Control – Expressions and Statements
Control

Control is the study of the semantics of execution paths through code.

What gets executed,
When, and
In what order?
Control

Control is achieved by two major ways:

- The use of expressions and statements.
- The use of procedures/function/method calls and return.
Control

Expression:

- In its pure (mathematical) form:
  - Returns a value
  - Produces NO side effects: does not change program memory.
  - Example: $3 + 4 \times 5$

Statement:

- Is executed for its side effects and returns no value.

Many languages do not distinguish between expressions and statements. They allow expressions to have side effects.
The earliest kind of control was the GOTO:

- Are simple imitations of the jump statement of assembly.

- Transfers control directly, or after a test, to a new location in the program.
Algol60 introduced improvements to structured control:

- Control statements transfer control to and from sequences of statements.

- Such statements have a:
  - Single entry point.
  - Single exit point.
• Expressions:

  – A simple arithmetic expression is $3 + 4 \times 5$

  – Operators can take one or more operands:

    • Unary operator: one operand e.g. $-3$
    • Binary operator: two operands e.g. $3 \times 5$
Control

• Expressions:

  – Operators can be written in three notations:

    • Infix  (Inorder traversal of the syntax tree of the expr)
      – (Left – Root – Right)
      – 3 + 4 * 5

    • Postfix (Postorder traversal)
      – (Left – Right – Root)
      – 3 4 5 * +

    • Prefix (Preorder traversal)
      – (Root – Left – Right)
      – + 3 * 4 5
Control

Expressions:

- Postfix and prefix are very powerful notations.

- With postfix and prefix notations, parenthesis are not necessary!

- Example: \((3 + 4) * 5\) is written
  - In postfix notation: 3 4 + 5 *
  - In prefix notation: * + 3 4 5

- The ambiguity of precedence is not present.
Control

Expressions:

With postfix and prefix notations, you can easily express associativity:

- Example: $3 4 5 + +$ is a right association
  - Equivalent to $3 + (4 + 5)$

- Example: $3 4 + 5 +$ is a left association
  - Equivalent to $(3 + 4) + 5$

However, they are more difficult to write, less writeable and less readable.
Control

Expressions:

- Many programming languages use:
  - Infix notation with predefined associativity and precedence to define binary operators:
    - E.g. 3 + (4 * 5)
  - Prefix notation to define functions
    - E.g. add (3, mul(4, 5))
Control

Expressions:

- There must be expressions that modify the execution/evaluation process such as:
  - If then else
  - Short circuit boolean operators
  - Case/switch expressions

- In most cases, such expressions need to have a defined manner of execution so that programs may be deterministic.
Control

- Side Effects:

  - A side effect is any observable change to memory, input, or output.

  - Programs must have side effects to be useful.

  - Example: \( x = y++ \) will increment \( y \), and save it into the memory location of \( x \).
Control

■ Strictness:

- An evaluation order for expressions is strict if all subexpressions of an expression are evaluated, whether or not they are needed to determine the value of the result, non-strict otherwise.

- Arithmetic is almost always strict.

- The Java short circuit && and || is not strict.
  - A && B && C
  - The evaluation of B is delayed until A is evaluated.
  - The evaluation of C is delayed until B is evaluated.
Control

Strictness:

- Some languages use a form of non-strictness called normal-order evaluation: no expression is ever evaluated until it is needed (Haskell). Also called delayed evaluation.

- A form of strict evaluation called applicative-order is more common: "bottom-up" or "inside-out".

- Still leaves open whether left-to-right or not.
Control

- Conditional Statements:
  - Are the most typical form of structured control in execution of a group of statements under certain conditions.
  - Involve a logical (boolean) test before entering a sequence of statements.
  - The if and case statements are the most common.
Control

Conditional Statements:

If statements:

- If-statement -> if (expression) statement [else statement]

The following if statement is ambiguous (has two parse trees):

- If (e1) if (e2) S1 else S2
  - Draw the two parse trees.
Control

• Conditional Statements:

  – If statements:

    • This ambiguity is called the dangling-else problem.

    • The syntax does not tell us which if the else is associated with.

    • C and Pascal resolve this:
      – The else is to be associated with the closest prior if that does not already have an else part.
      – Also known as the most closely nested rule.

    • Another disambiguating rule is to use a bracketing keyword such as Ada’s endif.
Control

• Conditional Statements:

  – Case and switch statements:
    • Ordinal values instead of booleans are checked.
    • Example in C:

      ```c
      switch(x)
      {
      case 0:
          ...
          break;
      case 1:
          ...
          break;
      default:
          //do nothing
          break;
      }
      ```
Control

• Conditional Statements:

  – Case and switch statements:

    • Java has a switch statement that is virtually identical to that of C.

    • In Ada (A more standard version of a case stmt):

      ```
      case x is
        when 0 ->
          ...
        when 2 .. 5 ->
          ...
        when others ->
          null;
        end case;
      ```
Control

- Conditional Statements:
  - Case and switch statements:
    - In ML, the case construct is an expression that returns a value, rather than a statement:
      ```
      case x of
          0 -> 2 |
          2 -> 1 |
          _ -> 10
      ;
      ```
Control

- Conditional Statements:

  - Loops and Variations on While:

    - C/C++/Java:
      - While (e) S

    - Ada:
      - While e loop S1 .. Sn end loop;

  - The condition must be boolean in Ada and Java, but not in C/C++.
    - One can say while (1) in C/C++
Control

- Conditional Statements:

  - Loops and Variations on While:

    - There is also a do while:
      - do S while(e)
      - Equivalent to:
        - S;
        - while (e) S

    - do while is a construct completely expressible using other constructs.

