Date: Thursday, October 19th, 2006
Time: from 16h00 to 17h30
Content: Everything we have seen in class up to C pointers.
  - Unix operating system
  - Shell Scripting
  - Python
  - C (including pointers)

Exact content of the midterm will be discussed in a latter class and posted on the web.
- **skinner.cs.mcgill.ca**
  - Hardware/OS: SunOS 5.8 sun4u sparc SUNW,Ultra-4
  - CPUs: 4 x 400 MHz (sparcv9 processors)

- **willy.cs.mcgill.ca**
  - Hardware/OS: SunOS 5.8 sun4u sparc SUNW,Ultra-80
  - CPUs: 4 x 450 MHz (sparcv9 processors)

- **nova.cs.mcgill.ca**
  - Hardware/OS: SunOS 5.8 sun4u sparc SUNW,Ultra-60
  - CPUs: 2 x 450 MHz (sparcv9 processors)

- **mimi.cs.mcgill.ca**
  - Hardware/OS: SunOS 5.8 sun4u sparc SUNW,Ultra-250
  - CPUs: 2 x 400 MHz (sparcv9 processors)
- **troy.cs.mcgill.ca**
  - Hardware/OS: Gentoo GNU/Linux running on a 2.6 kernel
  - CPUs: 2 x 3.40 GHz (Intel Pentium 4 processors)

- **freebsd.cs.mcgill.ca**
  - Hardware/OS platform: FreeBSD 5.5-RELEASE-p3
  - CPUs: 2 x 3.40 GHz (Intel Pentium 4 processors)
Give a regular expression that will match on the following:

- string “Quiz”
- line starting with string “Quiz” or a digit
- line ending with string “Quiz”
- the string quiz, where the characters can be any case, e.g., QuIz, quiz, Quiz, etc.
- the string quiz, where it can be mis-spelled with K for Q and W for U, e.g., kwiz, qwiz, etc.
- a string of at least 3 digits, starting with 7
- lines containing no non-numeric characters, but at least one numeric character.

You have a directory containing a lot of files and subdirectories, and you want to copy all of them except for the directory called big_dir. How do you do it?
The C programming language was created as a successor for B and BCPL.

It’s creation was parallel to the development of early Unix operating systems (1969-1973).

At the time, one of C’s strength was it’s portability.

The first widely available description of the language appeared in 1978, The C Programming Language (also known as the white book).

One of C’s most popular successor is C++, release in 1986.
Traditionally, Hello World is the first application you write when starting with a new programming language.

```c
#include <stdio.h>

int main(int argc, char *argv[]) {
    printf("Hello World");
    return 0;
}
```
From the user's perspective, building a C program can be broken down into three steps:

- **Writing the source**: Using an editor to write the source.
  - You can use any text editor to write C code.
  - Old-school C programmers often use Unix text editors such as Vi or Vim.
  - For large-scale projects, an IDE (integrated development environment) is preferable.
  - Whatever editor you use, it should feature syntax highlighting.

- C programs are usually composed of several source files (we will take a look at this latter).
The next step is to compile the program to a format the operating system can run.

A compiler is a program that translates a language to another.

- A C compiler translates C code to machine code.
- A Java compiler translates Java code to byte code.

For this course, we will use the GNU cc compiler (also known as gcc).

This compiler is installed on all the lab machines and servers.
By default, the gcc compiler produces an executable files named `a.out`.

You can execute your program by running the `a.out` file.
- Don’t forget that `a.out` must be chmod executable. The compiler usually takes care of this.

Executable are compiled for specific architecture. If you compile a program in the labs (Intel), it will not run on Mimi (Sun).
C Compilation Processor

Source code

Preprocessor

Compiler

Assembler

Libraries

Linker

Executable code
The preprocessor is the first step of the compilation process.

- It prepares the source files for the compiler.

- The preprocessor is responsible for . . .
  - Removing all the comments from the source files.
  - Executing the preprocessor directives (#define and #include).
As previously mentioned, the compiler translate source code from one language to another.

The gcc compiler translate C code to assembler.

