Principles of Programming Languages
Lecture 22

Wael Aboulsaadat

wael@cs.toronto.edu

http://portal.utoronto.ca/

Acknowledgment: parts of these slides are based on material by Diane Horton & Eric Joanis @ UoT

References: Scheme by Dybvig
Concepts of PL by Sebesta
Prog. in Prolog by Clocksin and Mellish
PL Concepts and Constructs by Sethi
ML for the Working Prog. By Paulson
PL Pragmatics by Scott

Components of an Imperative Language

- **Data types**
- **Variables, operators & Expressions**
- **Iteration construct**
- **Branching construct**
- **Subprogram construct**

→ Container construct

Container Construct

- **Syntactical variations**
  - Prototype
  - Implementation

- **Binary variations**
  - Application
  - Library
  - Component

Container Construct

- **Syntactical variations**
  - Prototype & implementation
  - Implementation
- **C/C++ separates between prototype and implementation of code**

```
class HelloWorld {
  public:
     HelloWorld();
    ~HelloWorld();

  public:
    //Methods:
    void on_button_clicked();

  protected:
   //Attributes
   Button ok_button;
};
```

hello.h

```
#include "hello.h"

HelloWorld::HelloWorld(){
   // ….
}

HelloWorld::~HelloWorld(){
    //…..
}

HelloWorld::ok_clicked(){
     //….
}
```

hello.cpp

Container Construct

- C/C++ separates between prototype and implementation of code

- Decreases writability: have to maintain 2 files per class/set of functions.

```
class HelloWorld {
   public:
      HelloWorld();
     ~HelloWorld();

  public:
     //Methods:
     void printMessage();

  protected:
    //Attributes
    Button ok_button;
};
```

hello.h

```
#include "hello.h"

HelloWorld::HelloWorld(){
   // ….
}

HelloWorld::~HelloWorld(){
    //…..
}

HelloWorld::printMessage(){
     //….
}
```

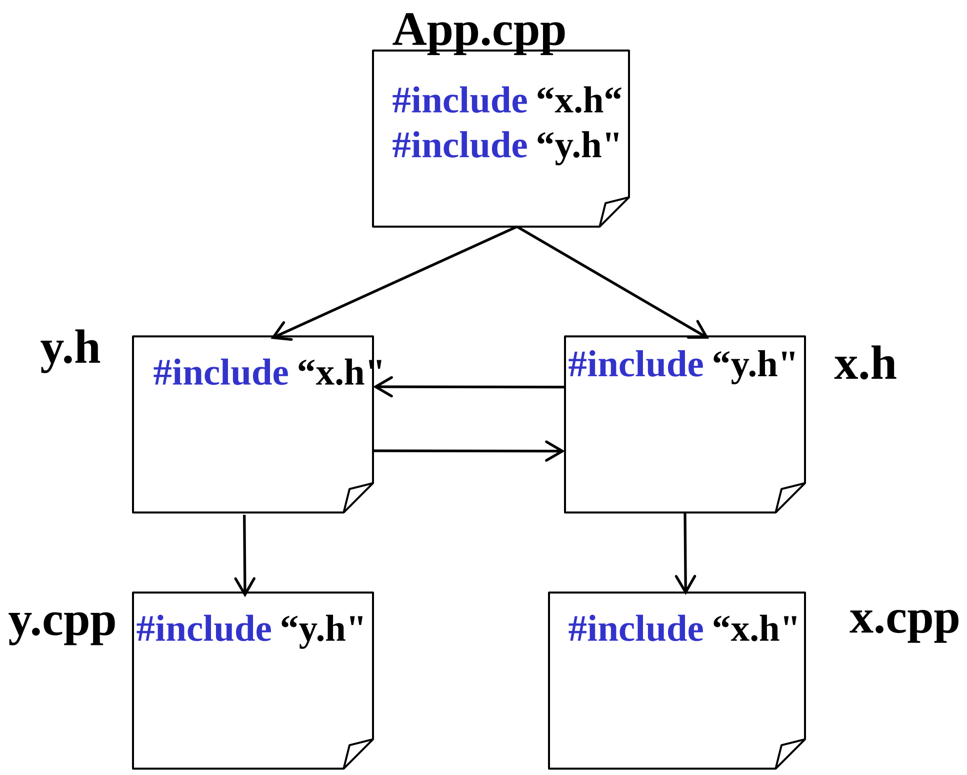
hello.cpp

```
#include "hello.h"


int main(){

      HelloWorld hello;
       hello.printMessage ();
}
```

App.cpp

Container Construct

- Languages which separates prototype from implementation often force programmer to handle circular references – which will break compilation!



