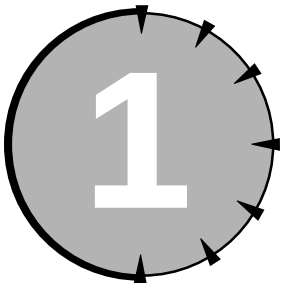
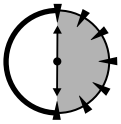
SESSION



What Is Programming Anyway?

Session Checklist

- ✔ Learn the principles of programming
- ✔ Learn to be a human computer processor
- ✔ Learn to change a tire



30 Min. To Go

Webster's New World College Dictionary lists several definitions for the noun "program." The first definition is "a proclamation, a prospectus, or a syllabus." Not much help there. It wasn't until the sixth definition that I found something reasonable: "a logical sequence of coded instructions specifying the operations to be performed by a computer in solving a problem or in processing data."

After thinking for a minute, I realized that this definition is a bit restrictive. First, in the phrase "a logical sequence of coded instructions . . . ," I don't know whether the instructions are encrypted, that is coded, or not, but I know the term "logical" is overly restrictive. I have written programs that don't do much of anything before crashing — actually, most of my programs crash before doing anything. That doesn't seem logical. Second, ". . . solving a problem or in processing data." What about the computer that drives the climate control system in my car?

It doesn't solve any problem that I'm aware of. I like my air conditioner the way it is — push a button to turn it on, push the button again to turn it off.

The biggest problem with Webster's definition is the phrase ". . . operations to be performed by a computer . . ." A program does not need to involve a computer at all. (Unless you count that muddy stuff between your stereo headsets. In that case, you can claim that anything you do involves "a computer.") A program can be a guide to anything possessing some modicum of intelligence — even me. (Assuming, of course, that I don't fail the modicum of intelligence limitation.) Let's consider how we might write a program to guide human behavior.

A Human Program

Writing a program to guide a human is much easier than writing a program to guide a machine. We have a lot familiarity with and, therefore, understanding of a human. The most important familiarity is that we share a common language. In this section, let's write a "human program" and study its parts. Let's consider the problem of changing a flat tire.

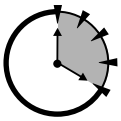
The algorithm

Changing a tire is relatively simple. The steps go something like this:

1. Raise the car.
2. Remove the lug nuts that affix the tire to the car.
3. Remove the faulty tire.
4. Mount the new tire.
5. Install the lug nuts.
6. Lower the car.

(I know that the words tire and wheel are not synonymous — you don't remove a tire from a car, you remove a wheel. Jumping back and forth between the words wheel and tire gets confusing, however. Just assume that the word tire includes the wheel on which the tire is mounted.)

At its core, this is the basis for a program. I could use these instructions to repair any of the many flats that I have experienced. More precisely, this is an algorithm. An algorithm is a description of the steps to be performed, usually at a high level of abstraction. An algorithm is to a program as a description of the principles of TV is to a TV circuit board.



20 Min. To Go

The processor

To make anything happen, an algorithm must be combined with some type of "do"er or processor. Our tire repair program assumes, for example, that there is someone to man (uh, I mean *person*) the jack, remove the lug nuts, and lift the tire into place. The objects mentioned — car, tire, and nuts — are powerless to move on their own.

Let's assume that our processor understands only a few words of English and understands them very literally. Let's assume that our processor understands these nouns, which are common in the tire-changing industry:

```
car
tire
nut
jack
wrench
```

(The latter two objects were not mentioned in the tire-changing algorithm, but were implied in phrases such as "replace tire." That's the problem with algorithms — so much goes unsaid.)

Let's further assume that our processor understands these verbs:

```
grab
move
release
turn
```

Finally, our processor person needs to be capable of counting and making simple decisions.

This is all that our tire-changing processor person understands. Any other command generates a blank stare.

The program

Given the vocabulary of our processor, it is clear that the processor cannot perform an operation such as "Remove the lug nuts from the car." The word "remove" is not in the processor's vocabulary. Further, no mention is made of the wrench with which to remove the lug nuts. (These are the types of things that go unsaid in normal speech.)

