What is a program?
- What is a program?
- Set of instructions written in a programming language that tells the computer what to do

The programming process
- Defining the problem
- Planning the solution
- Describing the algorithm
  - Pseudo-code
  - Flow chart
- Coding the program
- Compiling and testing the program
- Documenting the program

The programming process: Defining the problem
- What is the input
- What output do you expect
- How do you get from the input to the output

The programming process: Planning the solution
- Algorithm: detailed solutions to a given problem.
- The algorithm must:
  - Be unambiguous
  - Always terminate in a finite number of instructions
  - (Possibly!) solve the problem

Describing the algorithm: Pseudo-code

- Summing all integers to 100
- 1. Initialize counter to 1
- 2. Initialize summing variable to 0
- 3. Add counter to summing variable
- 4. Is counter equal to 100 yet?
Describing the algorithm: Pseudo-code

The algorithm has logic structure, but no language syntax.


Describing the algorithm: Flow chart

The programming process: Coding the program

- Translate algorithm into a formal programming language
- Within syntax of the language
- How to key in the statements?
  - Text editor
  - Programming environment
    - Interactive Development Environment (IDE)

The programming process: Compiling and testing the program

- Compiler
  - Translates from source module into object module
  - Finds syntax errors
- Linker
  - Combines object modules with libraries to create load module
  - Finds undefined external references
- Debugger
  - Execute program in a controlled environment
  - Run using data that tests all statements
  - Finds logic errors

The programming process: Documenting the program

- Much better when performed throughout the development!
- Material generated during each step
  - Problem definitions
  - Program plan
  - Comments within source code
  - Testing of procedures
  - Narrative
  - Layouts of input and output
  - Program listing
**Programming Languages**
- **1st Generation**: Machine Level
- **2nd Generation**: Assembly Level
- **3rd Generation**: High-level (C, Fortran, Pascal, BASIC, COBOL, C++, Java)
- **4th Generation**: Application-specific (SQL, MATLAB)
- **5th Generation**: Declarative languages (Prolog, regex)

**FORTRAN**
- FORTRAN = Formula Translator
- IBM custom language
- Developed between 1954 and 1966

**COBOL**
- COBOL = Common Business-Oriented Language
- Defined in 1960

**BASIC**
- BASIC = Beginners, All-purpose Symbolic Instruction Code
- Born in 1963

**C++**
- C++, or “C with classes”
- Born in 1983

**Programming constructs**

```
<table>
<thead>
<tr>
<th></th>
<th>Sequential</th>
<th>Conditional</th>
<th>Iterative</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block A</td>
<td>T</td>
<td>F</td>
<td></td>
</tr>
<tr>
<td>Block B</td>
<td>cond</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Block A</td>
<td></td>
<td>Block B</td>
<td></td>
</tr>
<tr>
<td>Block B</td>
<td></td>
<td>Block A</td>
<td></td>
</tr>
<tr>
<td>Block A</td>
<td></td>
<td></td>
<td>Block B</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
**IF-THEN-ELSE**

**In C**

```c
if (count < 0)
    count = count + 1;
```

**In LC-3**

```assembly
LD R0, count
BRpz endif
ADD R0, R0, #1
endif ; next inst. goes here
```

**WHILE-DO**

**In C**

```c```
while (count > 0)
{
    a = a + count;
    count--;
}
```

**In LC-3**

```assembly
LD R1, a
LD R0, count
while BRnz endwhile
ADD R1, R1, R0
ADD R0, R0, #-1
BR while
endwhile ST R1, a
ST R0, count
```

**DO-WHILE**

**In C**

```c```
do
{
    if (a < b)
        a++;
    if (a > b)
        a--;
} while (a != b)
```

**In LC-3**

```assembly
LD R0, a
LD R1, b
JSR Sub
repeat BRpz secondif
ADD R0, R0, #1
JSR Sub
secondif BRnz until
ADD R0, R0, #-1
until JSR Sub
BRnp repeat
```

**IF-THEN-ELSE**

**In C**

```c```
if (count < 0)
    count = count + 1;
```

**In LC-3**

```assembly
LD R0, count
BRpz endif
ADD R0, R0, #1
endif ; next inst. goes here
```

**WHILE-DO**

**In C**

```c```
while (count > 0)
{
    a = a + count;
    count--;
}
```

**In LC-3**

```assembly
LD R1, a
LD R0, count
while BRnz endwhile
ADD R1, R1, R0
ADD R0, R0, #-1
BR while
Endwhile: ST R1, a
ST R0, count
```

**DO-WHILE**

**In C**

```c```
do
{
    if (a < b)
        a++;
    if (a > b)
        a--;
} while (a != b)
```

**In LC-3**

```assembly
LD R0, a
LD R1, b
JSR Sub
repeat: BRpz secondif
ADD R0, R0, #1
JSR Sub
secondif: BRnz until
ADD R0, R0, #-1
until: JSR Sub
BRnp repeat```
```c
In C

```for (i = 3; i <= 8; i++)
{
    a = a + i;
}
```n

```c
In LC-3

```LD R0, a
AND R1, R1, #0
ADD R1, R1, #3
```n
```for ADD R2, R1, #-8
```n
```BRp endfor
```n
```ADD R0, R0, R1
ADD R1, R1, #1
```n
```BR for
```n
```endfor
```n

---

**Recommended exercises**

- Ex 6.4, 6.5, 6.7, 6.8
- Ex 6.11 write the assembly code and simulate
- Ex 6.12 write code and simulate
- Ex 6.14, 6.15, 6.16, 6.17, 6.19, all really good inverse problems
- Ex 6.13
- Ex 6.18 is good, but kind of challenging...