Container Construct

- Languages which separates prototype from implementation often force programmer to handle circular references – which will break compilation!

- ifdef ensures that a file is included by the compiler only once

```
#ifndef HELLO
#define HELLO
class HelloWorld {
   public:
      HelloWorld();
     ~HelloWorld();

  public:
     //Methods:
     void printMessage();

  protected:
    //Attributes
    Button ok_button;
};
#endif
```

hello.h

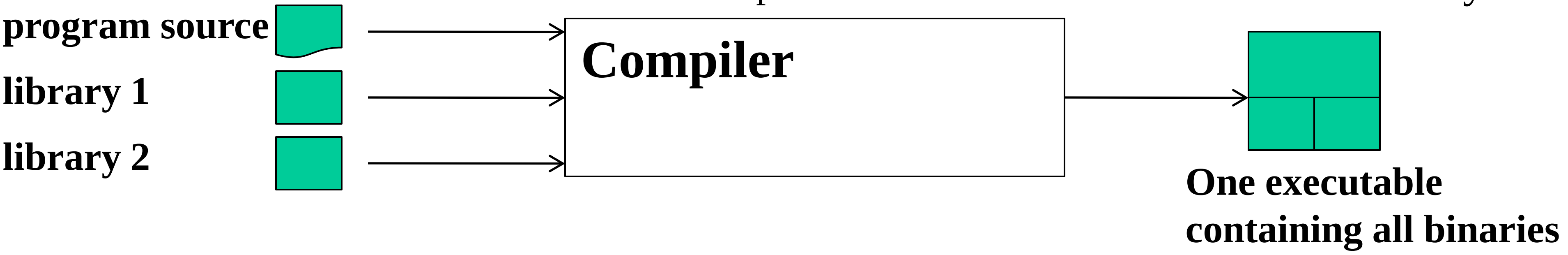
Container Construct

- Binary variations: Library
- A library is a set of functions packaged in one file.
- No main function/method. It can only be used from a program. You can’t run it!
- Libraries come in 2 variations: static or dynamic.

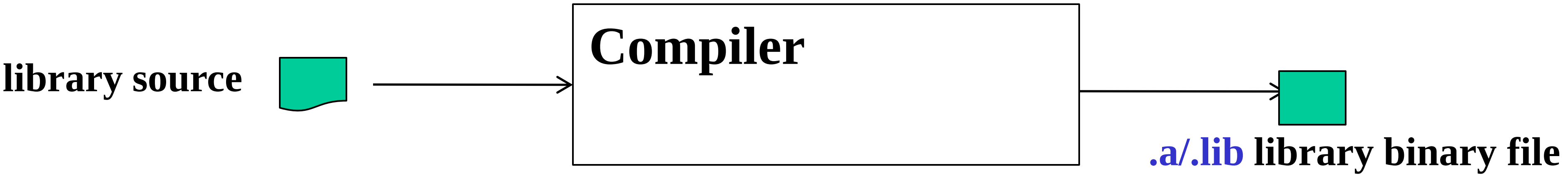
Container Construct

- **Binary variations: Library**

- **Static library**
  - Windows: .lib   Linux: .a (e.g. /usr/local/lib)
  - Linked with the program during compilation.
    - Calls to function in lib is replaced with relative address of func in binary