The following steps define the phrase "remove lug nuts" using terms that the processor understands:

```
1.  Grab wrench
2.  Move wrench to lug nut
3.  Turn wrench counterclockwise five times
4.  Remove wrench from lug nut
5.  Release wrench
```

Let's go over each step of this program in detail.

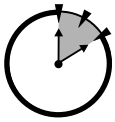
The processor begins with Step 1 and continues through each step in turn until reaching Step 5. In programming parlance, we say that the program flows from Step 1 through Step 5 even though the program doesn't go anywhere — it's the processor person.

In Step 1, the processor person retrieves the wrench. It is possible that the processor already has the wrench in hand; however, we cannot assume that. Step 2 places the wrench on the lug nut. Step 3 loosens the lug nut. Finally, Steps 4 and 5 return the wrench.

One problem with this algorithm jumps out immediately. How do we know that turning the wrench five times is sufficient to remove the lug nut? We could just make the number of turns large enough that it would remove the lug nut from any wheel. Not only is this solution wasteful, it may not even work. What will our processor do if the lug nut falls off and she is asked to turn the wrench again? Will the processor become confused and stop?

The following updated program utilizes our processor's limited decision-making capabilities to remove a lug nut properly:

```
1.  Grab wrench
2.  Move wrench to lug nut
3.  While lug nut attached
4.  {
5.     Turn wrench counterclockwise
6.  }
7.  Remove wrench from lug nut
8.  Release wrench
```



10 Min. To Go

Here the program flows from Step 1 through Step 2 just as before. Step 3 is something completely different. The processor person is asked to repeat all of the steps contained in the parentheses immediately following Step 3 until some condition is satisfied. In this case, to continue turning until the lug nut comes off. After the lug nut is no longer attached, the processor person continues executing at Step 7. Steps 3 through 6 are known as a processing loop because the processor loops in a circle.

This solution is far superior because it makes no assumptions about the number of turns required to remove any particular lug nut. Furthermore, this program is not wasteful by requiring more turns than necessary, and it does not ask the processor to turn a nut that is no longer there.

As nice as it is, this program still has a problem: it only removes a single lug nut. Most medium-size cars have five lug nuts on each wheel. We could repeat Step 2 through Step 7 five times, once for each lug nut. Removing five lug nuts doesn't work very well either. Compact cars typically have four lug nuts and large-size cars and most small trucks have six lug nuts.

The following program expands the previous solution to all of the lug nuts on a wheel, irrespective of the number of nuts holding the wheel in place.

```
 1.  Grab wrench
 2.  For each lug nut on the wheel
 3.  {
 4.     Move wrench to lug nut
 5.     While lug nut attached
 6.     {
 7.        Turn wrench counterclockwise
 8.     }
 9.     Remove wrench from lug nut
10.  }
11.  Release wrench
```

This program begins just as before with grabbing the wrench. Beginning with Step 2, the program loops through Step 10 once for each lug nut on wheel. Step 9 moves the wrench from one lug nut to the next before starting over at Step 2.

Notice how Steps 5 through Step 8 are still repeated until the given lug nut comes off the wheel. Steps 5 through 8 are known as an inner loop, while Steps 2 through 10 are the outer loop.

The complete program consists of the combination of similar solutions for each of the six steps in the original algorithm.

Computer processors

A computer processor works much like our human processor. A computer processor follows literally a string of commands built from a limited vocabulary.

Removing a wheel from a car seems like a simple task, and yet our processor person requires 11 commands to remove just a single wheel. How many instructions are required to move each of the thousands of pixels that make up a window on the computer screen when the user moves the mouse?

Done!

Unlike a human processor, a silicon processor is extremely fast. A Pentium II computer processor can execute roughly 100 million steps in a second. It still requires millions of commands to move the window around, but because the computer processor can execute them so quickly, the window appears to move smoothly across the screen.

REVIEW

This chapter introduced the principles of programming by describing how you might write a program to instruct a very stupid but extremely obedient mechanic in the intricacies of tire replacement.

- Computers do exactly what you tell them — no less, and certainly no more.
- Computer processors have a limited but well-defined vocabulary.
- Computer processors are smart enough to make simple decisions.

QUIZ YOURSELF

1. What are some nouns that a "human processor" that washes dishes might need?
2. What are some verbs?
3. What kind of decisions would a processor need to make?