University of Illinois at Urbana-Champaign
Dept. of Electrical and Computer Engineering

ECE 220: Computer Systems & Programming

Designing Loops

An Approach for Designing Control of Iterations

Want to provide you with
- a **step-by-step process**
- for **designing control for iterations**.

After discussing the steps,
- we'll walk through an example
- adapted from the LC-3 simulator.

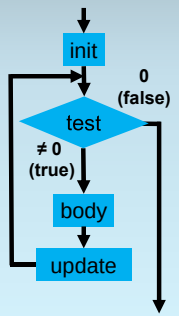
Finally, we'll do a think-pair-share.

Before Starting, Be Sure that You Know Why

0. What is the task that you're repeating?

Be sure that the answer is clear to you before you start.

Otherwise, why are you iterating?

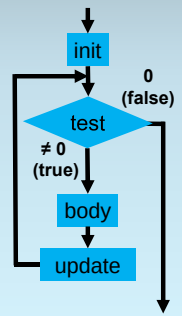


What Can You Assume When You Start an Iteration?

0. What is the task that you're repeating?
1. What is true at the start of "test" in each iteration?

You can make up variables and assumptions (called **invariants**).

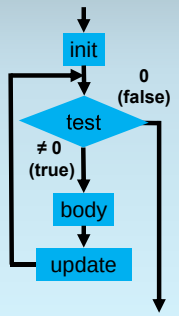
But you have to make them hold true in later steps.



Identify All Reasons for Stopping the Iteration

0. What is the task that you're repeating?
1. What is true at the start of "test" in each iteration?
2. When does the iteration stop (what is "test")?

You may have more than one answer.



Examples of Multiple Stopping Conditions

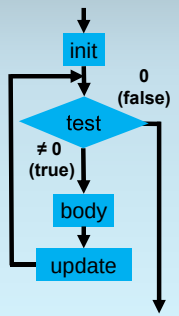
Type a number using the keyboard.
- Stop when **user presses <Enter>**.
- Stop when **user presses a non-digit**.
- Stop when **number overflows**.

Find the first letter 'A' in a string.
- Stop when **first 'A' is found**.
- Stop at **end of string**.

What Happens When the Iteration Stops?

3. What should be done when iteration stops?

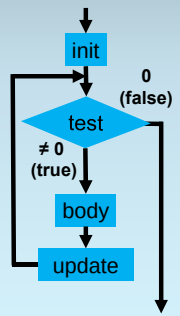
Answer may be different for different stopping conditions.



Set Up for the First Iteration with Init

3. What should be done when iteration stops?
4. How do you set up for the loop (what is "init")?

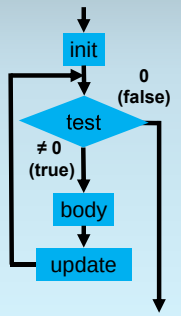
Initialization must ensure that invariants hold for first iteration.



Set Up for the Later Iterations with Update

3. What should be done when iteration stops?
4. How do you set up for the loop (what is "init")?
5. How do you update between iterations (what is "update")?

Update must ensure that invariants hold for next iteration.



Example: Dump LC-3 Memory

Let's do an example. In the LC-3 simulator,
- one can **examine the contents of memory**
- using the **dump** command.

Recall that LC-3 memory uses
- **16-bit addresses** with
- **16-bit addressable** memory.
- For each, we can use **four hex digits**.

Formatting for Dump Command Output

Look at a sample of the output…

address of first location on line

twelve locations per line

```
01F8:                                   E002 F022 F025 000A
0204: 0057 0065 006C 0063 006F 006D 0065 0020 0074 006F 0020 0074
0210: 0068 0065 0020 004C 0043 002D 0033 0020 0073 0069 006D 0075
021C: 006C 0061 0074 006F 0072
```

contents of a memory location

each address is a multiple of 12

contents not requested for these addresses

Function Signatures for Dumping and Reading Memory

What information is needed for this output?
- Starting address and
- ending address.

```
void dump_memory
    (int addr_s, int addr_e);
```

We also need access to LC-3 memory contents:

```
int read_memory (int addr);
```


Allow Dumped Memory Region to Wrap Around

We're almost ready to iterate.

But we have a problem:
- what **if a caller specifies**
- **0xF000 through 0x1000** (**`addr_e < addr_s`**)?

Do we
- refuse (return an error)?
- Or wrap around?