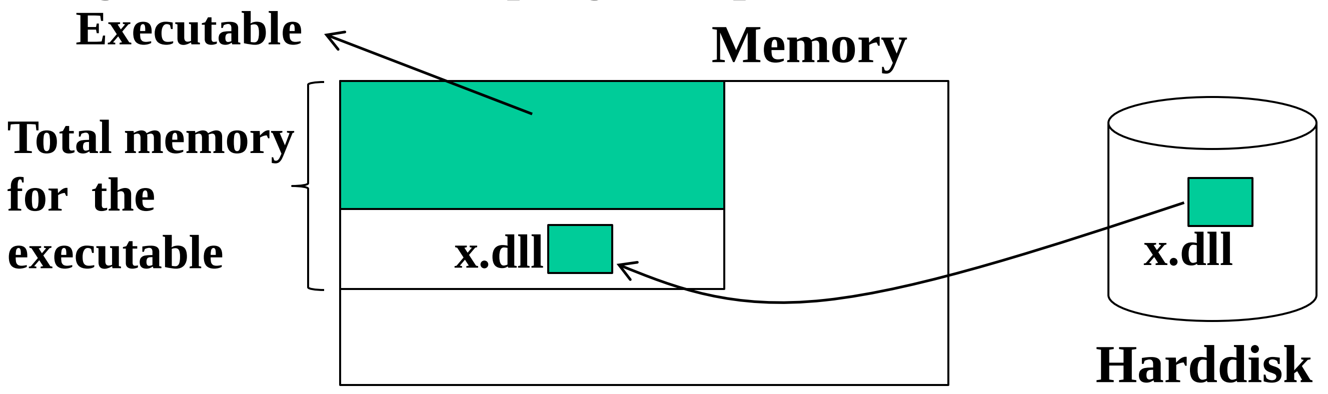
  - Compiler must support generating a static library from a set of functions (with no main function/method)



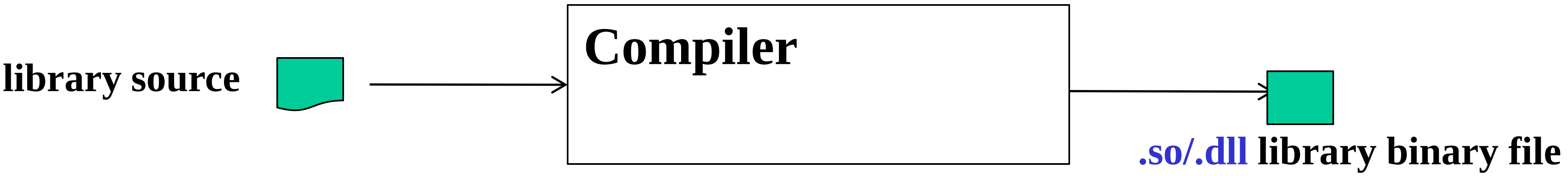
Container Construct

- **Binary variations: Library**

- **Dynamic library**
  - Windows: .dll (c:/windows/system32), Linux:.so (/usr/lib), Mac: .dylib
  - Loaded during runtime into the program space



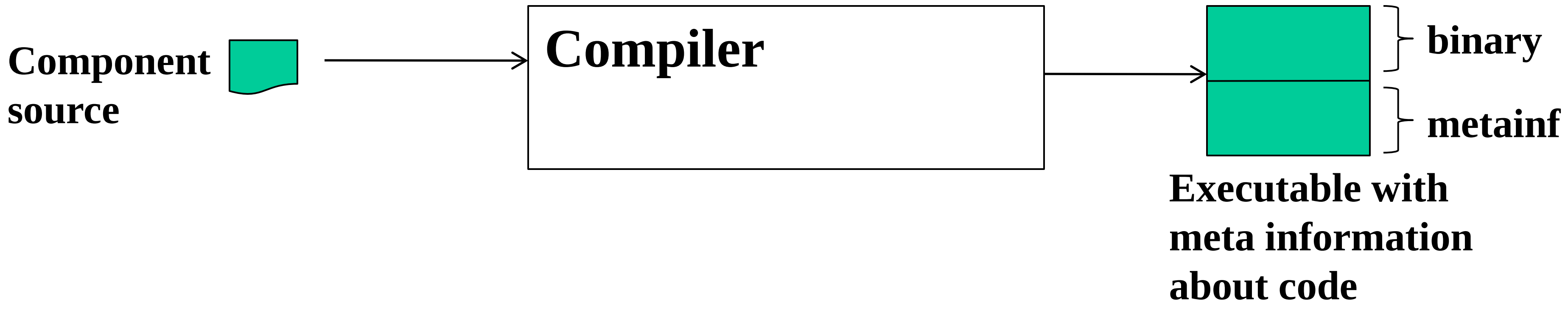
  - OS provides function to load library (Windows: LoadLibrary("x.dll"))
  - Compiler must support a mechanism to do the following:



