PFI lecture Function Technicalities

return_type function_name (***parameters***) {

function body

}

- The **return_type** is the place holder for:
  - **void** keyword for not returning any value
  - **int** (type keyword) for returning whole number
  - **double** (key keyword) for returning real number approximate
  - **char** (type keyword) for returning a single character
  - **bool** (type keywor) for returning true or false
  - **string** (user defined type) for returning a string
  - any built-in type or user defined type
- Base on what we want a function to accomplish, we choose the return type of the function that we want to write. We did the same in choosing the type of our variables.

PFI lecture Function Technicalities

return_type function_name (***parameters***) {

function body

}

- The **function_name** is the place holder for:
  - Any strings that is valid as a variable name can be used as a function name.
  - Even though a name may be valid, we want to make sure the name will not clash with other names in the same score (global or namespace) with the same signature (function name overloading).
- Base on what we want a function to accomplish, we choose function_name as descriptive as possible so our code is more comprehensible for fellow programmers. We use the same rules for picking variable names.

PFI lecture Function Technicalities

return_type function_name (***parameters***) {

function body

}

- The **parameters** is the place holder for:
  - empty space or void (no parameter is needed)
  - Type-one var-one (e.g. int n) for one parameter
  - type-one var-one, type-two var-two for two parameters
  - type1 var1, type2 var2, …, typek vark for k parameters
- Note: each pair of type and variable name is separated by a comma, not simicolon.
- Note: type is needed only in function definition or declaration. It is not needed in **function call**.

PFI lecture Function Technicalities

return_type function_name (***parameters***) {

function body

}

- The **parameters** basically introduce the “connection variables” that the function may use in the function body. For example, to computer the square root of a value, we need to have a variable holding the value.
- How does a parameter get its value and what does a parameter variable “mean” (semantics)?
- Additional syntax is needed for the different meanings

PFI lecture Function Technicalities

- **Pass-by-value parameters** (this is the only way in C programming language): A parameter is a local variable and has its own memory (a box). Upon a function call **a copy** of the value of actual argument is used to initialize the memory.

```
void by_value(int n){  // function declaration
        cout << n << endl;
        n = n + 1;
        cout << n << endl;
}
int main() {
        by_value(5); // may use literal
        int x = 10;
        by_value(x); // may use a variable
        cout << x << endl; // 10
        by_value(x+7); // may use an expression
        cout << x << endl; // 10
}
```

PFI lecture Function Technicalities

- **Pass-by-reference parameters**: A parameter is basically an alias to the actual argument, which must be a lvalue (memory location). The parameter does not have its own memory (a box). Upon a function call **the parameter** refers to actual argument memory (box).

```
void by_reference(int& n){  // note & after the type int
        cout << n << endl;
        n = n + 1;
        cout << n << endl;
}
int main() {
        by_reference(5);  // error, 5 may not have a memory location
        int x = 10;
        by_reference(x);
        cout << x << endl; // 11
        by_reference(x+7);//error, x+7 does not give a memory location
        int y = 0;
        by_reference(y);
        cout << y << endl; // 1
}
```

PFI lecture Function Technicalities

- **Pass-by-const-reference parameters**: A parameter is basically an alias to the actual argument, which must be a lvalue (memory location). The parameter does not have its own memory (a box). Upon a function call **the parameter** refers to actual argument memory (box). However, in the function body, **no modification to the memory is allowed.**

```
void by_const_ref(const int& n){  // note & after the type int
         cout << n << endl;
         n = n + 1; // error, comment out the line the code compiles
}
int main() {
         by_const_ref(5);  // ok, a temp location is created for 5
         int x = 10;
         by_const_ref(x);
         cout << x << endl;
         by_const_ref(x+7);//a temp location is created for the value of x+7
         int y = 0;
         by_const_ref(y);
         cout << y << endl;
}
```

PFI lecture Function Technicalities

- Use **Pass-by-value parameters** when we are only interested passing values to the parameters.
- Use **Pass-by-reference parameters** when we are interested in modifying the values of the actual arguments.
- Use **Pass-by-const-reference parameters** when we are only interested passing values to the parameters, and it takes more time to copy the values of objects (typically user defined class objects). It is basically for efficiency concerns.

PFI lecture Function Technicalities

- What is the type of an array name where its elements are of certain type? How to we pass an array to a function as a parameter?
- The array name where its elements are of certain type is a constant **pointer** to the type of the elements. The array name contains the address of the first element (index 0) of the array.
- Since the array name is the address of the first element, **pass-by-value of array name let us to modify the array elements**.

PFI lecture Function Technicalities

- An array example

```
void f( int[] a, int size){//note [] after the type int, a is an array
          for (int i=0; i < size; i=i+1)
                    a[i] = a[i] + i;
}
void g(int n){
          n = 77;
}
void h(int& n){
          n = 77;
}
int main() {
          const int size = 10;
          int x[size]={};
          f(x,10); // use size instead of 10 is better, 10 works here
          for (int i=0; i < size; i=i+1)
                    cout << x[i];
          f(x,size);
          // values of x array?
          for (int i=0; i < size; i=i+1)
                    g(x[i]);
          // values of x array?
          for (int i=0; i < size; i=i+1)
                    h(x[i]);
          // values of x array?
}
```