Functions
Call by value & Call by reference

Call by Value

- When an argument is passed into a parameter, only a copy of the argument's value is passed. Changes to the parameter do not affect

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

void increment(int n)
{
  n++;
  cout << "\tn inside the function is " << n << endl;
}

int main()
{
  int x = 1;
  cout << "Before the call, x is " << x << endl;
  increment(x);
  cout << "after the call, x is " << x << endl;

  return 0;
}
```

```
Before the call, x is 1
  n inside the function is 2
after the call, x is 1
```

Call by Reference

- When used as parameters, reference variables allow a function to access he parameter’s original argument. Changes to the parameter are also made to the argument.
- A reference variable is an alias for another variable. Any changes made to the reference variable are actually performed on the variable for which it is an alias.
- Reference variables are defined like regular variables, except you place an ampersand (&) in front of the name. For example, the following function definition makes the parameter refVar a reference variable:

```
void doubleNum(int &refVar)
{
refVar *= 2;
}
```

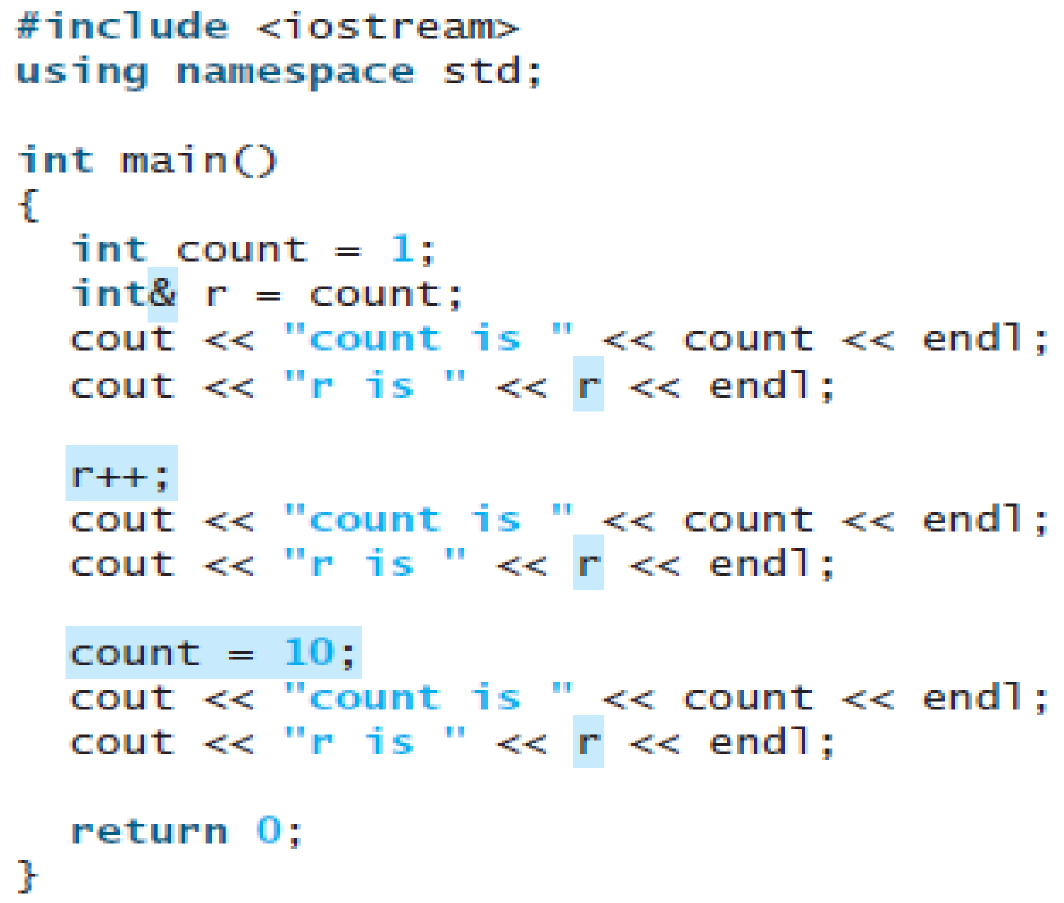
Call by Reference

- Some programmers prefer not to put a space between the data type and the ampersand. The following prototype is equivalent to the one above:

