**Array**

T a[s];

* a is and array of s number of T objects
* e.g. int a[10]
  + a is an array of 10 int
* Array declaration, [] is not an subscript operator for this

**Sizeof**

* Is an operator, not a function
* Int x; int g[100];
  + sizeof(int) = 4 bytes
  + sizeof(x) = 4 bytes
  + sizeof(g) = 400 bytes
  + sizeof(g[67]) = 4 bytes

**Function Declaration prototype**

* To inform about function name
* Return type and the various types of each parameters

**Function Declaration syntax**

* Type functionName(type <id>, … ); - semi-colon
* E.g. int foo(short, double)
  + foo is a function that takes in a short and a double and return an int
  + parameters name is not necessary, it is just a declaration

**Function Definition syntax**

* Type functionName(type <id>, … ) { } –a bracers as a body function
* E.g. int foo(short a, double b)
  + parameters name is a must

**Encapsulation**

* Function call (is an operator)
  + An expression when .exe invoke the execution of the desired function
* **Pass-by value semantics**
  + Copy values
* **Pass by reference**
  + Same values

Function Declaration: syntax:

- Bracers must not exist

- Have semi-colon

<return type> func\_name(<param Type> paramName\_maynotExist, ...);

Declaration:

void func\_no\_param();

int func\_with\_two\_param(int\* param1, int param2);

float func\_with\_no\_name(double, int, float);

Tips:

- Declaration parameters name is optional,

- in some projects, parameters name is important to represent which parameters coorrespond to which

- in some projects, not writing parameters name is to hidden from user

-------------------------------------------------------------------------

Function Definition syntax:

- Bracers must exist to indicate that it’s a definition

- Parameters name must exist, obviously, or else how are you going to represent that parameter type?

- No semi-colon

<return type> func\_name(<param Type> paramName\_must\_exist, ...)

{

return <return type>;

//return; in the case of type <void>

}

Definition:

void func\_no\_param(){}

int func\_with\_two\_param(int param1, int param2){

return 1;

}

float func\_with\_two\_param(int param1, int param2){

return 1.0f;

}

float func\_with\_two\_param(int\* param1, int param2){

return 1.0f;

}

-------------------------------------------------------------------------

Tips:

- In practice, try to keep 1 function with only 1 purpose

- function is used to unify similiar codes and for easy maintainence

- Is used to compute hidden things from users who have no access to the definition

- Keep in mind function calls is considered an expensive operations if you learn assembly, you know what I mean, but do not let this weight you down