그런즉 너희가 먹든지 마시든지 무엇을 하든지 다 하나님의 영광을 위하여 하라 (고전 10:31)

<img src="https://github.com/idebtor/nowic/blob/85a7852bf0f6000fded586dadaeac02c186027a6/images/cplus_ds_title.jpg?raw=true" width=1000>

__NOTE:__ The following materials have been compiled and adapted from the numerous sources including my own. Please help me to keep this tutorial up-to-date by reporting any issues or questions. Send any comments or criticisms to `idebtor@gmail.com` Your assistances and comments will be appreciated.

# Lab3 - References in C++
## Subjects
    1. reference variable
    2. pass by value
    3. pass by pointer
    4. pass by reference
    5. return by reference

C++ reference is a major concept in C++ programming language. Although it’s not as strong as pointers, nonetheless it allows us to use it to write efficient programs.

## 1. Reference variable
C++ reference allows you to create a second name (or __alias__) for the a variable that you can use to read or modify the original data stored in that variable. This means that when you declare a reference and assign it a variable, it will allow you to treat the reference exactly as though it were the original variable for the purpose of __accessing and modifying the value of the original variable__ even if the second name (the reference) is located within a different scope. This means, for instance, that if you __make your function arguments references__, and you will effectively have a way to change the original data passed into the function.

## 2. Pass by value
"Pass" means to provide an argument to a function. "Pass by value" is the most straightforward way to pass parameters. When a function is invoked, the arguments are copied to the local scope of the function.

## 3. Pass by pointer
"Pass by pointer" means to pass a pointer argument in the calling function to the corresponding formal parameter of the called function. The called function __can modify the value of the variable to which the pointer argument points.__
When you use pass-by-pointer, a copy of the pointer is passed to the function. If you modify the pointer inside the called function, you only modify the copy of the pointer, but the original pointer remains unmodified and still points to the original variable.

## 4. Pass by reference
"By reference" means that the argument you’re passing to the function is __a reference to a variable that already exists in memory__ rather than an independent copy of that variable. Since you are giving the function a reference to an existing variable, all operations performed on this reference will directly affect the variable to which it refers.
This is __powerful__ because it allows us to pass large objects into functions by giving another name to the object and not copying the large object, while also allowing us __avoid the hairy complications of passing by pointer__.
A reference variable can never point to null so there doesn’t need to be a null check in every function, and since a reference variable is not heap allocated, there is no worry of a memory leak or figuring out who is responsible for owning and deleting the object.

## 5. Return by reference
Just like returning pointers from functions, we can also __return references from functions.__ Returning references from a function also return an implicit pointer to the return value. Because of this reason, a function returning a reference can be used on __the left-hand side of the assignment.__

## Step 1. Reference variable
1. Create a source file, `times.cpp`, with the following code.
    - Multiply even number elements in the `list` by 10.
    - Use ranged-for loop and a reference variable.
    - With the macro `#if 1`, it will use the code between `#if 1` and `#else`, and ignore the rest of the code.
    - With the macro `#if 0`, it will use the code between `#else` and `#endif`, and ignore the upper portion of the code.

```
    #include<iostream>
    #include<vector>
    using namespace std;

    #if 1
    // multiply each element in the list by 10 if it is an even number
    // without using reference variable.
    int main(int argc, char *argv[]) {
        vector<int> list = { 0, 1, 2, 2, 4, 5, 6, 7, 8, 8, 10 };

        for (size_t i = 0; i < list.size(); i++) {
            cout << "your code here\n";
        }

        for (auto x: list) cout << x << " ";
        return 0;
    }
    #else
    // multiply each element in the list by 10 if it is an even number
    // using reference variable.
    int main(int argc, char *argv[]) {
        vector<int> list = { 0, 1, 2, 2, 4, 5, 6, 7, 8, 8, 10 };

        cout << "your code here\n";

        for (auto x: list) cout << x << " ";
        return 0;
    }
    #endif
```

