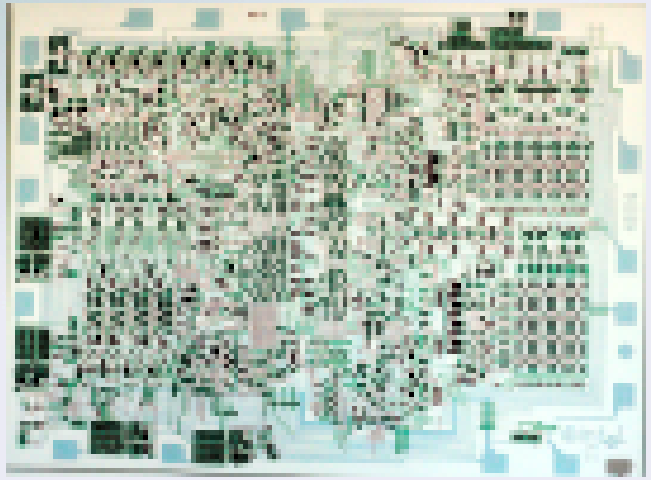

Lecture 12b: Programming in The C Language

CSIS11: **Computer Architecture and Organization**

Readings

- *Chapters 11-12* Patt and Patel: Introduction to Computing Systems...
- *The C Language* - Kernighan and Ritchie - Second Edition, strongly recommended
- Beej's Guide to C Programming (Tutorial) - Instructor Repo
- Beej's Guide to C Programming (Library Reference) - Instructor Repo

Grade Update

- All grades have been posted.
- Graded assignments are:
  - Lab 2, Lab 3 (upto 15 points)
  - Assignments 6, 7, 8, 9 (SMC)
  - Test 1 and Test 2
  - SSH Extra Credit

Final Project

- **Finish SMC -** 100%, must put result on Stack, must provide both quotient and remainder
- **Bonus up to 25% -** Have SMC execute for any legal number of inputs and operators, must be final result on stack as well as print to console. Must provide both quotient and remainder for division. Must provide error messages for overflow, underflow and divide by zero.
- **Bonus up to 50% -** Hardest grading, includes previous bonus and adds a user manual, comprehensive commenting and look like a professional finished project on Github

Course Requirements (Scoring)

Item	Number	Points @	Extra Credit	Weighting
Homework	8	10	10	25%
Tests	2	100	None	40%
Final Project	1	50	25	10%
Final Exam	1	200	None	25%

Today

- C Control Structures
- Guide to Writing Problems
- Homework Assignment 12

Control Structures

- **Conditional**
  - Making decision about which code to execute, based on evaluated expression
  - `if`
  - `if-else`
  - `if-else if-else`
- **Iteration**
  - Executing code multiple times, ending based on evaluated expression
  - `while`
  - `for`

If

```
if (condition)
  action;
```

- Condition is a C expression, which evaluates to TRUE (non-zero) or FALSE (zero).
- Action is a C statement, which may be simple or compound (a block).

Example If Statements

```
// legal, though problematic, don't use
if (x <= 10)
  y = x * x + 5;

// always use {}, thank me, later
if (x <= 10)
{
  y = x * x + 5;
  z = (2 * y) / 3;
}
```

Generating Code for If Statement

```
if (x == 2)
{
  y = 5;
}
```

```
LDR  R0, R6, #0  ; load x into R0
ADD  R0, R0, #-2 ; subtract 2
BRnp NOT_TRUE    ; if non-zero, x is not 2
AND  R1, R1, #0  ; store 5 to y
ADD  R1, R1, #5
STR  R1, R6, #1
NOT_TRUE ...             ; next statement
```

If-else

```
if (condition)
  action_if;
else
  action_else;
```

- Else allows choice between two mutually exclusive actions without re-testing condition.

