Variable Scope and Program

- L value - Associated with non-temporary objects
- `string str = "hello";`
- `str` - L Value
- `"Hello"` - R Value, Temporary

```
string x= findMax(a);
string & y = x;
cout << y << endl;
```

- R values can be moved (we do not need the value)
- L values can be copied

```
// Refrence to an R Value
string && str2 = "Hello";
```

Parameter Passing

Call by Value

- Copies the value of the parameter being passed
- Called function can modify the parameter but not the initial

Pass by Reference

- Can modify the original value
- Faster because you dont need to make a copy

Pass by Reference

- Cannot modify the value
- Should be used for large values

Call by rvalue refrence

- Move rvalue instead of copy
- Normally more efficient

```
vector <string> v("hello", "world");
cout << randomItem(v) << endl;  // L Value
cout << randomItem({"Hello", "World"}) << endl; // R Value
```

Given File Example

```
#include <iostream>
#include <vector>
#include <string.h>

using namespace std;


double ave( const vector<int> & arr, int n, bool & errorFlag)
{

    int sum = 0;
        for( int i = 0; i < arr.size( ); ++i )
          sum += arr[i];

        n = 100;
        errorFlag = true;
        return ((double)sum)/arr.size();
}


int main( )
{
  int nn = 5;
  bool err = false;

  vector<int> myArray {1, 2, 3, 4, 5};
  double average = 0.0;

  cout << "Before: average = " << average << ", nn = " << nn << ", err = "
<< err << endl;

  average = ave(myArray, nn, err);

  cout << " After: average = " << average << ", nn = " << nn << ", err = "
<< err << endl;

  return 0;
}
```

Return Passing

Return by Value

- makes a copy of a variable returned

Return by Reference

- Return a reference of the variable returned

Return by Const Reference

- Return the reference of the variable returned
- Return value cannot be modified by caller

Lifeline extended beyond function call for by const reference and reference.

Big Five in C++

- Five special functions provided in all c++ classes
  - Destructor constructor
  - Copy constructor
  - Move constructor
  - Copy assignment operator =
  - Move assignment operator =

A constructor is called whenever..

- An object goes out of scope
- Delete is called
  Invoked during
- Declaration
- Call by value, and return by value
- Not in Assignment operator

Problem with Defaults

- Usually dont work when data member is a pointer type
- If a class contains pointers as member variables and you want two copies of objects pointed at

Variable Scope

- If your program sometimes doesn't work. Then...
  You Have Bugs

- a local variable only exists within its scope

Templates

- Type independent patterns
- Allows for reusable code, and generic programming
- The template declaration indicates that Comparable is the template argument it can be replaced by any type to produce a real function.

```
// Return the maximum item in the array a
template<typename Comparable>
const Comparable& findMax(const vector<Comparable>& a){
        int maxIndex = 0;
        for(int i= 1; i < a.size(); i++){
                if(a[maxIndex] < a[i]){
                        maxIndex = i;
                }
        }
        return a[maxIndex];
}
```

For Example

- If a user needs to use the same function to hold a string, as well as integers. A Code example can be seen above.
- Also covered in COP3330 notes

  ### Function Objects
- Objects whose primary purpose is to define a function
- Using operator overloading : `operator ()`

  ### Memory Cell
- Can be used for any type object

#code #functions #pass-by-ref #assignment #l-value #r-value #defaults #memberFunctions #memberVariables #declaration #cpp