#### Sample Run:
```
$ g++ -std=c++11 times.cpp -o times
$ ./times
0 1 20 20 40 5 60 7 80 80 100
```

## Step 2. Pass by value

1. Create a source file, `setmax_val.cpp`, with the following code.
    - Use pass-by-value.
    - With a given list, find the max value, then set it to 99 in setmax().
    - Observe that the max value is not changed in main(0) even though the change was made in setmax(). The reason is that functions are using pass-by-value on purpose.

```
#include<iostream>
#include<vector>
#include<cassert>
using namespace std;
```

```
// gets the max value in the list and returns its index
int getmax(vector<int> vec) {
    assert(vec.size() > 0);
    auto max = vec[0];
    size_t idx = 0;
    for (size_t i = 0; i < vec.size(); i++) {
        cout << "your code here\n";
    }
    return idx;
}

// sets the max value in the list to 99
void setmax(vector<int> vec) {
    size_t idx = getmax(vec);
    cout << "your code here\n";
}

// With a given list, find the max value, then set it to 99 in setmax()
// In main(), we do not see the max value change that was made in setmax(),
// since functions are using pass-by-value on purpose.
int main(int argc, char *argv[]) {
    vector<int> list1 = {43, 10, 20, 75, 22, 33};
    vector<int> list2 = {33, 13, 45, 19, 39, 22};

    cout << ">list1: ";
    for (auto x: list1) cout << x << " ";
    cout << endl;
    setmax(list1);
    cout << "<list1: ";
    for (auto x: list1) cout << x << " ";
    cout << endl << endl;

    cout << ">list2: ";
    for (auto x: list2) cout << x << " ";
    cout << endl;
    setmax(list2);
    cout << "<list2: ";
    for (auto x: list2) cout << x << " ";
    cout << endl;

    return 0;
}
```

#### Sample Run:
```
$ g++ setmax_val.cpp -o setmax_val
$ ./setmax_val
>list1: 43 10 20 75 22 33
<list1: 43 10 20 75 22 33

>list2: 33 13 45 19 39 22
<list2: 33 13 45 19 39 22
```

## Step 3. Pass by pointer
1. Create a source file, `setmax_ptr.cpp`, with the following code.
    - With a given list, it finds the max value, then set it to 99.
    - Pass the list by pointer to `setmax()` function.

```
// gets the max value in the list and returns its index
```

```
int getmax(vector<int> vec) {
    assert(vec.size() > 0);
    auto max = vec[0];
    size_t idx = 0;
    for (size_t i = 0; i < vec.size(); i++) {
        cout << "your code here\n";
    }
    return idx;
}

// sets the max value to 99
// your code here - define setmax() here

// With a given list, find the max value, then set it to 99.
int main(int argc, char *argv[]) {
    vector<int> list1 = {43, 10, 20, 75, 22, 33};
    vector<int> list2 = {33, 13, 45, 19, 39, 22};

    cout << ">list1: ";
    for (auto x: list1) cout << x << " "; cout << endl;
    cout << "your code here - invoke setmax()" << endl;
    cout << "<list1: ";
    for (auto x: list1) cout << x << " ";
    cout << endl << endl;

    cout << ">list2: ";
    for (auto x: list2) cout << x << " "; cout << endl;
    cout << "your code here - invoke setmax()" << endl;
    cout << "<list2: ";
    for (auto x: list2) cout << x << " ";
    cout << endl;

    return 0;
}
```

#### Sample Run:
```
$ g++ setmax_ptr.cpp -o setmax_ptr
$ ./setmax_ptr
>list1: 43 10 20 75 22 33
<list1: 43 10 20 99 22 33

>list2: 33 13 45 19 39 22
<list2: 33 13 99 19 39 22
```

