APP 1: PROCEDURAL ABSTRACTION – PARAMETERISED PERSPECTIVE

ABSTRACTION

Abstraction is a mode of thought by which we concentrate on the general ideas rather than on the specific manifestations of these ideas.

In programming, abstraction refers to the distinction between *what* a piece of code does and *how* it is implemented. For example, in C++ consider the difference between a .h file (what the program does) and a .c file (how the program does it, i.e. the implementation).

At a lower level of programming, consider a function: when we call a function, we are only interested in *what* the function does rather than *how* it does it.

PARAMETER PASSING MECHANISMS

Generally if we simply make an expression or command into a function or procedure, then we construct an abstraction that will always perform (more or less) the same computation whenever called (of course there are exceptions??!!!).

To realize the full power of the abstraction concept we need to parameterize abstractions with respect to values on which they operate.

For Example, consider the following function in C:

```
float circumference(){
   float pi = 3.1416;
   float r = 1.0;
   return 2*pi*r;
}
```

The function call `circumference()` returns the circumference of a circle which has a radius of 1.0. Every time the function is called it is going to return the same value so it is of limited use. We can make the function abstraction more useful by parameterizing it with respect to `r`;

```
float circumference( float r){
   float pi = 3.1416;
   return 2*pi*r;
}
```

We can therefore call this function abstraction and pass it a value to operate on, e.g. `circumference(1.0), circumference(17.2).`

Passing Parameters

Formal Parameters; An identifier used within an abstraction to denote an argument (i.e. the parameter list at the top of a function).

***Actual Parameters*:** An expression (or other phrase) that yields an argument in a function call (i.e. the values passed to a function).

When a function is called, the parameters in the call and in the definition must match, given their types and number as reference. There could be:

- *Correspondence by keyword(specify param_name = value)*
- *Positional correspondence (comma or space separated list of params).*

Issues:

- *Handling unused arguments (implementation concern).*
- *Handling optional parameters.*
- *Handling indefinite number of arguments (e.g. printf in C).*

Copy Mechanisms

When calling a function/procedure actual and formal parameters are matched and perhaps values are copied to/out of the abstraction. There are a number of calling mechanisms used to accomplish this;

a) Call – by – value

- The actual parameter is a value.
- Example: C++

```
void print_value( int a){
  count<<"The value is:"<<a<<endl;
  a=34;
}
```

b) Call – by – result (Reference)

- The actual parameter must be a variable.
- Example: C++

```
void getValue( int &a){
  a=34;
}
```

c) Call – by – value_result

- Combines call – by – value and call – by – result
- The actual parameter must be a variable.
- Example:

```
void print_get_value( int &a){
  count<<"The value is: "<<a<<endl;
  a=34;
}
…….
b = 17;
print_value(b);
print_value(b);
```

This prints "value is 17" first then "value is 34" because the alteration of the formal parameter to 34 is copied back to the actual parameter.

Definition Mechanisms

Allows for formal parameter to be bound directly to the actual parameter

a) **Constant Parameter**
   Actual parameter (AP) can be a value or variable.
   Formal parameter (FP) is bound to AP, no assignment to FP is allowed inside abstraction:

   Example
   {--- pseudo Pascal Example ----}

```
procedure print_value(const a : integer)
begin
   print "value is", a;
   a := 34;
end;
```

b) **Variable (or reference) parameter**
   Also known as call – by – reference.
   AP must be a variable.
   FP renames AP, i.e. the FP is set up as an *alias* for AP.

c) **Procedural/Functional parameter**
   AP must be a procedure/function abstraction.
   FP renames AP
   A mention of FP in the function body is an "indirect" call to the procedure/function AP.