Container Construct

- Binary variations: Component

- Similar to library with meta information is added to binary to enable reflection
- Naming convention is enforced to identify set/get methods
- Examples: Microsoft COM components, JavaSoft JavaBeans

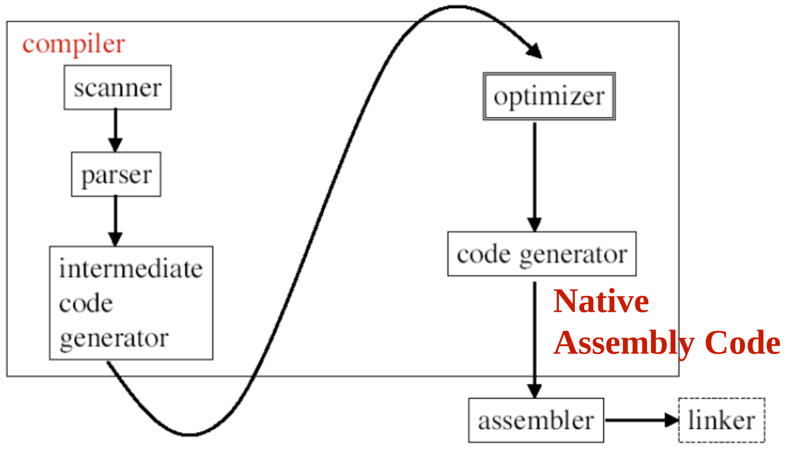


- Refer to reflection lecture for an example how Java supports this.

Container Construct

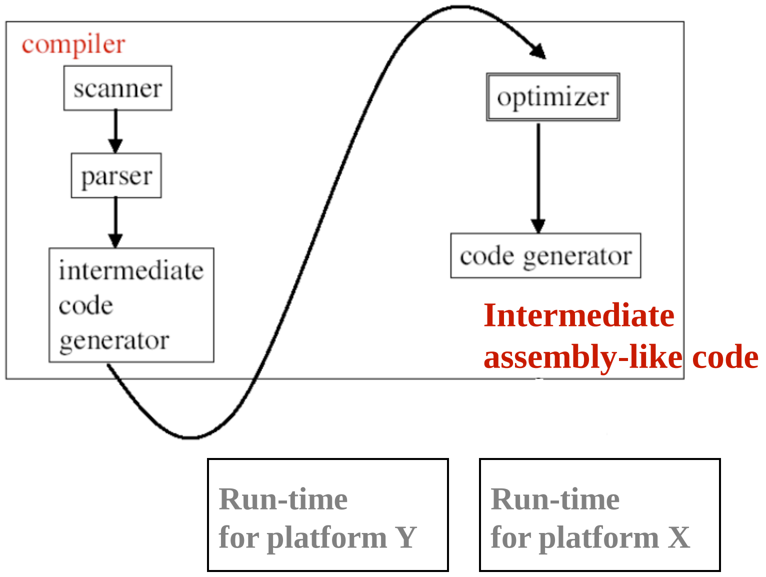
- **Binary variations: application**
- Can either be native or virtual-machine based
- Java/Microsoft .Net platform vs. native C/C++/Fortran app

Native Application



E.g. cprog.exe

Virtual-Machine Application



E.g. jvm javaprog

Components of an Imperative Language

- Data types
- Variables, operators & Expressions

→ Iteration construct

→ Branching construct

- Subprogram construct
- Container construct

Control Statements: iteration

- **The repeated execution of a statement or compound statement is accomplished either by iteration or recursion, here we look at iteration, because recursion is subprogram control.**

- **An Iteration statement is one that causes a statement or collection of statements to be executed zero, one or more times.**
  - E.g. `for( nIndex = 0; nIndex < 10; nIndex++)   // Java + C + C++`

- **General issues:**
  - How is iteration controlled?
  - Where is the control mechanism in the loop?

