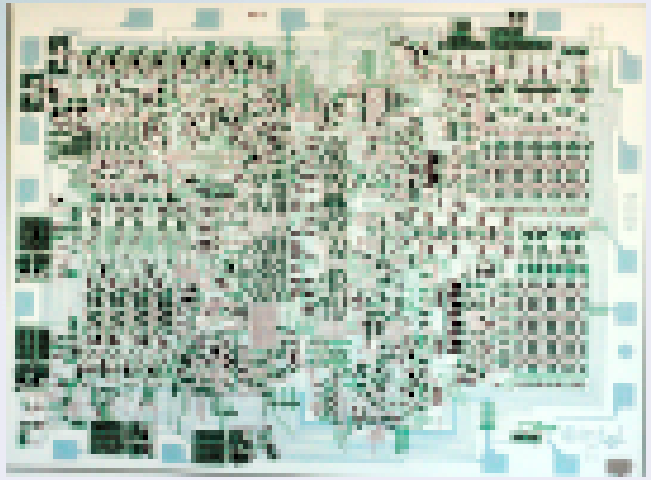

Lecture 13a: Programming in The C Language: Storage and Functions

CSIS11: **Computer Architecture and Organization**

Readings

- *Chapters 14* 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

Remaining Schedule

Day	Content	Chapter(s)
4/21	C: Introduction, Variables and Operators	11/12
4/23	C: Control Structures	13
4/28	**C: Storage and Functions**	**14**
4/30	C: Testing and Debugging	15
5/5	C: Pointers, Arrays and Structs	16
5/7	C: I/O	18
5/12	Review and Work on Final Project	
5/14	Review and Work on Final Project	
5/21	**Final Exam – Wed, May 21, 2025 6-8:30 PM**	

Implementing Storage in LC-3

Main

- Variables
  - Local
  - Global

Functions

- Parameter passing
- Return values

Allocating Space for Variables

Global data section

- All global variables stored here
- **R4** points to beginning

Run-time stack

- Used for local variables
- **R6** points to top of stack
- New (stack) frame for each block (goes away when block exited)

Offset = distance from beginning of storage area

- Global: LDR R1, R4, #4
- Local: LDR R2, R6, #3

Local Variable Storage

Local variables stored in stack frame

Symbol table "offset" gives the distance from the base of the frame

- A new frame is pushed on the run-time stack each time block is entered
- R6 is the stack pointer – holds address of current top of run-time stack

Symbol Table Example

```
int main() {
    int seconds;
    int minutes;
    int hours;
    double amount;
    }
```

Name	Type	Offset	Scope
amount	double	0	main
hours	int	2	main
minutes	int	3	main
seconds	int	4	main

Example: *C* Compiling to LC-3

```c
#include <stdio.h>
int X;
main() {
   int a;
   int A;
   int B;
   /* initialize */
   a = 5;
   X = 3;
   /* perform calculations */
   A = a - X;
   B = (a + X) + A;
   /* print results */
   printf("The results are: A = %d, B = %d\n",
          A, B);
}
```

Symbol Table

Name	Type	Offset	Scope
X	int	0	global
a	int	2	main
A	int	1	main
B	int	0	main

Example: Code Generation

```
; R6 contains locals, R4 contains Globals
; main
; a = 5
    AND R0, R0, #0
    ADD R0, R0, #5  ; a = 5
    STR R0, R6, #2  ; (offset = 2)
; X = 3
    AND R0, R0, #0
    ADD R0, R0, #3  ; X = 3
    STR R0, R4, #0  ; (offset = 0)
```

Example (continued)

```
; R6 contains locals, R4 contains Globals
; first statement:
; A = a - X;
   LDR R0, R6, #2  ; get a(offset = 2)
   LDR R1, R4, #0  ; get X
   NOT R1, R1      ; ~X
   ADD R1, R1, 1   : -X
   ADD R2, R0, R1  ; a - X
   STR R2, R6, #1  ; store in A, (offset = 1)
```

Example (continued)

```
; R6 contains locals, R4 contains Globals
;B = (a + X) + A;
     LDR R0, R6, #2  ; a
     LDR R1, R4, #0  ; X
     ADD R0, R0, R1  ; R0 is sum
     LDR R1, R6, #1  ; A
     ADD R2, R0, R1  ; R2 is sum
     STR R2, R6, #0  ; B (offset = 0)
```

Run example.asm in LC-3 Tools

Fundamental Ideas in Programming

- Abstraction - simplify a series of actions
- Encapsulation - keep the data specific to the task

