WARNING

You might have to download and install a fair amount of software on your computer before you can do this homework assignment, especially if you are using Microsoft Windows. Therefore I STRONGLY advise you to start downloading the software AS SOON AS POSSIBLE.

==========

SOFTWARE REQUIREMENTS

To do this assignment you need access to a computer that has the "lex" and "yacc" programs (or their GNU equivalents called "flex" and "bison") as well as a C compiler. Follow the instructions below, based on the kind of computer you have access to, to check whether you have the right tools on the computer or not. If you do not have access at all to a computer with the required tools, then see the 'Other cases' section below.

---------- Unix or Linux

If your computer uses a Unix or Linux operating system then the tools you need are probably already installed on your computer. To check that, open a terminal window (a terminal emulator, like "xterm" or "rxvt") to get a shell prompt, and try the following commands:

```
which lex
which flex
which yacc
which bison
which cc
which gcc
```

These commands will either give you a result like "/usr/bin/lex", in which case your computer has the corresponding program, or it will give you an error message like "command not found" (or no message at all, depending on your operating system), in which case your computer does not have the corresponding program. If you have at least one of the lex or flex programs, one of the yacc or bison programs, and one of the cc or gcc programs, then you are fine and you can start working on the homework assignment.

If you have both lex and flex, then use flex.
If you have both yacc and bison, then use bison.
If you have both cc and gcc, then use gcc.

If some of the tools you need are missing on your computer then you will have to use the package management system on your computer to install them. Use this package management system to install the packages for flex, bison, and gcc, and you will be fine. Flex, bison, and gcc are tools that are used very often by programmers so they are pretty much guaranteed to be available as packages on all Unix-like operating systems. Different Unix-like operating systems use different package management systems though, so read the documentation for your operating system if you need to learn how to install packages.

Come see me if you have problems.

---------- Mac OS X

Download and install the Xcode development environment from
https://developer.apple.com/xcode/
Then download and install the "Command Line Tools" from
https://developer.apple.com/download/more

After that, open a terminal window to get a shell prompt, and try the following commands:

```
which lex
which flex
which yacc
which bison
```

```
which cc
which gcc
```

These commands will either give you a result like "/usr/bin/lex", in which case your computer has the corresponding program, or it will give you an error message like "command not found" (or no message at all, depending on your operating system), in which case your computer does not have the corresponding program. If you have at least one of the lex or flex programs, one of the yacc or bison programs, and one of the cc or gcc programs, then you are fine and you can start working on the homework assignment.

If you have both lex and flex, then use flex.
If you have both yacc and bison, then use bison.
If you have both cc and gcc, then use gcc.

Come see me if you have problems.

---------- Microsoft Windows

If you use Microsoft Windows, the best thing to do is to install on your computer the Cygwin system. Cygwin is a set of programs that simulates a Unix machine on a Microsoft Windows computer. It includes flex (the GNU version of lex), bison (the GNU version of yacc), and gcc (the GNU C compiler).

WARNING: the Cygwin system is huge. It may require a day or two to download everything if you want to install the complete system. Fortunately you only need a small part of the whole system to do this homework assignment, which should take much less time to download. Use a fast Internet connection if possible.

Here is a step-by-step guide of how to install Cygwin on a Microsoft Windows computer.

- Download and run the setup-x86_64.exe program (or the setup-x86.exe program if you have an older 32 bit computer) from the Cygwin web site: https://www.cygwin.com/install.html

- Click on Next and select "Install from Internet".

- Click on Next three times and select "Direct Connection", then click on Next again.

- The setup program will then connect to the Cygwin web site and download to your computer a list of mirror sites from which you can download the rest of the software.

- In the "Available Download Sites" menu, select a download site somewhere in China or Japan (the closer the site is to you, the faster the download will probably be after that).

- Click on Next and the setup program will then use that download site to download the complete list of all the Cygwin software that you can install on your computer.

- You will then get a menu called "Select Packages".

- Click on the "+" inside a square next to the "Devel" menu in the window.

- Go down the list (it’s very long, make your window full screen, it will help...) and select the following three software packages:
  - bison
  - flex
  - gcc-core
  To select each of these, click once on the work "Skip" next to the software name. This word "Skip" should then change into a version number.

