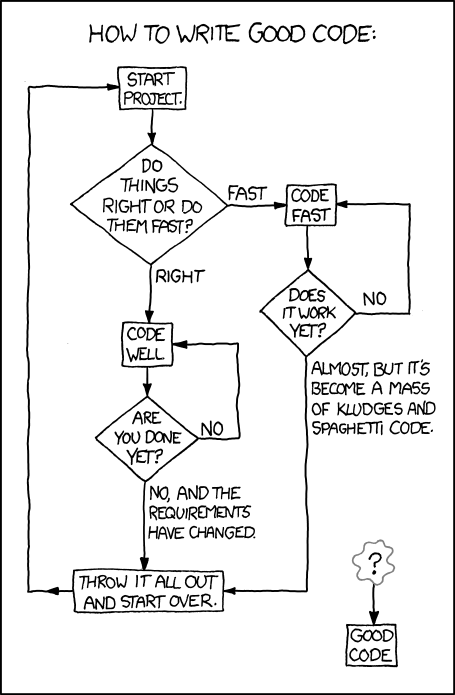
Methods of Algorithm Description

An algorithm is a step–by–step procedure for solving a problem; programming languages are essentially a way of expressing algorithms.

BOSTES expects that you are able to develop and interpret algorithms using pseudocode and flowcharts.

## 

## 

## Pseudocode:

Pseudocode uses English-like statements with defined rules of structure and keywords.

The pseudocode keywords are:

|  |  |
| --- | --- |
| f**or each procedure or subroutine** | BEGIN *name*  END *name* |
| **for binary selection** | IF *condition* THEN  *statements*  ELSE  *statements*  ENDIF |
| **for multi-way selection** | CASEWHERE expression evaluates to  A: process A  B: process B  ………………..  OTHERWISE: process …  ENDCASE |
| **for pre-test repetition** | WHILE *condition*  *statements*  ENDWHILE |
| **for post-test repetition** | REPEAT  *statements*  UNTIL *condition* |
| **for FOR / NEXT loops** | FOR *variable* = *start* TO *finish* STEP *increment*  *statements*  NEXT *variable* |

In pseudocode:

• keywords are written in capitals

• structural elements come in pairs, eg for every BEGIN there is an END, for every IF there is an ENDIF.

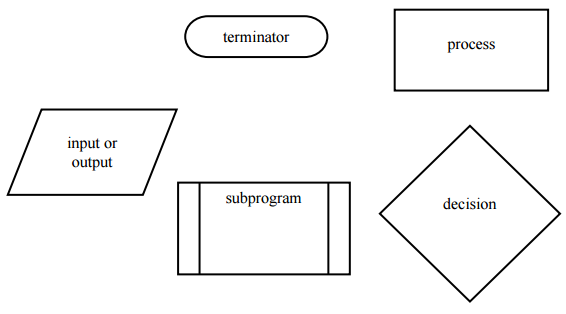
• indenting is used to identify control structures in the algorithm

• the names of subprograms are underlined. This means that when refining the solution to a problem, a subroutine can be referred to in an algorithm by underlining its name, and a separate subprogram developed to show the logic of that routine. This feature enables the use of the top-down development concept, where details for a particular process need only be considered within the relevant subroutine.

Flowcharts

Flowcharts are a diagrammatic method representing algorithms, which are read from top to bottom and left to right.

**Flowchart elements**

Flowcharts use the following symbols connected by lines with arrowheads to indicate the flow. It is common practice to show arrowheads to avoid ambiguity.

Flowcharts using these symbols should be developed using only the standard control structures (described on the following pages).

It is important to start any complex algorithm with a clear, uncluttered main line. This should reference the required subroutines, whose detail is shown in separate flowcharts.

A subroutine should rarely require more than one page, if it correctly makes use of further subroutines for detailed logic.

Programming Structures

**Control structures**

Algorithms are developed using the basic control structures of sequence, selection, repetition and subprograms. A description of each of these structures, together with examples of their use, follows.

1. **Sequence**

In a computer program or an algorithm, sequence involves simple steps which are to be executed one after the other. The steps are executed in the same order in which they are written.

|  |  |
| --- | --- |
| **Pseudocode** | **Flowchart** |
| process 1  process 2  … …  process n |  |

**Example Problem:**

Write a set of instructions to add two numbers and display the answer.

|  |  |
| --- | --- |
| **Pseudocode** | **Flowchart** |
|  |  |

1. **Selection**

Selection is used in a computer program or algorithm to determine which particular step or set of steps is to be executed. This is also referred to as a ‘decision’. A selection statement can be used to choose a specific path dependent on a condition.   
There are two types of selection: binary selection (two possible pathways) and multi-way selection (many possible pathways).