Ultimate form of if-else...

```
if (comparison)
{
    code;
}
else if (comparison)
{
    code;
}
else
{
    code;
}
```

While

```
while (test)
  loop_body;
```

- Executes loop body as long as test evaluates to TRUE (non-zero)
- Note: Test is evaluated before executing loop body

Generating Code for While

```
x = 0;
while (x < 10)
{
  printf("%d ", x);
  x = x + 1;
}
```

Infinite Loops

```
x = 0;
while (x < 10)
{
  printf("%d ", x);
}
```

- Loop body does not change condition...
- ...so test is never false
- Common programming error that can be difficult to find

For

```
for (init; end-test; step)
  statement
```

- Executes loop body as long as test evaluates to TRUE (non-zero).
- Initialization and step code included in loop statement.
- Note: Test is evaluated before executing loop body

Example For Loops

```
/* -- what is the output of this loop? -- */
for (i = 0; i <= 10; i++)
{
  printf("%d ", i);
}

/* -- what does this one output? -- */
letter = 'a';
for (c = 0; c < 26; c++)
{
  printf("%c ", letter+c);
}
```

Nested Loops

- Loop body can (of course) be another loop

```
/* print a multiplication table */
for (mp1 = 0; mp1 < 10; mp1++)
{
  for (mp2 = 0; mp2 < 10; mp2++)
{
    printf("%d\t", mp1*mp2);
  }
  printf("\n");
}
```

Another Nested Loop

- Here, test for the inner loop depends on counter variable of outer loop

```
for (outer = 1; outer <= input; outer++)
{
  for (inner = 0; inner < outer; inner++)
{
    sum += inner;
  }
}
```

For vs. While

- In general:
  - For loop is preferred for counter-based loops
    - Explicit counter variable
    - Easy to see how counter is modified each loop
  - While loop is preferred for sentinel-based loops
    - Test checks for sentinel value.
- Either kind of loop can be expressed as other, so really a matter of style and readability

Break and Continue

- `break;`
  - used only in *switch* statement or *iteration* statement
  - passes control out of the "nearest" (loop or switch) statement containing it to the statement immediately following
  - usually used to exit a loop before terminating condition occurs (or to exit switch statement when case is done)
- `continue;`
  - used only in *iteration* statement
  - terminates the execution of the loop body for **this iteration**
  - loop expression is evaluated to see whether another iteration should be performed
  - if for loop, also executes the step action

Quick Guide to Developing C Programs

- Rules for Success
- Template for C Program
- Variables
- Input and Output
- Control

Rules for Success

1. **End each statement with a semicolon (;)**
2. **Include necessary header files** (`#include <stdio.h>` for input/output functions)
3. **Every C program needs a main() function**
4. **Check your braces { }**, ensure opening and closing braces are properly paired
5. **Initialize variables before using them**
6. **Use proper data types**, match variable types with their intended use
7. **Add comments to explain your code**, use `//` for ALL comments

Template for a C Program

```c
#include <stdio.h>

// Function: main
// Description: counts down from user input to STOP
void main()
{
    // variable declarations

    // prompt user for input

    // perform task

    // print message

}
```

Variables

- **Have to be declared** prior to using
- Keep scope in mind

```
// Integers
int myIntegerVariable

// Characters (not character string)
char myCharacterVariable

// Float
float myFloatVariable
```

Input

```
// simple input
scanf("%d\n", startPoint);

// multiple input, separate entries with a blank
scanf("%d %d", startPoint, counter);
```

Formatting options:

- `%d` - decimal integer
- `%c` - ASCII character
- `%f` - floating-point number

Output

```
// simple print
printf("%d\n", startPoint);

// print expressions, not just variables
printf("%d\n", startPoint - counter);

// print multiple expressions with a single statement
printf("%d %d %d\n", counter, startPoint, counter);
```

Formatting options:

- `%d` - decimal integer
- `%c` - ASCII character
- `%f` - floating-point number

Control Structures

- **Conditional**
  - Making decision about which code to execute, based on evaluated expression
  - `if`
  - `if-else`
  - `if-else if-else`
- **Iteration**
  - Executing code multiple times, ending based on evaluated expression
  - `while`
  - `for`

Time for an Assignment

1. Fetch/merge Instructor repo to your system
2. Create a C folder at the same level as assembly
3. Copy week_12 over from Instructor to Student