  Note: when you select software like that, the setup program might automatically select some additional software for you. For example, when

you select gcc-core, the setup program will also select a few other related packages that are necessary for gcc to run correctly. Do not worry about this: just select the three programs above, and let the setup program automatically select everything else for you.

Note: if you want to, you can select more packages than just the three packages above, when installing the Cygwin software. Of course the more software you select, the longer it will take to download... Do NOT select everything, as it would probably take a full day or two on a high speed connection to download the whole of Cygwin!

- Click on Next twice and wait for the setup program to download and install everything...

- Once everything is installed, you should have a "Cygwin" icon on your desktop (an icon that looks like a green right arrow inside a black letter "C"). Run this program.

- You should then get a window which will give you a Unix command line interpreter, where you can type all the usual Unix commands (ls, cd, mkdir, etc.)

- Try the following commands at the command prompt:

```
which flex
which bison
which gcc
```

Each command should give a result like "/usr/bin/flex" or "/usr/bin/gcc". This will tell you whether you have correctly installed flex, bison, and gcc. If you get an error message like "flex: command not found" then make sure you correctly typed the commands above. If you are sure you made no mistakes, then come see me.

- Once you have installed flex, bison, and gcc, you can start doing the homework assignment.

Note: since Cygwin is an implementation of Unix that works in top of the Microsoft Windows operating system, it means you will need to know all the basic Unix commands to use Cygwin!
To know in which current directory you are, type: pwd
To list the content of a directory, type: ls
To move into a directory called xyz, type: cd xyz
To move up out of the current directory, type: cd ..
If you do not know how to use Unix commands, you can find a good tutorial here: http://www.ee.surrey.ac.uk/Teaching/Unix/
You can also download the tutorial to your computer using this link: http://www.ee.surrey.ac.uk/Teaching/Unix/download.html
Let me know if you have any problem.

==========

MORE INFORMATION

More information about lex and yacc can also be found on the Internet. For example http://dinosaur.compilertools.net/ is a good starting point.

The complete manuals for flex, bison, and gcc can be found on the following GNU web sites:
http://www.gnu.org/software/flex/
http://www.gnu.org/software/bison/
http://www.gnu.org/software/gcc/

==========

SUBMITTING THE HOMEWORK ASSIGNMENT

Late homework assignments will not be accepted, unless you have a valid written excuse (medical, etc.). You must do this assignment alone. No team work or "talking with your friends" will be accepted. No copying from the Internet. Cheating means zero.

For each programming question below, you should submit the specification of the lexer, if any, the specification of the parser, if any, and a file with your English answer to the question, if any. All those files must be named in the same way:
- the name should start with "example"
- followed by the question number
- and end with ".l" for a lexer specification, ".y" for a parser specification, and ".txt" for an answer in English.

For example, when answering question 4 below, you will have to submit the specification of a lexer, the specification of a parser, and an answer written in English. The file containing the specification of the lexer must then be named "example4.l", the file containing the specification of the parser must be named "example4.y", and your answer written in English must be in a file named "example4.txt". All files must be named using lowercase letters (NO uppercase letters!)

Following this naming convention is very important, as I will use a computer program to automatically compile, run, test, and print your homework assignment. Submitting files with the wrong names will result in your assignment not being processed correctly by my program, which means you will get a zero and you will have submitted your homework assignment for nothing. I will not have time to change the program I use just because one or two students failed to follow the instructions in this homework assignment. It is YOUR responsibility to ensure that the files you submit are the right files with the right names.

To submit the homework assignment, do the following:

a) Use the "mkdir" command to create a directory, using your student ID number as the name of the directory. For example:
```
mkdir 1234567890
```

b) Put ALL the necessary files for your assignment into that directory.
- Do NOT create sub-directories.
- Make sure all your programs still compile and work.
- Do NOT put into that directory executable files (.exe) or automatically generated files (like y.tab.c).

