When drawing flowcharts on an exam you can use the traditional symbols or the ones used in RAPTOR.

- **flowchart symbols**
- **terminal** capsule, race track **start / end module( ) / return**
- **process** rectangle **RAPTOR calls this assignment**
- **decision** diamond
- **input** parallelogram **RAPTOR**
- **output** parallelogram **RAPTOR**
- **call** unique symbol used in **RAPTOR** for a pre-defined process
- **predefined process** rectangle with a line across the top or down the sides
- **connector** connects flowlines that come together
- **flowline** shows flow of direction between symbols

**Note:** RAPTOR uses an oval for a terminal symbol.

- **program development cycle** understand, plan, code, translate, test
- **logic planning tools**
  - flowchart / pseudocode
  - modules / subcharts **RAPTOR has subcharts and procedures**
  - hierarchy chart
- **syntax, semantics, and logic**
  - the syntax of a language is the rules you must follow to create a valid statement
  - the semantics of a statement – describes what the statement does
  - logic is the development of a solution for a program. You develop the program “logic” as a set of tasks
- **control structures** sequence, selection, loop (aka repetition and iteration)
  - single entry point
  - single exit point
  - execution of control structures is sequential
  - structures can be STACKED
  - structures can be NESTED
  - Pseudocode keywords: get, put, if – endif, while – endwhile, for – endfor, etc.
- **data types used in Pseudocode**
  - integer int integer number
  - real double floating-point number (number with decimal digits)
  - text string any sequence of 0 or more characters in “ “
  - boolean boolean true/false values only
variable
  o a named location in memory that stores a value while a program is executing
  o variables usually must be DECLARED before they can be used
  o variables can be INITIALIZED in their declaration
  o an assignment symbol is used in RAPTOR to declare a variable
    \texttt{quantity} \leftarrow 0
variable declaration syntax
  o \texttt{<data type> <name> [ = <initial value> ] [ , <name> [ = <initial value> ] ] ...}
  o \texttt{<data type>} is \texttt{int}, \texttt{double}, \texttt{string} or \texttt{boolean}
  o \texttt{<name>} is any valid identifier
  o \texttt{<initial value>} is the value being assigned to the variable in the declaration (optional)

named constant  [ Note – RAPTOR doesn’t support named constants – use the convention described below ]
  o a “read-only” or “constant” variable
  o \texttt{final <data type> <name> = <constant value>}
  o a named constant must be initialized in the declaration!
    \texttt{final int NUM_TESTS = 3}
  o by convention the identifier for a named constant uses all uppercase letters and underscores

literal
  o a literal is an \textit{unnamed value} (the value is not stored in a variable or named constant)
  o a literal has a data type
    \begin{itemize}
    \item \texttt{int} literal \hspace{1cm} \texttt{0, -3, 40}
    \item \texttt{double} literal \hspace{1cm} \texttt{-3.9, 2.5, .2349568}
    \item \texttt{string} literal \hspace{1cm} \texttt{“CIS 103”, “Enter a number “, “Your total is “ + total}
    \item \texttt{boolean} literal \hspace{1cm} \texttt{true, false \hspace{2cm} guessedNumber = false, EOF = true, found = false}
    \end{itemize}
  o a literal in a program is often referred to as a \textit{“magic number”} \hspace{1cm} \texttt{total / 3} \hspace{1cm} (what is 3?)

identifier
  o an identifier is anything you create a name for: variable, named constant, module, etc.
  o rules for creating a valid identifier
    \begin{itemize}
    \item use letters A-Z, a-z, and digits 0-9 \hspace{1cm} [ languages may or may not be \textit{case-sensitive} ]
    \item use any valid special characters (such as an \texttt{underscore})
    \item the identifier can not begin with a digit or contains any spaces
    \item the identifier can not be a keyword in the language (pseudocode)
    \end{itemize}

variable naming conventions
  o use an underscore to separate words \hspace{1cm} \texttt{gross_pay, total_sales, grand_total_sales}
  o use camel casing \hspace{1cm} \texttt{grossPay, totalSales, grandTotalSales}

named constant naming convention
  o use all uppercase letters and separate words with underscores
  o Examples: \texttt{COMMISSION_RATE, FED_TAX_RATE, NUM_DRAWERS}