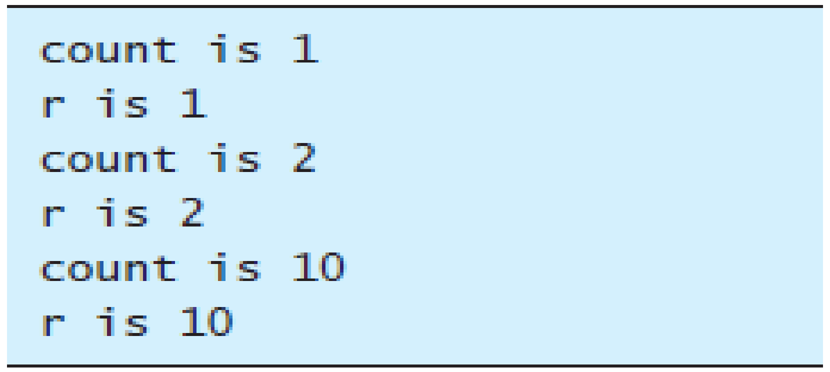
  ```
  void doubleNum(int &);
  ```

- The ampersand must appear in both the prototype and the header of any function that uses a reference variable as a parameter. It does not appear in the function call.

Understanding Reference Variables

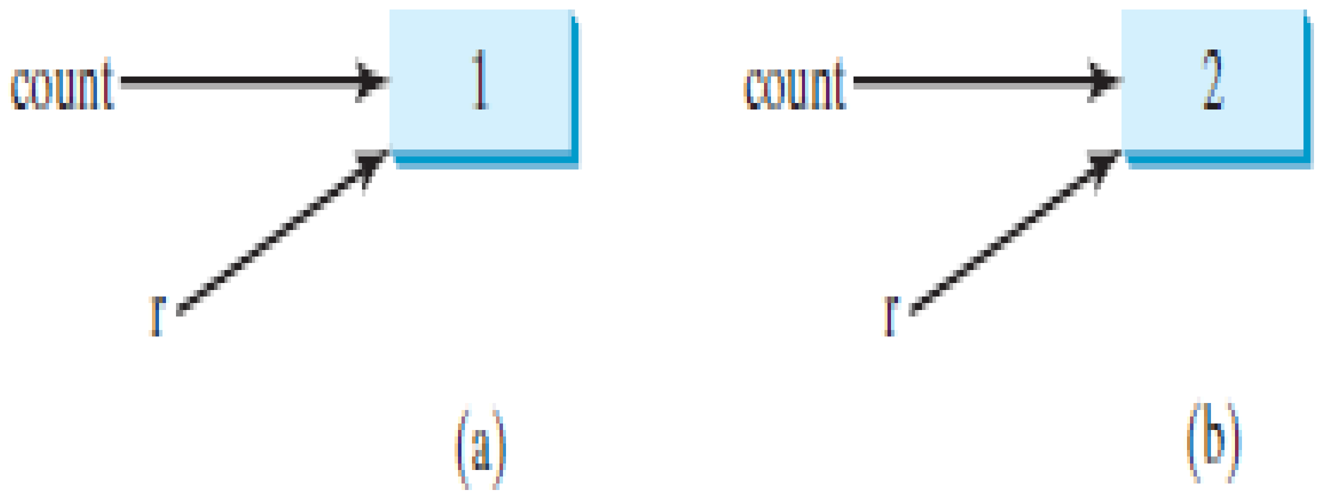


```cpp
#include <iostream>
using namespace std;

int main()
{
  int count = 1;
  int& r = count;
  cout << "count is " << count << endl;
  cout << "r is " << r << endl;

  r++;
  cout << "count is " << count << endl;
  cout << "r is " << r << endl;

  count = 10;
  cout << "count is " << count << endl;
  cout << "r is " << r << endl;

  return 0;
}
```



```
count is 1
r is 1
count is 2
r is 2
count is 10
r is 10
```

Reference Variables



r and count share the same value.

• Example: Pass by Reference

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

void increment(int& n)
{
  n++;
  cout << "n inside the function is " << n << endl;
}

int main()
{
  int x = 1;
  cout << "Before the call, x is " << x << endl;
  increment(x);
  cout << "After the call, x is " << x << endl;

  return 0;
}
```

```
Before the call, x is 1
  n inside the function is 2
After the call, x is 2
```

When you pass an argument by reference, the argument must be a variable. When you pass an argument by value, the argument can be a literal, a variable, an expression, or even the return value of another function.