Let's take the Hello World example.

```c
#include <stdio.h>

int main(int argc, char *argv[]) {
  printf("Hello World");
  return 0;
}
```
main:
pushl %ebp
movl %esp, %ebp
subl $8, %esp
andl $-16, %esp
movl $0, %eax
subl %eax, %esp
subl $12, %esp
pushl $.LC0
call printf
addl $16, %esp
movl $0, %eax
leave
ret
main:
save %sp, -112, %sp
st %i0, [%fp+68]
st %i1, [%fp+72]
sethi %hi(.LLC0), %o1
or %o1, %lo(.LLC0), %o0
call printf, 0
nop
mov 0, %i0
ret
restore
The assembler takes assembly code and transforms it into object code.

Although object code is mostly composed of machine code, it cannot be executed by the operating system.

- Object code does not have the necessary references to external functions and libraries to properly operate.
A linker takes the various outputs of a compiler and combines them to create an application.

- Sources files are compiled separately by the compiler.
- Those sources might reference a function that exists elsewhere.
- The compiler leaves empty references to those functions.
- The linker fills those references using the compiled output of all the files and the libraries available on the system.

Once all the empty references have been resolved, the linker combines all the compiler output to create an executable.
- C itself is a relatively small programming language.
- Most of its functionalities is provided through function libraries.
  - C provides a library for read/write to files and the screen.
  - C provides a library to handle complicated math functionalities.
  - C provides a library to retrieve the current time from the OS.
- A programmer is free (and encouraged) to use these libraries.
- The linker takes care to resolve references to library calls.
As previously mentioned, Gcc is the Gnu C Compiler.

Gcc encapsulates all the different step of the compilation process.

- Create main.i, the preprocessed version of main.c
  
  ```sh
gcc -E main.c
  ```

- Create main.s, the assembler code of main.c
  
  ```sh
gcc -S main.c
  ```

- Create main.o, the object code of main.c
  
  ```sh
gcc -c main.c
  ```

- Create a.out, the compiled executable of main.c
  
  ```sh
gcc main.c
  ```
Gcc options

- -o filename: allows you to specify the name of the output executable (instead of a.out).
- -v: enable verbose mode (more output information).
- -w: suppresses warning messages (bad idea)
- -W: extra warning messages (good idea)
- -Wall: all warning messages (best idea)
- -O1: Optimize code for size and speed.
- -O2: Optimize even more.
C vs Java - Similarities

- C and Java have very similar syntax.
  - Variable / function declarations
  - Variable types: char, int, long, float, double
  - Conditional statements: If, For, While

- The notion of visibility is similar
  - Variables declared in functions only exists in functions
C vs Java - Difference

- C programming is much more low level
  - Pointers and memory allocation
- C is not object oriented
  - No classes, no static methods, no interfaces.
  - Libraries are completely different (no LinkedList, etc).
  - Structures allow to group data together
- C doesn't have Strings or boolean
  - Strings are replaced by character arrays.
  - boolean simply doesn't exist.
- C is a single pass compiler
  - Need to declare functions
  - Header files
- C has a preprocessor
A C function has the same syntax as a Java function.

```c
type function_name (parameters)
{
    local variables

    C Statements

}
```

Functions have a return type, just like Java.

However, unlike Java, they are not part of a class.

In C, all functions behave as they were static.
Two types of variables exist in C

- Primitives
- Pointers

C primitives are very similar to Java primitives

- Char (1 byte, -127 to 128)
- Unsigned char (1 byte, 0 to 255)
- Short (2 bytes, -32768 to 32767)
- Int (4 bytes, \(-2^{91}\) to \(2^{91} - 1\))
- Float (4 bytes, ...)
- Double (8 bytes, ...)

An unsigned variable is a numerical variable without a negative bit (thus allowing for larger numbers).

Notice there are no booleans or strings!
Global Variables

- Variables not declared in a function are referred to as global.
- Global variables can be accessed by any function in the program.
- Global variables are very similar to static variables, only one copy exists.
- Global variable should be avoided
  - Since any functions can access global variable, it's difficult to control access to those variable (an complicated debugging).
  - They are not considered clean.