Let's **wrap around**.

Can We Use Just One Iteration?

Notice that
- we end just before **`addr_e`**.
- (So using the same address shows all of memory.)

How can we iterate over addresses in the case shown?

Do we need two iterations for the two yellow regions?

Leverage 32-bit Integer to Create a Virtual Copy

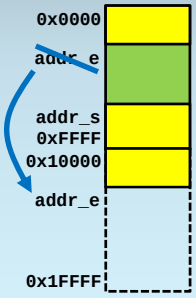
Can we make it simple?

What if we add 0x10000 to addr_e?

Now we can use our loop "address" AND'd with 0xFFFF.

Effectively, we have a virtual copy of the address space.

Our **loop "address" just goes up!**



Let's Design the First Iteration

Now it's time to iterate.

The output format appears below.

```
01F8:                                 E002 F022 F025 000A
0204: 0057 0065 006C 0063 006F 006D 0065 0020 0074 006F 0020 0074
0210: 0068 0065 0020 004C 0043 002D 0033 0020 0073 0069 006D 0075
021C: 006C 0061 0074 006F 0072
```

0. What is the task that you're repeating?

Print one row.

Need to Make Up Variables for Our Assumptions

```
01F8:                                   E002 F022 F025 000A
0204: 0057 0065 006C 0063 006F 006D 0065 0020 0074 006F 0020 0074
0210: 0068 0065 0020 004C 0043 002D 0033 0020 0073 0069 006D 0075
021C: 006C 0061 0074 006F 0072
```

1. What is true at the start of "test" in each iteration?
   - **a. `start` holds first address for line.**
   - **b. `start` is a multiple of 12.**

We have to make these invariants hold true!

Only Print Requested Addresses

```
01F8:                                   E002 F022 F025 000A
0204: 0057 0065 006C 0063 006F 006D 0065 0020 0074 006F 0020 0074
0210: 0068 0065 0020 004C 0043 002D 0033 0020 0073 0069 006D 0075
021C: 006C 0061 0074 006F 0072
```

2. When does the iteration stop (what is "test")?

   Stop when **`start >= addr_e`**.

Recall that we do not print the contents of **`addr_e`**.

Nothing to Do After the Iteration Finishes

```
01F8:                                   E002 F022 F025 000A
0204: 0057 0065 006C 0063 006F 006D 0065 0020 0074 006F 0020 0074
0210: 0068 0065 0020 004C 0043 002D 0033 0020 0073 0069 006D 0075
021C: 006C 0061 0074 006F 0072
```

3. What should be done when iteration stops?

   Nothing.

When the iteration finishes, the function is done.

`start` Should be the Largest Multiple of 12 `<= addr_s`

```
01F8:                                   E002 F022 F025 000A
0204: 0057 0065 006C 0063 006F 006D 0065 0020 0074 006F 0020 0074
0210: 0068 0065 0020 004C 0043 002D 0033 0020 0073 0069 006D 0075
021C: 006C 0061 0074 006F 0072
```

4. How do you set up for the loop (what is "init")?

```
start = (addr_s / 12) * 12
```

We need **`start`** to be **`<= addr_s`**, and a multiple of 12.

`start` Should be a Multiple of 12

```
01F8:                                   E002 F022 F025 000A
0204: 0057 0065 006C 0063 006F 006D 0065 0020 0074 006F 0020 0074
0210: 0068 0065 0020 004C 0043 002D 0033 0020 0073 0069 006D 0075
021C: 006C 0061 0074 006F 0072
```

5. How do you update between iterations (what is "update")?

```
start = start + 12
```

We need **`start`** to be first address for next line, and a multiple of 12.

Our Function So Far…

```
void dump_memory (int addr_s, int addr_e)
{
    int start;
    if (addr_s >= addr_e) {
        addr_e += 0x10000;
    }
    for (start = (addr_s / 12) * 12;
         start < addr_e; start = start + 12) {
        // print one row
    }
}
```

We need another iteration.

Two Steps for Printing a Row

```
01F8:                                   E002 F022 F025 000A
0204: 0057 0065 006C 0063 006F 006D 0065 0020 0074 006F 0020 0074
0210: 0068 0065 0020 004C 0043 002D 0033 0020 0073 0069 006D 0075
021C: 006C 0061 0074 006F 0072
```

Is printing a row just an iteration?

No! It's a sequence:
- **print the address**, then
- **print contents** of 12 memory locations.

