

Activity 12

Marching orders—*Programming languages*

Age group Middle elementary and up.

Abilities assumed Giving and following simple instructions.

Time About 30 minutes or more.

Size of group At least two people, suitable for the whole class.

Focus

Communication.

Language.

Giving and following instructions.

Summary

Computers are usually programmed using a “language,” which is a limited vocabulary of instructions that can be obeyed. One of the most frustrating things about programming is that computers always obey the instructions to the letter, even if they produce a crazy result. This activity gives children some experience with this aspect of programming.

Technical terms

Programming languages; instruction sets; RISC computing.

Materials

Each child will need:

pencil and paper, and

tape measure, yard-stick, or something similar to measure approximate distances.

You will also need:

cards with pictures such as the ones in Figure 12.1.

What to do

1. Discuss whether it would be good if people followed instructions exactly. For example, what would happen if you pointed to a closed door and said “Go through that door.”

Explain that computers work by following lists of instructions, and that they do exactly what the instructions say—even if they are incorrect (or ludicrous). This activity involves some experiments that show how difficult it is to give instructions if they are followed to the letter.

2. The first exercise involves communicating a picture. Choose a child and give them a simple image, such as one of the pictures in Figure 12.1 (for younger children, simpler images like Figure 12.1a are best). Their task is to stand in front of the class and describe the picture so that the other children can reproduce it on their own piece of paper without seeing it. The object is to see how quickly and accurately the exercise can be completed—it is not a competition between the class and the child giving the instructions. The children are allowed to ask questions to clarify the instructions. Repeat this with a few images. You should choose images whose complexity matches the children’s ability.
3. Repeat the exercise, but this time do not allow the class to ask questions. The children will find this a lot more difficult, as there is no opportunity to make clarifications, and they are completely dependent on the ability of the child at the front to give clear instructions. It is best to use a simpler image for this exercise, as the children can get lost very quickly.
4. Now try the exercise with the instructing child hidden behind a screen, without allowing any questions, so that the only communication is in the form of instructions. The children will likely find that the lack of visual feedback makes the task very challenging.

Point out that this form of communication is most like the one that computer programmers experience when writing programs. They give a set of instructions to the computer, and don’t find out the effect of the instructions until afterwards. This should help the children to realize that it is very difficult to write a program that works correctly the first time. The *What’s it all about?* section below gives some material for further discussion about the reliability of computer programs.