    - It is what is called a syntactic sugar (adds flavor and flexibility)
Conditional Statements:

- A break can be used inside a loop to break this loop.

```plaintext
while (e)
    S1
    S2
    break
    S3
end while
```
Control

Conditional Statements:

- A continue skips the remainder of the current iteration.

while (e)

S1
S2
continue
S3
end while
Control

Conditional Statements:

A for-loop is a special kind of looping:

```plaintext
for (e1; e2; e3) S;
```

Diagram:
- **Initializer**
- **Test**
- **Update**
Control

• Conditional Statements:
  
  – For loop in C/C++/Java:
    
    ```
    for (i=0; i < 5; i++){
    }
    ```

  – For loop in Ada:
    
    ```
    for i in 0 .. size -1 loop
    ...
    end loop;
    ```

  The initialization and update are more compact
Control

Conditional Statements:

Typical restrictions primarily involve the counter i:

- The value of i usually cannot be changed in the body of the loop.
- The value of i is usually undefined after the loop.
- Must be a restricted type, may not be declared as a parameter to a procedure.
- This varies from one language to another.
Control

Conditional Statements:

Questions to ask about the variable i:

- Is the bound evaluated only once?
- What if the lower bound is greater than the upper bound?
- Is the control variable still defined even with the use of a break or continue?
Exception Handling:

Exception handling is the control of error conditions or other unusual events during the execution of a program.
Control

Exception Handling:

- Examples of exceptions include:
  - Runtime errors:
    - Out of range array subscripts.
    - Division by zero.
  - In interpreted languages, exceptions can include static errors such as:
    - Syntax
    - Type errors.
  - An exception can be any unusual event, such as an input failure or timeout.
Exception Handling:

- Exception handling can cause an implicit transfer of control within a program.
- We try a given piece of code.
- If an unusual event happens, and exception is thrown.
- The exception is then caught by an exception handling code.
Control

■ Exception Handling:

- It largely imitates, in a programming language, hardware interrupts or traps where the processor transfers control automatically to a location that is specified in advance according to the kind of error or interrupt.

- Exception handling attempts to avoid an operating system taking control of a program, which means avoiding crashing and abortions.

- Programs exhibiting a behavior away from abortions and crashing tend to be very robust.
Control

Exception Handling:

- Exception handling also contributes to reliability and security of applications:
  - Programs recover from errors and continue execution.
Control

■ Exception Handling:

☑ Even when using exception handling mechanisms, it is almost impossible to catch and handle every single type of error that may occur due to:

■ Design negligence.

■ Very low level errors that may cause the OS to interfere such as:
  ☑ Hardware failure
  ☑ Memory allocation problems
  ☑ Communication problems
Control

Types of Exception Handling:

- Asynchronous exceptions:
  - Include drastic failures such as memory, communication, hardware failures.
  - May occur at any time.
  - Programs usually cannot do anything about such kinds of exceptions.
  - The operating system usually interferes.

- Synchronous exceptions:
  - Include errors that a program can handle.
  - Example: Array index out of bound
Control

- Exception Handling:

  - It was originally pioneered by PL/I in the 1960’s.

  - Significantly advanced by CLU in the 1970’s.

  - Now virtually all major languages including C++, Java, Ada, ML, Lisp have built in exception handling mechanisms.

  - C, Scheme, and Smalltalk do not have exception handling mechanisms.
Control

Exception Handling:

- Exception handling integrates very well with object oriented mechanisms in Java and C++.

- It integrates well with functional mechanisms of ML and common Lisp.

- Some languages that do not have built in exception handling mechanisms have libraries that support it.
Java Exception Handling Example:

class Exc2 {
  public static void main(String args[]) {
    int d, a;

    try {  // monitor a block of code.
      d = 0;
      a = 42 / d;
      System.out.println("This will not be printed.");
    } catch (ArithmeticException e) {  // catch divide-by-zero error
      System.out.println("Division by zero.");
      a = 0;
    }
    System.out.println("After catch statement.");
  }
}
Exceptions:

- Typically, an exception is represented by a data object.

- In Java, every exception is an object created upon occurrence of the condition causing the exception, and is passed to the exception handling mechanism.

- In functional languages, the data object is a value of some type.

- The type of exception object can be usually either predefined, or user defined as in Java.
Control

- Exceptions:

  - In C++, there is no special exception type.

  - Any structured type such as a struct or class can represent an exception:

    ```
    struct myDefinedCPlusPlusException{
        int wrongValue;
    }
    ```
Exceptions:

- When an exception is raised, typically:
  - The current computation is abandoned.
  - The runtime system begins a search for a handler.
  - In Ada, Java, and C++, the search begins with the handler section of the block in which the exception was raised.
  - If no handler is found, then the handler of the outer block is consulted. This is called propagating the exception.
  - If the outermost block is reached and there is no handler, the program exits.
Control

 Exceptions:

- When an exception handler is found, typically:
  - We can return to the point at which the exception was first raised, and begin execution again with that statement or expression.
    - This is called the resumption model of exception handling.
    - We need to store the original environment so that we can re-establish execution again.
  - Alternatively, we can continue execution of code immediately following the block of code in which the handler was found (Java).
    - This is called the termination model.
    - It discards all blocks until a handler is found.
Control

Exception Specification:

- Is a list of exceptions added to the declaration of a function guaranteeing that the function will only throw the exceptions in the list, and no others.

- This is used in C++

- Exception specification guarantees what exceptions can appear in which contexts.

- Without an exception specification, any code can throw any exception at any point in time!

- C++ does NOT require the use of exception specifications.