Phys4051: C Lecture 1

- What is a (computer) program?
- C Syntax & Functions
- Difference between ANSI C and LabWindows C

Computer Hardware

Block diagram of a typical microcomputer (from Jones)
Major Components:
- CPU (= Central Proc. Unit)
- RAM / ROM (= Memory)
- I / O (=Input / Output)
- Address and Data Bus

Computer Hardware (2)

1. Build your own simplified 2 bit ALU (Arithmetic / Logic Unit) consisting of:
   - Half Adder, Half Subtractor and Multiplier
2. Use it, i.e., program it

Simple ALU (Arithmetic/Logic Unit)

ALU (2) Instruction Code

<table>
<thead>
<tr>
<th>Code</th>
<th>A₀</th>
<th>A₁</th>
<th>Purpose</th>
<th>Q₀</th>
<th>Q₁</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Add</td>
<td>Sum</td>
<td>Carry Bit</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Subtract</td>
<td>Diff.</td>
<td>Borrow Bit</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Multiply</td>
<td>Product</td>
<td>Don’t Care</td>
</tr>
</tbody>
</table>

Simple CPU (“Hardcoding”)
Program for Our CPU

<table>
<thead>
<tr>
<th>Cycle</th>
<th>Instruction</th>
<th>Code</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Add 0 + 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Multiply 1*1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Subtract 1 from 1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Creating Executable Code
- **Compiled Code**
  - (ASCII) Source code is compiled into a standalone executable file.
  - Example: C
  - LabWindows Environment
- **Interpreted Code**
  - An interpreter (i.e., a program) compiles the code (usually by line by line) at run time.
  - Example: BASIC
  - Java Virtual Machine (VM)

Conclusions about Programs:
- A program consists of **instructions** and **data**.
- A CPU can only execute binary instructions, i.e., “machine language.”
- Higher level programming languages use mnemonics and variables.
- To convert the instructions from the higher level language back to machine language a **compiler** is used.

C Language Components
- **Keywords:**
  - Examples: main, for, while, if, else..
- **Variables (Properties)**
  - Examples: integers, floating point...
- **Operators**
  - Example: +, -, *, /, <, &,
  - Example: !,
- **Functions (Methods)**
  - ANSI C, Compiler Supplied & User Defined

C Syntax
- C is **CASE SeNsITiV**!
  - Example:
    ```c
    int volt = 1;
    int Volt = 2;
    int Volt = 3;
    int Volt = 4;
    int volt = 5, Volt = 6;
    ```

First C Example
- **Program**:
  ```c
  #include <stdio.h>
  main()
  {
    printf("Hello");
  }
  ```
- **Output**:
  Hello
C-Program: Compilation

- The C-compiler makes two passes:
  - First it pre-compiles: it looks for variables, functions prototypes, files and information to be pasted into the current program.
  - After the first pass, the program is compiled into executable code.

C-Program Execution

- Execution always begins at the first (and only) "main" statement.
- Execution (normally) ends when the end of the main "function" is reached, i.e. at the closing bracket that follows "main."

C Syntax: Functions

- The C language uses a function based syntax.
- A C program consists of various functions that may call other functions.
- Every C program must contain one function called main(). This is where the program starts!

C Syntax: Most Simple Function

- Function Prototype:
  ```c
  #include <stdio.h>
  main()
  ```
- Function Header:
  ```c
  { }
  ```
- Statement(s):
  ```c
  printf("Hello");
  ```

C Functions: Arguments

- A C function can have any number of arguments (or none) but it can return at most only one value.
C Functions: Types

1. Functions that are part of ANSI C
   - Every (ANSI-C) compiler can implement them. Examples of these are: simple Math, (File) I/O, type conversion functions

2. Compiler Supplied Functions
   - Additional functions depending on the compiler. (LabWindows supplies Windows Interface functions and Instrument control functions.)

3. User Defined Functions
   - Whatever the user chooses to create. The user may or may not include functions listed above.

Function Elements 1: Prototypes (Declarations)

- Prototypes Consist of:
  - Function Return Type
  - Function Name (First letter Uppercase)
  - Arguments (variable type and name)

- Sample Function Prototypes:
  - float Calc( int x, float y);
  - void ClearScreen( void );

Function Elements 2: Body: Header & Statements

- Function Header
  - Identical to function prototype (except for semicolon!)

- Statements
  - All statements end in semicolon!

Function Elements 3: “Calling” the Function

1. To “call” (= execute) a function, only the function name and the arguments are used.

2. Example: (see prev. slide for prototypes)
   - a = Calc( u, 1.3);
   - ClearScreen();

3. If the function returns a value, assign that value to a variable!

Function Example 1: KeV

- Task: Convert the (working) program #1 on the next slide so that:
  - the computations for the kinetic energy are done in a separate function;
  - call this function from main.

- (For simplicity, the program does not contain any I/O statements.)

- Note: NO VARIABLE TYPE DECLARATIONS ARE NEEDED WHEN CALLING A FUNCTION!
Function Example: KeV (2): Program 1

Comments: //complete, working program

Header
main()
{
    float ke;
    float V = 100.0;
    ke = 1.6E-16 * V;
}

Function Example: KeV (3)

Prototype: float KeV( float V);

Header: float KeV( float V)
{
    float ke;
    ke = 1.6E-16 * V;
    return( ke );
}

Function Example: KeV (4)
“Complete*” Program #2

Prototype: float KeV( float V);

Header: float KeV( float V)
{
    float fke;
    float fv = 123.0;
    ke = KeV( fv );
}

Function Call: float KeV( float V)
{
    float ke; /*due to space constraints the rest of the function is not shown; see the previous slide!*/

Function Example: KeV (5)

- Function Conclusions:
  - Both programs execute the same calculations but the second version is more "readable."
  - Portable / Reusable
  - Global / Local Variables