2 Call-by-Reference Parameters

- Call-by-value means that the formal parameters receive the values of the arguments.
  - When the values of formal parameters are changed in a function body, the arguments in the function call won't be changed.
- Call-by-reference parameters allow us to change the variable used in the function call
  - Arguments for call-by-reference parameters must be variables, not numbers

2.1 Call-by-Reference Example

```
void get_input(double& f_variable)
{
        using namespace std;
        cout << " Convert a Fahrenheit temperature"
                << " to Celsius.\n"
                << " Enter a temperature in Fahrenheit: ";
        cin >> f_variable;
}
```

- '&' symbol (ampersand) identifies **f_variable** as a call-by-reference parameter
  - Used in both declaration and definition!

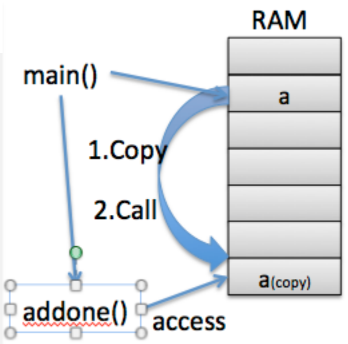
2.1.1 Call-By-Reference Details

- Call-by-reference works almost as if the argument variable is substituted for the formal parameter, not the argument's value.
- In reality, the memory location of the argument variable is given to the formal parameter.
  - Whatever is done to a formal parameter in the function body, is actually done to the value at the memory location of the argument variable.

2.2 Call Comparisons: Call By Reference vs Value

2.2.1 sample10.cpp (Call-By-Value)

```cpp
#include <iostream>

using namespace std;

int addone(int a){
  cout << "a=" << a << endl;
  a += 1; // add 1
  cout << "a+1=" << a << endl;
  return a;
}

int main (int argc, char *argv[]) {
  int a = 5;
  int b = addone(a);

  cout << "a=" << a << " b=" << b << endl;

  return 0;
}
```

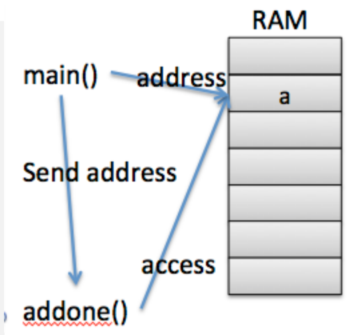


You can change the value of the input variable but that does not affect the original variable.

```
a=5
a+1=6
a=5  b=6
```

2.2.2 sample11.cpp (Call-By-Reference)

```cpp
#include <iostream>

using namespace std;

int addone(int &a){
  cout << "a=" << a << endl;
  a += 1; // add 1
  cout << "a+1=" << a << endl;
  return a;
}

int main (int argc, char *argv[]) {
  int a = 5;
  int b = addone(a);

  cout << "a=" << a << " b=" << b << endl;

  return 0;
}
```



If you change the value of the input variable, the value of the original variable is changed.

```
a=5
a+1=6
a=6  b=6
```

2.3 Mixed Parameter Lists

- Call-by-value and call-by-reference parameters can be mixed in the same function
- Example: `void good_stuff(int& par1, int par2, double& par3);`
  - par1 and par3 are call-by-reference formal parameters
    - Changes in par1 and par3 change the argument variable
  - par2 is a call-by-value formal parameter
    - Changes in par2 do not change the argument variable

2.4 Choosing Parameter Types

- How do you decide whether a call-by-reference or call-by-value formal parameter is needed?
  - Does the function need to change the value of the variable used as an argument?
  - Yes? Use a call-by-reference formal parameter
  - No? Use a call-by-value formal parameter
- Function need to return more than one values.

```
#include <iostream>
#include <cmath>

using namespace std;

void rotate(double a, double &x1,double &y1){
  double xp = x1 * cos(a) - y1 * sin(a);
  double yp = x1 * sin(a) + y1 * cos(a);
  x1 = xp;
  y1 = yp;
}

int main (int argc, char *argv[]) {
  double x1 = 1.0;
  double y1 = 0.0;

  cout << "(" << x1 << "," << y1 << ")" << endl;

  rotate(M_PI / 4.0, x1 , y1);
  cout << "(" << x1 << "," << y1 << ")" << endl;
  :
  :
  return 0;
}
```