C++

references

What is a reference

- A **reference** is an alternative name (alias) for an object.
- Reference is usually found in parameter passing to functions:

```
int swap(int& i, int& j) {
  int tmp = i;
  i = j;
  j = tmp;
}
```

```
int main (void) {
  int i = 3;
  int j = 5;
  swap(i, j); // now i is 5 and j 3
}
```


Creation of reference variable

- The notation “int& r = j;” means that r is a reference to j (another name for j)
- A reference allows indirect manipulation of an object, *“like”* pointers, with a simplified syntax
- A reference must always be bound to an object so it must be initialized when it is created:

```
int i = 1;
int& j = i; // ok
int& k;     // not ok. Compile time error
```

- Unlike a pointer, in C++, a reference can NEVER be reseated (redirected). It ALWAYS refers to the same object it was first initialized to.
- Note: the value of the object, that the reference refers to, can change

Initialization and assignment

-
```
int j = 1;
int& k = j;  // k and j are eq. to 1
j += 10;     // k and j are eq. to 11
k = 2;       // k and j are eq. to 2
```

- Assignment to a reference only changes the value of the referenced object
- Note: the address of the referenced object and the reference are always equal: &k == &j (see above code) is always true

Address of and reference

- & can be used to get the address of a variable or can be used in a reference assignment. Do not confuse them.
- float i;
  float* j = &i; // j is pointer storing the address of i

  float& j = i; // j is a reference (another name) to i

Call by reference

- A common usage of reference is passing parameters to functions
- Example:
  void inc(int& i) { i++; }
  int main() { int j=1; inc(j); std::cout << j << std::endl; }
- In this example, the var i in inc() is a reference to the var passed as argument
- Any change to i in inc() will appear also outside of inc. In the example above, j will be increased in main()

Call by reference

- In C, function calls are by value. If you want to change the content of parameters inside a function, you need to pass pointers.
- In C++ you have the choice between call by value (like in C) or call by reference (like seen previously)

Why use call by reference

- For side effects: if you want the function to be able to change the value of the passed arguments.
- For efficiency: when passing arguments by value, the function gets in fact a local copy of the arguments. In the case of big objects (passed as arguments), the copy can be expensive. Passing an object by reference to a function is cheap because it does not require a copy to be created (only the memory address is passed).

Differences between pointers and references

- A reference is always bound to an object, not a pointer
- A reference CAN NOT be reseated (can not change which object it refers to). Pointer can point to different objects at different times
- Syntax of reference is more convenient. There is no need of dereferencing operators (*, ->, …)

When to use reference ?

- Use references when you can and use pointers when you have to
- References should be preferred if you do not need reseating
- Practically you will probably use references in public interfaces while pointers will be used in the (hidden) implementation of the class