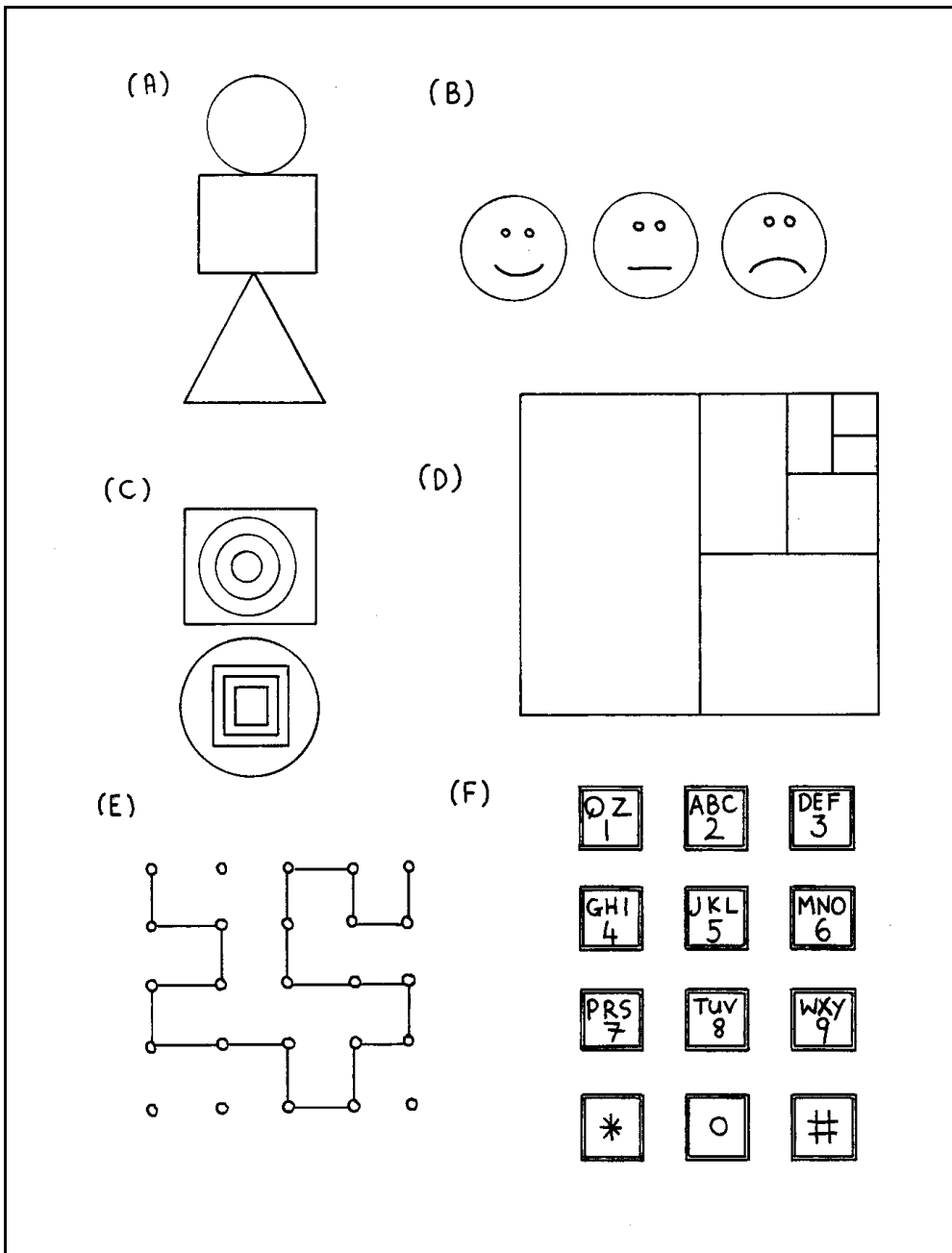


Figure 12.1: Some pictures for children to describe

5. This exercise can be extended by having the children write down instructions to carry out a given activity, such as constructing a paper dart or getting to some particular place.

For example, they could write instructions for each other about how to get to a mystery location around the school using only instructions of the form “Go forward x meters,” “turn left,” (90 degrees), and “turn right” (90 degrees). Have them follow each others’ instructions and evaluate how accurate they are. Children should be encouraged to refine their instructions until they have the desired effect.

An important issue in computing is the number of different instructions that are available. For example, the “turn right” instruction above could be eliminated because the same effect can be achieved by doing “turn left” three times. Discuss the trade-off between having lots of instructions available (it’s quicker to specify a route) and having only a few (it’s easier to remember them, but a longer list of instructions is needed).

Ask the children what instructions they would like to have available for navigating around the schoolyard. Can they think of places that can’t be reached with a limited instruction set? (For example, if there is no instruction for going up, it may not be possible to get to a room that is upstairs.)

Variations and extensions

There are all sorts of activities that can be described by a list of instructions drawn from a small, specialized vocabulary. For example, the children could locate a book in the library using instructions like “Go up one shelf,” “Point to the first book on that shelf,” “Point to the next book to the right,” and so on. Children can create their own limited set of instructions, and then try using them to describe an activity. The main element is that instructions must be written down and given in advance, and followed exactly.

Other examples of this kind of instruction include cake recipes and knitting patterns.

What’s it all about?

Computers operate by following a list of instructions, called a *program*, that has been written to carry out a particular task—whether it be word processing, accounting, game playing, or controlling a nuclear power plant. Programs are written in languages that have been specially designed for controlling the operation of computers. Literally hundreds of different languages have been invented, with names like Ada, BASIC, C, C++, COBOL, FORTRAN, LISP, Pascal, and RPG. Each language has a limited set of instructions that provide the vocabulary with which the programmer specifies what they want the computer to do. In principle, if a task can be programmed in one language, it can be programmed in any other one as well. This is an important idea in computer science that is known as the “Church-Turing thesis.” However, some languages are more suitable for some purposes than others, and hence the apparent babel of invented tongues.

Regardless of what language they use, programmers must become adept at specifying *exactly* what they want the computer to do. Unlike human beings, a computer will carry out

instructions to the letter even if they are patently ridiculous. The importance of getting the instructions right becomes clearer when one considers the consequences of an error in the program of a computer in an automated teller machine, a space shuttle launch, a nuclear power plant, or the signals on a train track. Even an error in a word processing program can ruin a software company if their program gets a reputation for losing people's work regularly. Errors like this are commonly called "bugs" in honor (so it is said) of a moth that was once removed ("debugged") from an electrical relay in an early 1940s electronic calculating machine.

It is a fact of life that large-scale computer software systems will never be bug-free. This became a major issue when the USA was working on the Strategic Defense Initiative ("Star Wars") program, a computer controlled system that was intended to form an impenetrable defense against nuclear attack. Some computer scientists claimed that it could never work because of the complexity and inherent unreliability of the software required. Software needs to be tested carefully to find as many bugs as possible, and it wouldn't be feasible to test this system since one would have to fire missiles at the United States to be sure that it worked!

There is a trade-off between having very large, complicated programming languages that make all sorts of instructions available, and small, simple ones that might have as few as only two instructions. This trade-off arose in a small way in the exercise, where the "turn right" instruction could be removed from the instruction language since the same effect could be achieved by issuing a turn left command three times. The advantage of complex languages is that the resulting programs are generally shorter, and hence easier to understand. The advantage of small languages is that they are easier to learn, and usually very efficient for computers to run. The so-called *reduced instruction set computers* (or RISC computers) use very small instruction sets so that they can run very quickly, in contrast to *complex instruction set computers* (CISC). Fortunately humans rarely have to program in the cut-down RISC languages, because systems are available (called compilers) that convert a program from a language that is easy for a human to read into the RISC language which runs very efficiently on the computer.

RISC computers generally run faster because they can be constructed to be "lean and mean," and are now generally favored over traditional CISC machines. However, people tend to use programming languages that are richer and result in shorter, more elegant programs, that are compiled automatically into a form that the RISC machine can use.

Further reading

Harel discusses programming languages in Chapter 3 of *Algorithmics*.