expression
  o an expression is something that can be evaluated to get a value. The value will be of some data type
    \begin{itemize}
    \item variable
    \item literal
    \item named constant
    \item arithmetic expression \hspace{1cm} \texttt{5 * n + 7 \hspace{1cm} length * width * height \hspace{1cm} PI * radius * radius}
    \item relational expression \hspace{1cm} \texttt{x < y \hspace{1cm} x > y \hspace{1cm} score >= 0 \hspace{1cm} score <= 100}
    \item logical expression \hspace{1cm} \texttt{score < 0 \hspace{1cm} or \hspace{1cm} score > 100 \hspace{1cm} score >= 0 \hspace{1cm} and \hspace{1cm} score <= 100}
    \item string expression \hspace{1cm} \texttt{name = firstName + “ “ + lastName \hspace{1cm} “Hello “ + name + “!”}
    \item combinations of the above components
    \end{itemize}
• statement
  o a statement in a high-level language or pseudocode is equivalent to one structure in a flowchart (with a declaration being the only exception in RAPTOR)
    ▪ “sequence” statements
      • assignment statement  \( x = 3 \)  \( \text{RAPTOR: SET } x \text{ TO } 3 \)
        o format for an assignment statement is  \( \text{variable} = \text{expression} \)
        o LHS variable = (is assigned) the value of the RHS expression
          ▪ LHS means left-hand-side, RHS means right-hand-side
        o LHS must always be a variable, RHS can be any valid expression
          ▪ \( x = 3 \)  //this is a valid assignment statement
          ▪ \( 12 = \text{number} \)  //this is NOT a valid assignment statement
        o the data type of the variable and the expression should be compatible
          ▪ int number  \( \text{number} = 5 \)  //compatible
          ▪ int number  \( \text{number} = \text{"five"} \)  //not compatible
          ▪ boolean found  \( \text{found} = \text{true} \)  //compatible
          ▪ boolean found  \( \text{found} = \text{"true"} \)  //not compatible
    • input statement  \( \text{get number} \)
    • output statement  \( \text{put "The total is " + total} \)
    • call statement  \( \text{processRecord( ), calculatePay( )} \)
  ▪ “selection” statements
    • if statement (single selection)
      if guess = number then
        put “You guessed the number!”
      endif
    • if–else statement (dual selection)
      if guess = number then
        put “You guessed the number!”
      else
        put “Sorry, try again”
      endif
  ▪ “loop” statements
    • pre-test loop  \( \text{[ pseudocode while and for loops ]} \)
      int count = 1
      while count <= 5
        put count
        count = count + 1
      endwhile
    • post-test loop  \( \text{[ pseudocode do .. while loop ]} \)
      int count = 1
      do
        put count
        count = count + 1
      while count <= 5
    • mid-test loop
      o (no example for this, mid-test loops are not commonly used in Java and C++)
operators
  • arithmetic operators: +, -, *, /, %
  • relational operators: <, <=, >, >=
  • equality operators: = (is assigned)
  • assignment operator: +=, -=, *=, /=, %=
  • modulus operator: %

- Provide for a more succinct way to express an assignment operation.
- totalScore += score
- a += b is the same as a = a + b
- a *= b is the same as a = a * b

- Evaluating an arithmetic expression
  - Precedence: process operators with higher precedence before operators with lower precedence.
  - Associativity:
    - Either left-to-right (most binary operators) or right-to-left (assignment).

- Condition
  - Condition: the name given for controlling an expression evaluated in a selection or loop control structure.

- Modularization
  - Abstraction: start with the BIG picture.
  - Algorithm: the logic you develop for a task that will be implemented in a module.
  - Functional cohesion: a module is cohesive if it performs a SINGLE TASK.
  - Coupling: modules may be loosely-coupled (desirable) or tightly-coupled (avoid).

- In RAPTOR and many programming languages, execution begins with a module named main.

- Errors
  - Syntax: No syntax errors but program produces incorrect output.
  - Logic: division by zero, invalid index, file not found, assignment between a string and a number.

- Pseudocode
  - Conventions for pseudocode - see document in learning resources folder.

- Loop control structure
  - Definite counter-controlled loop: initialize lcv / test / update lcv.
  - Indefinite sentinel-controlled loop: compare input value to the sentinel value.
    - Sentinel value: (a dummy value outside the range of valid data) used to exit an input loop.
    - RAPTOR: End_OF_Input -1 999.

- Local variables versus global variables: RAPTOR procedure versus subchart.

- Test
  - True/false matching:
    - Multiple choice:
    - Short answer:
      - Draw a flowchart / write pseudocode
      - Modules, modularizing
      - Evaluate arithmetic expressions and identify:
        - Order operators are evaluated in.
        - Determine the value of an expression.