- Activity: What is pass-by-value? What is pass-by-reference? Show the result of the following programs:

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

void maxValue(int value1, int value2, int max)
{
  if (value1 > value2)
    max = value1;
  else
    max = value2;
}

int main()
{
  int max = 0;
  maxValue(1, 2, max);
  cout << "max is " << max << endl;

  return 0;
}
```

(a)

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

void maxValue(int value1, int value2, int& max)
{
  if (value1 > value2)
    max = value1;
  else
    max = value2;
}

int main()
{
  int max = 0;
  maxValue(1, 2, max);
  cout << "max is " << max << endl;

  return 0;
}
```

(b)

- Activity 2

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

void f(int i, int num)
{
  for (int j = 1; j <= i; j++)
  {
    cout << num << " ";
    num *= 2;
  }

  cout << endl;
}

int main()
{
  int i = 1;
  while (i <= 6)
  {
    f(i, 2);
    i++;
  }

  return 0;
}
```

(c)

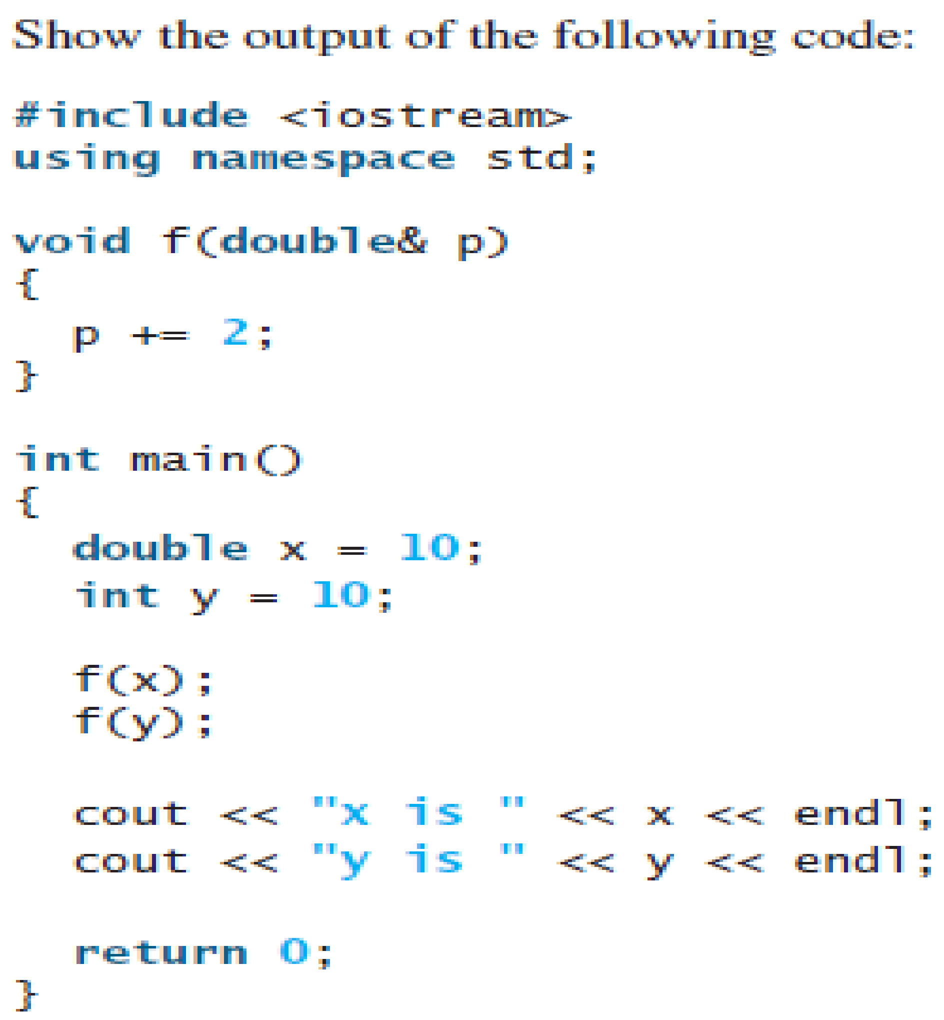
```
#include <iostream>
using namespace std;