The directory must then contain the following files (assuming you answered all the questions in the assignment) and nothing else:

```
example1.l   example2.txt example4.l   example5.l   example6.l
example1.txt example3.l   example4.txt example5.txt example6.txt
example2.l   example3.txt example4.y   example5.y   example6.y
```

c) For students using Unix or Mac OS X, use the "tar" command to create an archive of the directory. For example:
```
tar -cvf 1234567890.tar 1234567890
```
This command should be run from the parent directory that contains the 1234567890 directory, otherwise it will not work.

For students using Microsoft Windows, create a compressed zipped folder using the zip software on your computer. The resulting zipped file should be named like:
```
1234567890.zip
```
- make sure you create a zip file, not a rar file!
- use ".zip" in the file name, not ".ZIP".

d) For students using Unix or Mac OS X, use the "tar" command to verify the content of the archive file. For example:
```
tar -tvf 1234567890.tar
```

For students using Microsoft Windows, you can verify the content of the compressed zipped folder by double-clicking on it and looking at the content.

If the archive or zipped folder does not contain all the files required for the homework assignment (see above for the list) then you did something wrong. Try again or ask me for help.

If the archive or zipped folder contains files which should not be there (executable files, or automatically generated files like y.tab.c) then delete the archive or zipped folder, remove the extra files from the original directory, and re-create the archive or zipped folder like you did in (d) above.

e) Submit the archive or zipped folder on iSpace.

WARNING: do NOT send me Microsoft Word documents! All the ".txt" files containing your English answers should be plain ASCII text files, nothing else. Use Notepad or Wordpad on Microsoft Windows to write your answers, do not use Microsoft Word.

For the students using Cygwin on Microsoft Windows, you can check what the files look like by using one of the following commands in a Cygwin window (which command will work depends on exactly which Cygwin packages you chose to install on your computer):

```
cat the-name-of-the-file
less the-name-of-the-file
more the-name-of-the-file
```

You can use these commands on .l, .y, and .txt files, and you will see the exact content of the files. If you try to use these commands on .exe or .doc files you will only see garbage, since .exe and .doc files are binary files.

If you have any problem or question, contact me BEFORE the deadline for the homework assignment!

==========

HOMEWORK ASSIGNMENT QUESTIONS - Introduction to Lex and YACC

Explain in English, step by step, and very carefully every answer you give!

1) Download the tutorial lexyacc.pdf from the iSpace web site (ASCII and HTML versions are also available from https://ds9a.nl/lex-yacc/)

Implement, compile, and test the example at the beginning of Section 3 on page 3 of the tutorial. The first command "lex example1.l" uses the lexer specification in file example1.l to generate the full C code for a lexer (use "flex" with the -lfl option if your computer does not have the "lex" command). The generated C code is stored in a file called lex.yy.c. Compiling that C file using the C compiler cc (or gcc, depending on what you have on your machine) creates a lexer program that you can then execute by typing: ./example1
Note: in the tutorial, ^D (EOF) means pressing the Control and d keys at the same time on the keyboard, which indicates the end of the input (EOF means End Of File).

Once you have tested the example1 program, use your favorite editor to look at the content of the lex.yy.c file. You will see that the code that was generated for the lexer is about 1500-2000 lines long, even though the specification you created in the file example1.l is only about 7 lines. Most of the code in the file lex.yy.c is actually generic code that is always the same for all generated lexers. The lexer implements a deterministic finite automaton which is described by the tables of numbers towards the beginning of the file. Use the search feature of your editor to search for the string "example1" in lex.yy.c and you will see that most of the things you typed in the file example1.l have been copied here and there in the file lex.yy.c.

Give the following input to the lexer:

```
st artstartst1artsstoppjlk
```

1A) What is the output from the lexer?

1B) Explain why the output looks like this.

(Write your answers in English in an ASCII text file called example1.txt in the same directory).

Give the following input to the lexer:

start start stop start

1C) What is the output from the lexer?

1D) Why are some of the lines indented by one space character?

1E) Why is there an empty line at the bottom of the output?

2) Implement the second example lexer from Section 3.1 of the tutorial.

2A) What is the output from the lexer when giving the following three sentences as input to the lexer:

ab12
12ab
a12b

2B) Explain why the output looks like this (including the empty lines).

(Write your answers in English in an ASCII text file called example2.txt in the same directory, and so on).