Starting a Gas-Powered Car: Step-by-Step Process

1. **Insert the key into the ignition switch** and turn it to the "ON" position
    - This activates the car's electrical system, powering up the dashboard lights and accessories
2. **Turn the key to the "START" position**
    - This **closes the ignition circuit** and sends electrical current from the battery to the starter solenoid
3. **The starter solenoid activates**
    - It **pushes the starter drive gear forward** to engage with the flywheel/flexplate
    - It **closes heavy contacts** that connect the battery to the starter motor

4. **The starter motor engages and spins**
    - The **pinion gear meshes with the flywheel ring gear** attached to the engine's crankshaft
    - This provides the initial mechanical energy to **rotate the crankshaft**
5. **The crankshaft rotation begins the four-stroke cycle**
    - As the pistons move, they create the intake, compression, power, and exhaust strokes
6. **The engine computer (ECU) activates the fuel system**
    - The **fuel pump pressurizes fuel** from the tank to the engine
    - **Fuel injectors spray precise amounts of fuel** into the intake manifold or directly into cylinders

7. **The ignition system produces sparks**
    - The **spark plugs fire** in the correct sequence
    - This **ignites the compressed air-fuel mixture** in each cylinder
8. **Combustion begins and becomes self-sustaining**
    - Once enough cylinders fire, the engine generates its own momentum
    - When the engine reaches sufficient RPM, you can **release the key** from the start position
9. **The starter motor automatically disengages**
    - A **one-way clutch** in the starter drive prevents damage from the now-faster spinning flywheel
    - The starter motor stops when you release the key

Solving problems

- Each of the 9 steps could be a function in a program
- Each step is a specific task which might spawn sub-tasks
- With each step having its own set of data

Function

- Smaller, simpler, subcomponent of program
- Provides abstraction
  - Hide low-level details
  - Give high-level structure to program, easier to understand overall program flow
  - Enables separable, independent development
- C functions
  - Zero or multiple arguments (or parameters) passed in
  - Single result returned (optional)
  - Return value is always a particular type
- In other languages, called procedures, subroutines, ...

Example of High-Level Structure

```
void main()
{
  draw_game();  /* draw simple grid */
  determine_side(); /* choose X for X & Z for 0 */

  /* Play game */
  while (no_outcome_yet())
  {
    X_turn();
    Z_turn();
  }
}
```

Structure of Tic-Tac-Toe program is evident, even without knowing implementation.

Functions in C

Definition

```
int factorial(int n)
{
  int i;
  int result = 1;
  for (i = 1; i <= n; i++) {
    result = result * i;
  }
  return result;
}
```

Function call -- used in expression

```
a = x + factorial(f + g);
```

Functions in C

Definition

Type of return value

Type of all arguments

```
int factorial(int n)
{
  int i;
  int result = 1;
  for (i = 1; i <= n; i++) {
    result = result * i;
  }
  return result;
}
```

Name of function

exits function with specified return value

3. Use return value in expression

2. Execute function

1. Evaluate arguments

Function call -- used in expression

```
a = x + factorial(f + g);
```

Function Requirements

1. Function must be **declared** before using
2. Function may be **defined** at a later time
3. Functions must have:
    i. Type of return value or void
    ii. Type of all arguments or void
    iii. Unambiguous name and not *main*

Function Development

In developing a C program, the typical process is to do the following:

1. Have a single file for *main*, the core program
2. Group all function *declarations* in a header file, *xxxx.h*
3. Group all *definitions* in a program file(s), xxxx.c

Sometimes, both the header files and definition files will be in a *local* library

C Function Examples

1. Function with No Arguments and No Return Value
2. Function with Arguments but No Return Value
3. Function with Arguments and Return Value

With best practices of declaration then definition

```c
// Function with No Arguments and No Return Value
#include <stdio.h>

// Function declaration
void error(void);

int main()
{
   // ...code...
   // Code prior to this function, has a un-resolvable error
   // Error Function call
    error();
    return 1;
}

// Function definition
void error(void) {
    printf("An error has occurred, there is nothing we can do.\n");
}
```

```c
// Function with Arguments but No Return Value
#include <stdio.h>

// Function declaration
void printSum(int a, int b);

int main() {
    int x = 5, y = 7;
    // Function call with arguments
    printSum(x, y);
    return 0;
}

// Function definition
void printSum(int a, int b) {
    printf("The sum of %d and %d is %d\n", a, b, a + b);
}
```

```c
//Function with Arguments and Return Value
#include <stdio.h>

// Function declaration
int calculateArea(int length, int width);

int main() {
    int l = 10, w = 5;
    // Function call with arguments that returns a value
    int area = calculateArea(l, w);
    printf("The area of the rectangle is: %d square units\n", area);
    return 0;
}

// Function definition
int calculateArea(int length, int width) {
    return length * width;
}
```

For Homework, see the Instructor repository for C/Week_13