## Step 4. Pass by reference
1. Create a source file, `setmax_ref.cpp`, with the following code.
    - With a given list, it finds the max value, then set it to 99.
    - Coding `setmax()` function only will be enough for this step.
    - The `list` argumnet in `setmax()` is called `pass by reference`.

```
// gets the max value in the list and returns its index
int getmax(vector<int> vec) {
    assert(vec.size() > 0);
    auto max = vec[0];
    size_t idx = 0;
    for (size_t i = 0; i < vec.size(); i++) {
        cout << "your code here\n";
    }
    return idx;
}

// sets the max value to 99
// your code here - define setmax() here

// With a given list, find the max value, then set it to 99.
int main(int argc, char *argv[]) {
    vector<int> list1 = {43, 10, 20, 75, 22, 33};
    vector<int> list2 = {33, 13, 45, 19, 39, 22};

    cout << ">list1: ";
    for (auto x: list1) cout << x << " ";
    cout << endl;
    setmax(list1);
    cout << "<list1: ";
    for (auto x: list1) cout << x << " ";
    cout << endl << endl;

    cout << ">list2: ";
    for (auto x: list2) cout << x << " ";
    cout << endl;
    setmax(list2);
    cout << "<list2: ";
    for (auto x: list2) cout << x << " "; c
    out << endl;

    return 0;
}
```
#### Sample Run:
```
$ g++ setmax_ref.cpp -o setmax_ref
$ ./setmax_ref
>list1: 43 10 20 75 22 33
<list1: 43 10 20 99 22 33

>list2: 33 13 45 19 39 22
<list2: 33 13 99 19 39 22
```

## Step 5. Return by reference
1. Create a source file, `setmax_ret.cpp`, with the following code.
    - With a given list, it finds the max value, then set it to 99.
    - Code `getmax()` such that it __returns__ the max element itself __by reference__, but not the index of the max element.
    - Code `setmax()` such that it uses the returned reference as a lvalue reference. Recall that a __function__ returning a reference can be used on __the left-hand side of the assignment.__
    - Make sure that the `list` argument in both `getmax()` and `setmax()` should be pass-by-reference.

```
// gets the max value in the list and returns the reference of the max value
// your code here - define getmax() here

// sets the max value to 99
// your code here - define setmax() here

// With a given list, find the max value, then set it to 99.
int main(int argc, char *argv[]) {
    vector<int> list1 = {43, 10, 20, 75, 22, 33};
    vector<int> list2 = {33, 13, 45, 19, 39, 22};
    cout << ">list1: ";
    for (auto x: list1) cout << x << " ";
    cout << endl;
    setmax(list1);
    cout << "<list1: ";
    for (auto x: list1) cout << x << " ";
    cout << endl << endl;

    cout << ">list2: ";
    for (auto x: list2) cout << x << " ";
    cout << endl;
    setmax(list2);
    cout << "<list2: ";
    for (auto x: list2) cout << x << " ";
    cout << endl;

    return 0;
}
```
#### Sample Run:
```
$ g++ setmax_ret.cpp -o setmax_ret
$ ./setmax_ret
>list1: 43 10 20 75 22 33
<list1: 43 10 20 99 22 33

>list2: 33 13 45 19 39 22
<list2: 33 13 99 19 39 22
```

## Step 6. touppers.cpp
The C/C++ language do offer a function, `toupper()` to convert a character to an uppercase, but not for a word or a string. Write a function called `touppers()` that converts a string to all uppercases in place where it does not require extra memory.

- Use a skeleton file, `touppers.cpp`, provided.
- Use a reference variable and `toupper()`, but not functions in <algorithm>.

Sample Run:
> g++ touppers.cpp -o touppers
> ./touppers
> Enter words: Hello Mr. Kim
> HELLO MR. KIM

# Files to submit for this Lab:
- times.cpp
- setmax_val.cpp
- setmax_ptr.cpp
- setmax_ref.cpp
- setmax_ret.cpp
- touppers.cpp

One thing I know, I was blind but now I see. John 9:25
