INF-1100
Algorithms, control flow, and variables

Åge Kvalnes

University of Tromsø, Norway

September 17, 2014

How do we program a computer?

In general, it's all about a systematic transformation of a problem between layers of abstraction:

1. Problem: A definition of the problem in natural language.
2. Algorithm: A precise description of a sequence of steps that when followed solves the problem. Constraints on the algorithm:
   - Unambiguous: Each step is precisely described. It should not be possible to interpret a step in different ways.
   - Terminate: After completing all steps of the algorithm the problem should be solved.
   - Computable: Each step of the algorithm must involve actions that are actually computable by a computer.

How do we program a computer?

3. Program: A program expressed in a computer language that carries out the different steps of the algorithm. Over 1000 different computer languages available.
4. Instruction Set Architecture (ISA): The program must be translated into instructions that the computer can perform.
5. Microarchitecture: Implementation of the ISA at the hardware level.
6. Logic circuits: Basic building blocks are combined to implement elements of the microarchitecture. Many different ways to implement different types of basic functionality (such as addition/division).
7. Devices: Each logic circuit building block is constructed using tiny electronic devices (such as CMOS or NMOS transistors). The speed and properties (e.g. power consumption) may vary depending on the materials used for construction.

Constructing algorithms: Stepwise refinement

Going from a problem description to an algorithm involves a process known as *systematic decomposition*, or *stepwise refinement*.

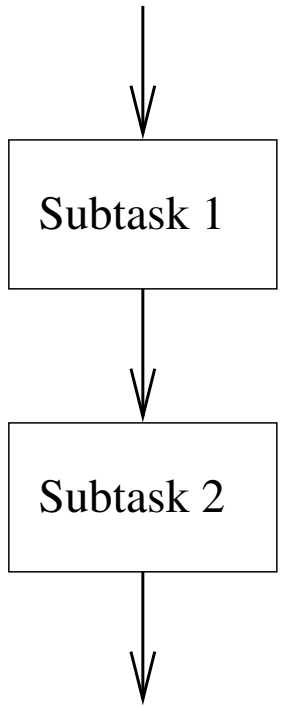
The problem is decomposed into smaller subtasks that in turn are decomposed into even smaller subtasks, etc. In the end you are at the machine instruction level.

There are essentially three ways to decompose a task:

- Sequential.
- Conditional.
- Iterative.

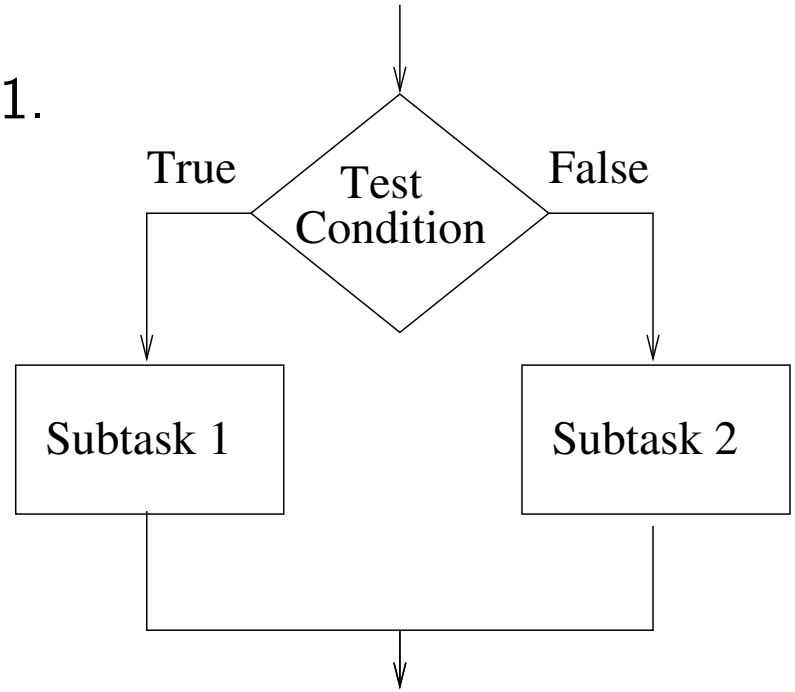
Decomposing tasks: The *sequential* construct

- A series of subtasks that are performed sequentially.
- Each subtask is only performed once.
- Each subtask can be broken down into two new subtasks etc.



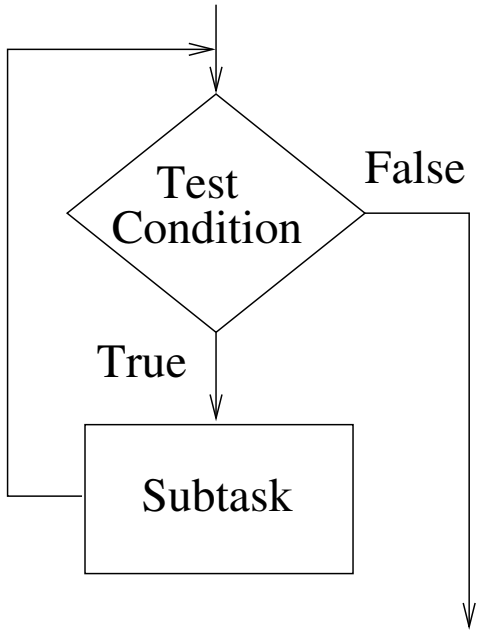
Decomposing tasks: The *conditional* construct

- If condition is true do subtask 1.
  If not, do subtask 2.



Decomposing tasks: The *iterative* construct

- Do a subtask as long as a condition holds.



Mapping the constructs to C

Sequential:

- A series of expressions, formed from operators and operands.
- Expressions typically perform arithmetic and logical operations on fundamental data types (characters, integers, and floating point numbers).
- A semi-colon (;) delimits the expression.

```
x = x*10 + 100;
x = x & 0xff;
```

Mapping the constructs to C

Conditional:

- **if/else** statements.
- Curly braces ({ and }) delimit and group what to conditionally perform.

```
if (x < 10) {
   y = y*10;
} else {
   y = y*2;
}
```

Mapping the constructs to C

Iterative:

- **while** statements.
  - Execute some expression while a condition is true.
- Curly braces ({ and }) delimit and group what should be iterated.

```
y = 0;
x = 0;
while (x < 10 {
   y = y + x;
   x = x + 1;
}
```

Mapping the constructs to C

Iterative:

- **for** statements.
  - Execute some expression *before the first iteration*.
  - Execute expression that determines whether to stop iteration.
  - Execute some expression after each iteration.
- Curly braces ({ and }) delimit and group what should be iterated.

```
y = 0;
for (x = 0; x < 10; x = x + 1) {
   y = y + x;
}
```

Number guessing game

Problem: Program picks a number between 0 and 99 and asks the user to guess the number.

Algorithm:

1. Pick a random number.
2. Input number from user.
3. If user typed correct number, stop program.
4. Inform user whether the program's number is higher or lower and goto step 2.

Number guessing game II

Problem: User picks a number between 0 and 99 and the program should guess the number.

Algorithm:

1. Guess the number.
2. Ask user whether the guess was correct or if the user's number is higher or lower.
3. If user's number was found, stop program. If not, goto step 1.

Number guessing game II: Guessing the smart way

Algorithm:

1. Set the search space to 0..99.
2. Guess the number in the middle of the search space.
3. If user's number found, stop program.
4. If user's number is higher, set lower range of search space to guessed number.
5. If user's number is lower, set higher range of search space to guessed number.
6. Goto step 2.

How many guesses are needed to find the secret number?