How Do We Print the Row Address?

Where is the address for the row?

`start`

How can we print the address?

`printf ("%04X: ", start & 0xFFFF);`

Why mask out the high bits?

Remember that start may point into the virtual copy (`start` > 0x10000).

Use Additional Formatting Features for Nice Output

```
printf ("%04X: ", start & 0xFFFF);
```

What does "04" mean?

4 is the **field width**
- printf outputs **at least 4 characters** and
- **right-justifies** the output
- with **leading spaces**

0 means **use leading zeroes** instead

Let's Design the Second Iteration (Inner Loop)

Now it's time to iterate again.

The output format appears below.

```
01F8:                                   E002 F022 F025 000A
0204: 0057 0065 006C 0063 006F 006D 0065 0020 0074 006F 0020 0074
0210: 0068 0065 0020 004C 0043 002D 0033 0020 0073 0069 006D 0075
021C: 006C 0061 0074 006F 0072
```

0. What is the task that you're repeating?

Print one memory location.

Need to Make Up Variables for Our Assumptions

```
01F8:                                   E002 F022 F025 000A
0204: 0057 0065 006C 0063 006F 006D 0065 0020 0074 006F 0020 0074
0210: 0068 0065 0020 004C 0043 002D 0033 0020 0073 0069 006D 0075
021C: 006C 0061 0074 006F 0072
```

1. What is true at the start of "test" in each iteration?
   a. **`addr` holds address of the memory location to print.**
   b. **`index` is the line index (0 to 11).**

We have to make these invariants hold true!

Stop After Printing 12 Locations

```
01F8:                                   E002 F022 F025 000A
0204: 0057 0065 006C 0063 006F 006D 0065 0020 0074 006F 0020 0074
0210: 0068 0065 0020 004C 0043 002D 0033 0020 0073 0069 006D 0075
021C: 006C 0061 0074 006F 0072
```

2. When does the iteration stop (what is "test")?

Stop when **`index >= 12`**.

Using **`index`** makes the test easier.

Add a Line Feed After the Iteration Finishes

```
01F8:                                   E002 F022 F025 000A
0204: 0057 0065 006C 0063 006F 006D 0065 0020 0074 006F 0020 0074
0210: 0068 0065 0020 004C 0043 002D 0033 0020 0073 0069 006D 0075
021C: 006C 0061 0074 006F 0072
```

3. What should be done when iteration stops?

Print a line feed character.

When the iteration finishes, end the printed line.

Initialize Both `index` and `addr`

```
01F8:                                   E002 F022 F025 000A
0204: 0057 0065 006C 0063 006F 006D 0065 0020 0074 006F 0020 0074
0210: 0068 0065 0020 004C 0043 002D 0033 0020 0073 0069 006D 0075
021C: 006C 0061 0074 006F 0072
```

4. How do you set up for the loop (what is "init")?

```
index = 0, addr = start
```

Comma operator allows multiple initializations.

Update Both `index` and `addr`

```
01F8:                                   E002 F022 F025 000A
0204: 0057 0065 006C 0063 006F 006D 0065 0020 0074 006F 0020 0074
0210: 0068 0065 0020 004C 0043 002D 0033 0020 0073 0069 006D 0075
021C: 006C 0061 0074 006F 0072
```

5. How do you update between iterations (what is "update")?

```
index++, addr++
```

That's it for the loop control!

Print One Location's Contents or Spaces

How do we print one location?

```
if (addr >= addr_s &&
    addr < addr_e) {
    printf ("%04X ", read_memory
            (addr & 0xFFFF));
} else {
    printf ("     ");
}
```

Location contents desired?

Use function to read memory.

Mask out high bits.

The Web Page Has the Code

That's it!

The code is on the web page.

Time for Another Think-Pair-Share

As before, let's do a group exercise in lecture.

The process:

1. I give you a problem.
2. You form groups of 3-4 people.
3. Talk about ways to solve the problem.
4. Once enough of the groups have finished, one group volunteers to share their answer.
5. We go over the group's answer together.

The Task: Print a Hollow Square of Asterisks

Write **C** function **print_square**
- to print a square of asterisks
- filled with spaces
- of a certain size.

```
*****
*   *
*   *
*   *
*****
```

Assumptions and rules:
- Size should be given by caller.
- Define the meaning of the return value (if any).
- Arguments should be checked (you decide how).
- Use loops to print the square.

