cs.trinity.edu/~bmassing/CS1321_2001fall/Homeworks/HW02/Problems/test-unary.cpp`
[12] `http://www.cs.trinity.edu/~bmassing/CS1321_2001fall/Notes/pgmguidelines/index.html`