Lecture Notes 12

Subprograms

- Subprogram Characteristics – Characteristics of subprograms (coroutines are an exception)
  - Subprograms have single entry point
  - Calling program unit is suspended during execution of subprogram
  - Control returns to the calling program unit when subprogram exits
- Subprogram Definition – Describes the interface and actions of the subprogram abstraction
- Subprogram Call (Activation or Invocation) – Explicit request to execute subprogram
- Active Subprogram – A subprogram that has been called but has not completed execution
- Subprogram Header – Part of the subprogram definition that specifies type of subprogram (if more than one supported), name of the subprogram (if not anonymous), and the subprogram signature
- Parameter Profile – Contains the number, order, and types of formal parameters
- Protocol (Signature or Specification or Interface) – The parameter profile and return type of the subprogram
- Prototype – Subprogram declaration that specifies its protocol
- Formal Parameters (sometimes just called Parameters) – List of names (and possibly types) passed in to the subprogram
- Actual Parameters (or Arguments) – Values bound to the Formal Parameters during a subprogram call
- Positional Parameters – Formal parameters that are bound based upon order of actual parameters
- Keyword Parameters – Actual parameters that are bound to formal parameters using the formal parameter name
  - Example of date subprogram binding year, month, and day parameters using names
    ```
    date(year=2020,month=10,day=3)
    ```
- Function vs Procedure – Function is purely mathematical evaluated for a value, with no side-effects, where procedure does not produce a value, is executed for side-effects
  - Usually lines are blurred between functions and procedures

Subprogram Semantics

- Environment – Determines allocation of memory
- Activation Record (Stack Frame) – Automatic allocation for local objects of procedure block

- Defining Environment (Static Environment, global) – Where the procedure is defined
- Calling Environment (Dynamic Environment) – Where the procedure is called from
- Closure – Subprogram code plus defining environment

Parameter Passing

- Parameter Semantics – Allowed passing of information using parameters
  - In mode – Information is passed into the subprogram
  - Out mode – Information is passed out of the subprogram
  - Inout mode – Information can be both passed in and out of subprogram
- Pass by Value – Formal parameters are replaced by values of actual parameters
  - C and Java – Only pass by value, but parameters are local variables
- Pass by Result – Actual parameter is copied out from the subprogram

```
void foo(int x, int y){
    x = 3;
    y = 4;
}
main(){
    int a;
    foo(a, a);
    // a will be 3 or 4 depending upon order of copying out
}
```

- Pass by Value-Result (Copy-in, Copy-out or Copy-Restore) – Actual parameter is copied in as value, then result is copied back out upon return to caller

```
void foo(int x, int y){
    x++;
    y++;
}
main(){
    int a = 1;
    foo(a, a);
    // a will be 2 after copy-in/copy-out
    // instead of 3 with pass by reference
}
```

- Pass by Reference (Aliasing) – Parameters are an alias for the arguments
  - FORTRAN – Only passes by reference
  - C++ and Pascal – Use & and `var` respectively
- Pass by Name (Delayed Evaluation) – Evaluation of argument is not done until actual use of parameter
  - Thunks – Arguments of Pass by Name (Thought of functions evaluated when parameter referenced)
  - Arrays Have Side-Effects

```
void intswap(int x, int y){
    int t = x;
    x = y;
    y = t;
}
```

```
intswap(i, a[i]);

int sum(int a, int index, int size){
    int temp = 0;
    for(index = 0; index < size; index++){
        temp += a;
    }
    return temp;
}
...
xtotal = sum(x[i], i, 10);
```

Subprograms as Parameters/Variables

- Shallow Binding – The binding environment is the Calling Environment
- Deep Binding – The binding environment is the Defining Environment
- Ad hoc Binding – The binding environment is the environment of call statement that passed the subprogram as a parameter
- Late Binding – The subprogram invoked is not known until runtime
- Function Pointers – Mechanism for indirect subprogram invocation provides late binding (used in C & C++)
- Delegate – A method pointer that has been made an object (method action is delegated)