3) Implement the third example lexer from Section 3.2 of the tutorial.

3A) Give to the lexer the following input:

}" x{-/-; }}

What is the output?

3B) Such a configuration file is obviously not grammatically correct, but the lexer still accepts the input and does not generate any error. Why?

4) Implement the lexer and parser from Section 4.1 of the tutorial. Note: the header code described at the top of page 8 should be put into the file example4.y, followed by "%%" (on a line by itself, without the double quotes), followed by the grammar specification from the middle of page 7. The start symbol for the grammar is assumed to be the first nonterminal in the grammar (in the present case, the "commands" nonterminal). Note that a YACC grammar specification for a production uses ":" where I have used "->" in class, and a semicolon has to be present after the last production for a given nonterminal. Use the command "bison -y" instead of "yacc" if you do not have yacc on your computer (i.e. use "bison -y -d example4.y" to generate the parser from example4.y).

Once you have tested the example4 program, use your favorite editor to look at the content of the y.tab.c file. You will see that the code that was generated for the parser is about 450 lines long (over 1000 lines long if you used bison instead of yacc), even though the specification you created in the file example4.y is only about 40 lines. Most of the code in the file y.tab.c is generic code that is always the same for all generated parsers. The parser implements a pushdown automaton which is described by the various tables of numbers towards the beginning of the file. Use the search feature of your editor to search for the string "example4" in y.tab.c and you will see that most of the things you typed in the file example4.y have been copied here and there in the file y.tab.c. Also look at the file y.tab.h. You will see in it C code like this (the exact number will vary depending on whether you use yacc or bison):

```
#define NUMBER 258
#define TOKHEAT 259
#define STATE 260
```

```
#define TOKTARGET 261
#define TOKTEMPERATURE 262
```

This code defines the tokens generated by the lexer (tokens, which are categories of lexemes, are simply named using integers in the C code generated by lex and yacc).

4A) Give the following input to the program:

```
target temperature 17
target temperature -17
```

What is the output?

4B) Explain why in details, for each line of output.

5) Implement the lexer and parser from Section 4.2 of the tutorial. Note: the variable yytext contains the most recent lexeme matched by the lexer, represented as a C string (an array of characters). The variable yylval should contain a C value for the lexeme (the semantic value associated with the lexeme). Hence the need to use the atoi function in the NUMBER case in the lexer, which is used to transform the lexeme (C string) stored in yytext (for example "23") into a semantic value (an integer, in this example) that can be stored into yylval (the integer 23). The semantic values put in yylval by the lexer can then be accessed in the parser using one of the $n pseudo-variables (for example $3 in the action for the target_set production of the grammar used by the parser is the value of yylval corresponding to the token NUMBER computer by the lexer).

To create example5.y, copy example4.y and replace the grammar productions for target_set and heat_switch with the new ones from Section 4.2 of the tutorial. Everything else remains the same.

5A) Give the following input to the program:

```
target temperature 17
heat off
heat on
```

Show the result.

5B) Explain in detail how the two characters "17" from the input are processed by the lexer and parser to be finally printed by the parser as an integer.

6) Implement the lexer and parser from Section 4.3 of the tutorial. Copy example3.l into example6.l and modify that file appropriately. Copy example5.y into example6.y and completely change the grammar. Insert the following C code on a line by itself at the beginning of *both* example6.l and example6.y, right after the "%{" at the beginning of each file:

```
#define YYSTYPE char *
```

This tells the lex and yacc programs that the generated lexer and parser should use the "char *" type for the shared global variable yylval. Also modify the specification of the tokens (the %token line of the file example6.y) so that all the tokens returned by the code in example6.l are defined in example6.y.

Check that the lexer and parser work with the example given at the end of Section 4.3 in the tutorial.

6A) Give the following input to the program:

```
zone "foo/bar"{file "bar-foo"; foo; type hint; "foo/bar"; file "foo";};
```

What is the output?

6B) Explain why.

6C) What do you think about the quality of the error handling of your program?

Note: you can read the rest of the tutorial, it is going to be a good introduction to the kind of things you will have to do in later homework assignments. You can skip Section 5 though.