**Binary selection**

In binary selection, if the condition is met then one path is taken, otherwise the second possible path is followed. In each of the examples below, the first case described requires a process to be completed only if the condition is true. The process is ignored if the condition is false. In the second case, there is an alternative process if the condition is false.

|  |  |
| --- | --- |
| **Pseudocode** | **Flowchart** |
| **Binary Selection (1 process only)**  IF condition THEN  process 1  ENDIF |  |
| **Binary Selection (2 processes)**  IF condition THEN  process 1  ELSE  process 2  ENDIF |  |

**Example Problem:**

Determine the message to be displayed for a ‘guess the number’ game.

|  |  |
| --- | --- |
| **Pseudocode** | **Flowchart** |
|  |  |

**Multi-way selection**

Multi-way selection allows for any number of possible choices, or cases. The path taken is determined by the evaluation of the expression. Multi-way selection is often referred to as a case structure.

|  |  |
| --- | --- |
| **Pseudocode** | **Flowchart** |
| CASEWHERE expression evaluates to  choice a : process a  choice b : process  OTHERWISE : default process  ENDCASE | **Note:** As the flowchart version of the multi-way selection indicates, only one process on each pass is executed as a result of the implementation of the multi-way selection. |

**Example Problem:**

Write a set of instructions that describes how to respond to all possible signals at a set of traffic control lights.

|  |  |
| --- | --- |
| **Pseudocode** | **Flowchart** |
|  |  |

1. **Repetition**

Repetition allows for a portion of an algorithm or computer program to be executed any number of times dependent on some condition being met. An occurrence of repetition is usually known as a loop.

An essential feature of repetition is that each loop has a termination condition to stop the repetition, or the obvious outcome is that the loop never completes execution. This is known as an infinite loop and is obviously undesirable. The termination condition can be checked or tested at the beginning or end of the loop, and is known as a pre-test or post-test respectively. Following is a description of each of these types of loop.

**Repetition: pre-test**

A pre-tested loop is so named because the condition has to be met at the very beginning of the loop or the body of the loop is not executed. This construct is often called a guarded loop. The body of the loop is executed repeatedly while the termination condition is hue.

|  |  |
| --- | --- |
| **Pseudocode** | **Flowchart** |
| WHILE condition is true  process(es)  END WHILE |  |

**Example Problem:**

Determine a safety procedure for travelling in a carriage on a moving train

|  |  |
| --- | --- |
| **Pseudocode** | **Flowchart** |
|  |  |

**Repetition: post-test**

A post-tested loop executes the body of the loop before testing the termination condition. This construct is often referred to as an *unguarded loop.* The body of the loop is repeatedly executed until the termination condition is true. An important difference between a pre-test and post-test loop is that the statements of a post-test loop are executed at least once even if the condition is originally true, whereas the body of the pre-test loop may never be executed if the termination condition is originally true. A close look at the representations of the two loop types makes this point apparent.

|  |  |
| --- | --- |
| **Pseudocode** | **Flowchart** |
| REPEAT  process(es)  UNTIL condition is true |  |

**Example Problem:**

Determine a process to beat egg whites until they are fluffy

|  |  |
| --- | --- |
| **Pseudocode** | **Flowchart** |
|  |  |

1. **Subprograms**

Subprograms, as the name implies, are complete part programs that are used from within the main program section. They allow the process of refinement to be used to develop solutions to problems that are easy to follow. Sections of the solution are developed and presented in understandable chunks, and because of this, subprograms are particularly useful when using the top-down method of solution development.

When using subprograms it is important that the solution expression indicates where the main program branches to a subprogram. It is equally important to indicate exactly where the subprogram begins. In pseudocode, the statement in the main program that is expanded in a subprogram is underlined to indicate that further explanation follows. The expanded subprogram section should be identified by using the keywords BEGIN SUBPROGRAM followed by the underlined title used in the main program. The end of the subprogram is marked by the keywords END SUBPROGRAM and the underlined title used in the main program.

When using flowcharts, a subprogram is shown by an additional vertical line on each side of the process box. This indicates that the subprogram is expanded elsewhere. The start and end of the subprogram flowchart uses the name of the subprogram in the termination boxes.

|  |  |
| --- | --- |
| **Pseudocode** | **Flowchart** |
| BEGIN MAINPROGRAM  process 1  process 2  process 3  process 4  END MAINPROGRAM  BEGIN SUBPROGRAM process 2  do this  do that  END SUBPROGRAM process 2 |  |

In many cases a subprogram can be written to do the same task at two or more points in an algorithm. Each time the subprogram is called, it may operate on different data. To indicate the data to be used one or more *parameters* are used. The parameters allow the author to write a general algorithm using the *formal parameters*. When the subprogram is executed, the algorithm carries out its task on the *actual parameters* given at the call. The parameters to be used by a subprogram are provided as a list in parentheses after the name of the subprogram. There is no need to include them at the end of the algorithm.

**Example Using a single Parameter:**

|  |  |
| --- | --- |
| **Pseudocode** | **Flowchart** |
| BEGIN MAINPROGRAM  read (name)  read (address)  END MAINPROGRAM  BEGIN SUBPROGRAM read (array)  set pointer to first position  get a character  WHILE character is not the end of data AND there is room in the array store character in the array at the position given by the pointer  increment the pointer  get a character  ENDWHILE  END SUBPROGRAM read  Note: At the first call of this subprogram the characters are read into the array called ‘name’, at the second call the characters are read into the array called ‘address’. |  |