- **Types of iteration constructs:**
  - Counter controlled loops
  - Logically controlled loops
  - User controlled loops
  - Data structures controlled loops

Control Statements: iteration cont'd

- **User controlled loops:**
  - Test the condition in the middle of the loop and break if false
  - Language must provide break, continue or similar statement, why?
  - E.g.

```
loop                              // Ada
     if somevariable < somevalue
        exit
end loop
```

```
while( true ){      // C++
      if(somevariable < somevalue)
         break;
}
```

Control Statements: iteration cont’d

- **Logically controlled loops:**
  - Execution of loop continues as long as certain logical condition is true
  - Types:
    - Pretest:
      - Test the condition before entering the loop. Might not execute the loop.
      - E.g.

```
while( x < 10 ) do                // Pascal
while(x < 10)                     // C/C++/Java
```

    - Posttest:
      - Test the condition at the end of the loop. Execute loop at least once.
      - E.g.

```
repeat                            // Pascal
…..
until ( x < 10 );

do{                               // C
…..
}while (x < 10);
```

Control Statements: iteration cont’d

- **Counter controlled loops:**
  - Execution of loop n times
  - Loop variable: memory location where the count value is maintained
  - Loop parameters: initial, terminal and stepsize
  - E.g.
    - Ada

```
for count in 1..10 loop
    sum := sum + count;
end loop;
```

    - Pascal

```
for x := 1 to100 do
for x := 100 downto 1 do
```

    - C/C++/Java

```
for (Index = 0 ; nIndex < 10; nIndex++ )
for (nIndex1 = 0, nIndex2 = 0; nIndex1 < 10 & nIndex2 < 10;
                                              nIndex1++, nIndex2++)
```

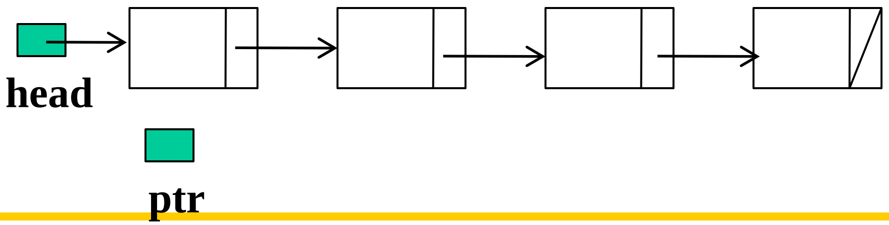
Control Statements: iteration cont'd

- **Data structures controlled loops:**
  - Loop is controlled by the number of elements in a data structure
  - Control mechanism is a call to a function that returns the next element in some chosen order
  - The loop variable is assigned the current element in the data structure
  - Clu was the first to introduce

```
for i in from_to_by(first, last, step) do
   …
end
```

  - C copied

```
for ( ptr=header; ptr != NULL ; ptr=ptr->next ) {
     // C for-loop can be used to create user defined iterator
}
```



Control Statements: iteration cont’d

- **Data structures controlled loops:**
  - Most new languages include this type of loop
  - Enhances readability & writeability

```
// Perl
@names = (“John”, “Ted”, “Lee”);
foreach $name(@name) {
    print $name;
}

#Python
lst = [10,20,30]
for num in lst:
      print num
```

Control Statements: selection

- **Provides the means of choosing between two or more execution paths in a program**

- **Issues:**
  - What is the form and type of the control expression?
  - What is the selectable segment form?
  - How should the meaning of nested selectors be specified?

- **Types:**
  - Single way selection
  - Two-way selection
  - N-way selection

Control Statements: selection cont'd

- **Single way selectors:**
  - If the boolean expression is evaluated to true, do something.
  - E.g.

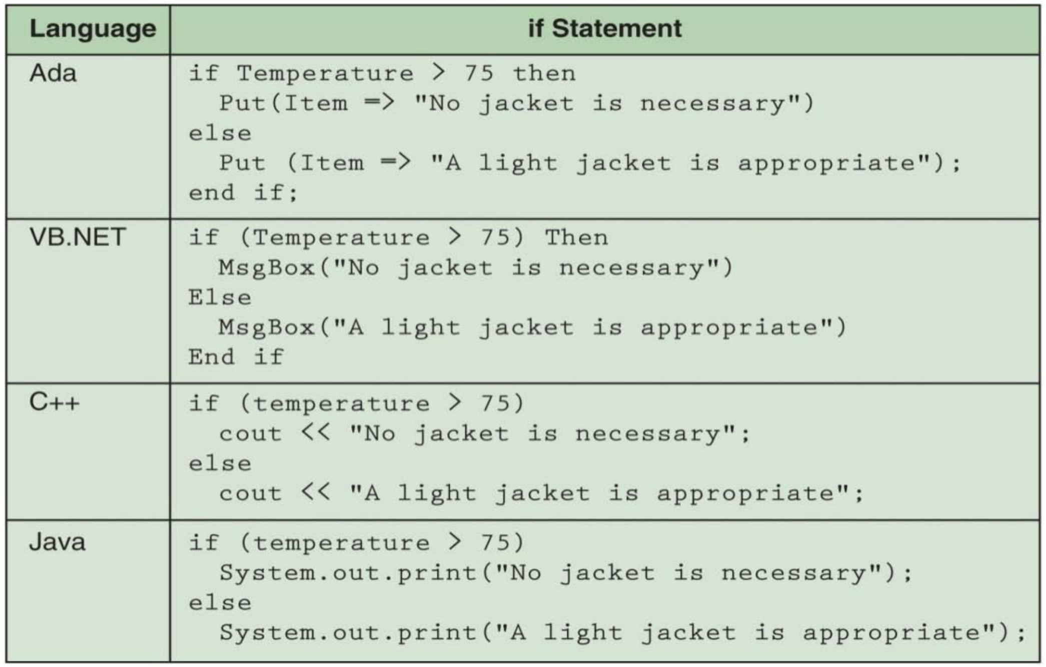
```
IF (x < 10 )      // Fortran I
  print x
```

  - Problem: can select one value only!

Control Statements: selection cont'd

- **Two way selectors:**
  - Pick one out of two execution paths
  - <statement> could be single or compound
  - E.g.

```
if( x < 10 ) then   // Pascal
    <statement>
else
    <statement>
```



Language	if Statement
Ada	`if Temperature > 75 then` `Put(Item => "No jacket is necessary")` `else` `Put (Item => "A light jacket is appropriate");` `end if;`
VB.NET	`if (Temperature > 75) Then` `MsgBox("No jacket is necessary")` `Else` `MsgBox("A light jacket is appropriate")` `End if`
C++	`if (temperature > 75)` `cout << "No jacket is necessary";` `else` `cout << "A light jacket is appropriate";`
Java	`if (temperature > 75)` `System.out.print("No jacket is necessary");` `else` `System.out.print("A light jacket is appropriate");`

Control Statements: selection cont’d

- **N-way selectors:**
  - Pick one out of n execution paths
  - E.g.

```
case index of
    1,3:    statement1;
    2,4:    statement2;
    6..9:   statement3;
    12:     statement4;
    else    statement5
end
```

Pascal

```
switch ( index )
  {
   case 1:
   case 3:  statement1;
            break;
   case 2:
   case 4:  statement2;
            break;
   case 6:
   case 7:
   case 8:
   case 9:  statement3;
            break;
   case 12: statement4;
             break;
   default: statement5;
}
```

C/C++/Java

Control Statements: issues to consider

- **What are the selection statements in the language?**
  - What is the form and type of the expression that controls the selection?
  - Can a single statement, a sequence of statements, or a compound statement be selected?
  - How should unrepresented selector expression values be handled, if at all?
  - Is execution flow through the structure restricted to include just a single selectable segment?

- **What are the repetition statements in the language?**
  - Type and scope of the loop variable?
  - Value of the loop variable at loop termination?
  - Can the loop variable be modified in the loop body?
  - Should the test for loop completion be at the top or bottom of the loop?
  - Should the loop parameters be evaluated only once, or once for every iteration?

What Did We Cover?

- **Logic Language**
  - Prolog
- **Functional languages**
  - Scheme and ML
- **Principles of Imperative Programming Languages**
  - Grammar
  - Data Types
  - Variables, operators & expressions
  - Iteration constructs
  - Branching constructs
  - Subprogram constructs
  - Container constructs

Exam

- 4 Questions (100): Prolog(30), Scheme(20), ML(15), PL Concepts(35)

- 3 Hours. 15 pages (including cover page, empty page and an aid page at the end!)

- Examination Type D:
  Printed lecture slides and textbooks permitted. No other aids are allowed.

- Sample questions. Office hours before exam. Stay tuned!

- Review the programming language (s) you have learned before……☺