void f(int& i, int num)
{
  for (int j = 1; j <= i; j++)
  {
    cout << num << " ";
    num *= 2;
  }

  cout << endl;
}

int main()
{
  int i = 1;
  while (i <= 6)
  {
    f(i, 2);
    i++;
  }

  return 0;
}
```

(d)

A student wrote the following function to find the minimum and maximum number between two values a and b. What is wrong in the program?

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

void minMax(double a, double b, double& min, double& max)
{
  if (a < b)
  {
    double min = a;
    double max = b;
  }
  else
  {
    double min = b;
    double max = a;
  }
}

int main()
{
  double a = 5, b = 6, min, max;
  minMax(a, b, min, max);

  cout << "min is " << min << " and max is " << max << endl;

  return 0;
}
```

- Activity 4



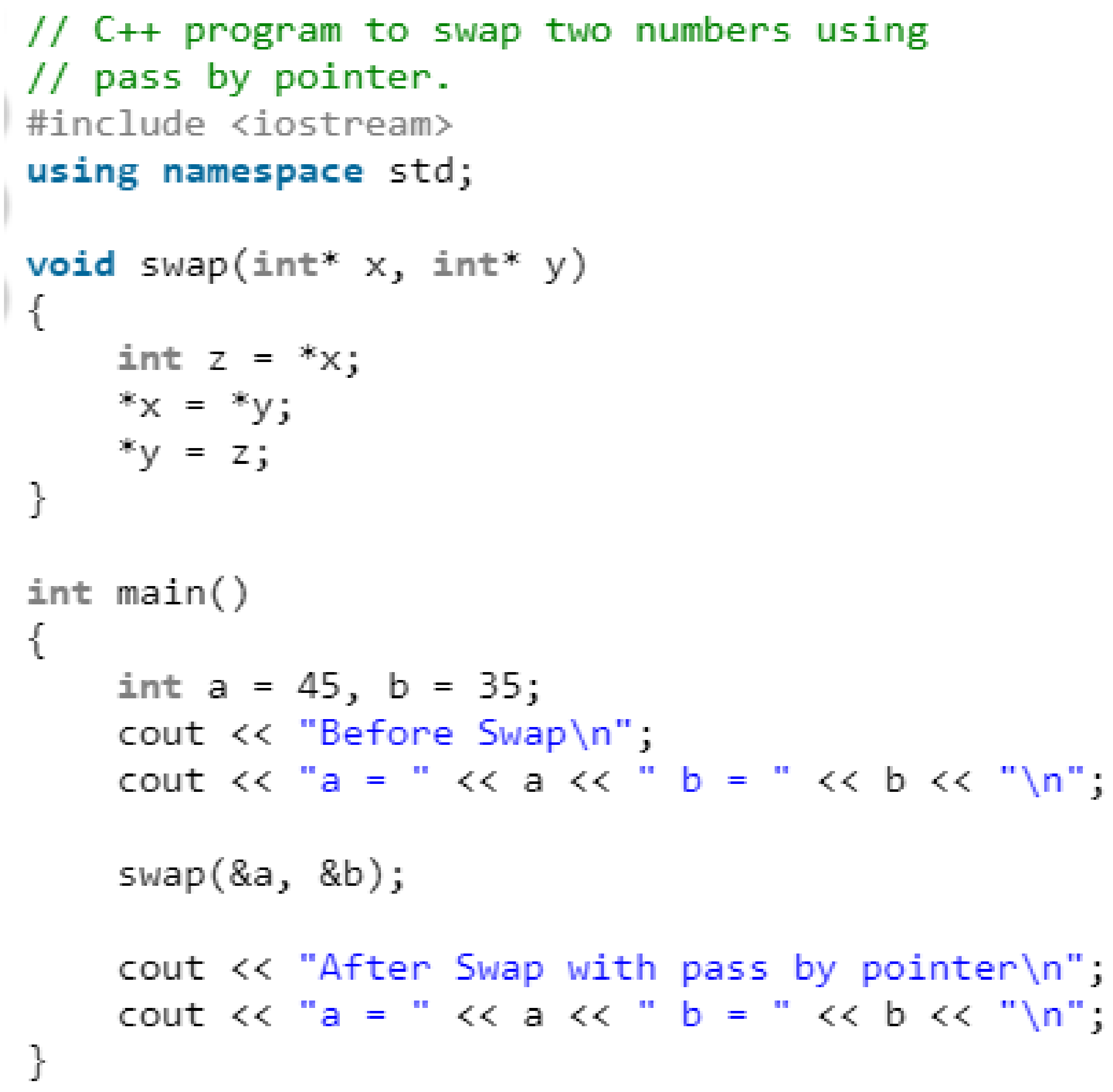
Show the output of the following code:

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

void f(double& p)
{
  p += 2;
}

int main()
{
  double x = 10;
  int y = 10;

  f(x);
  f(y);

  cout << "x is " << x << endl;
  cout << "y is " << y << endl;

  return 0;
}
```

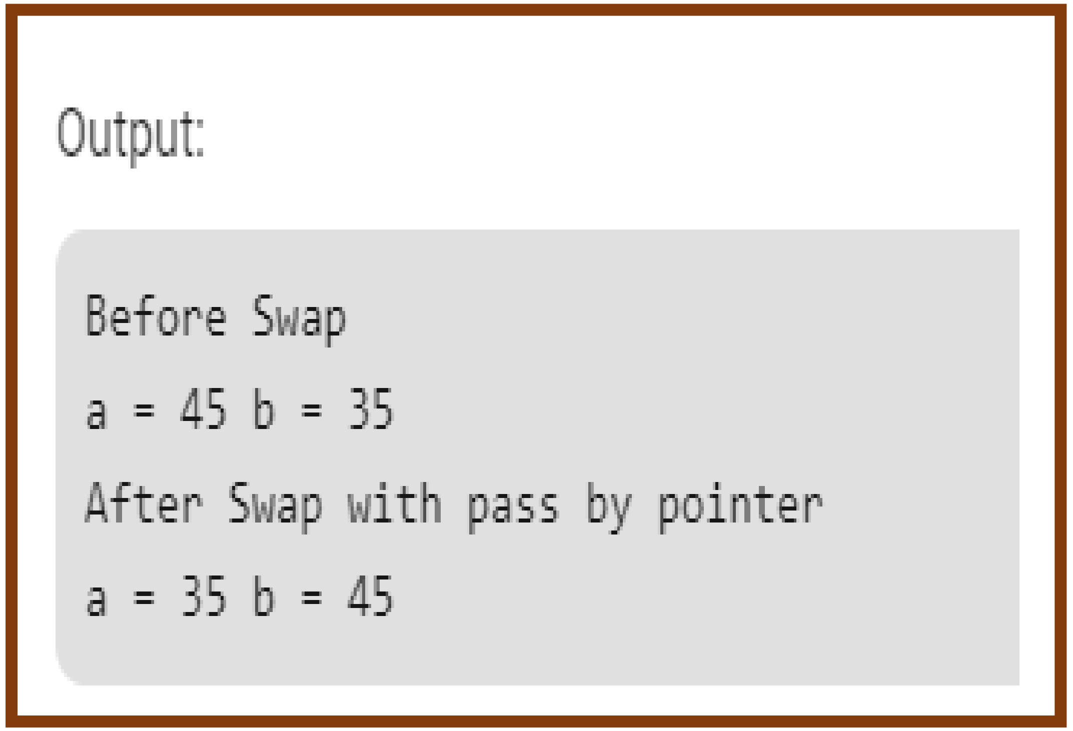
Pass by Pointer

- In C++, we can pass parameters to a function either by pointers or by reference. In both the cases, we get the same result. So the following questions are inevitable; when is one preferred over the other? What are the reasons we use one over the other?

Example



```cpp
// C++ program to swap two numbers using
// pass by pointer.
#include <iostream>
using namespace std;

void swap(int* x, int* y)
{
    int z = *x;
    *x = *y;
    *y = z;
}

int main()
{
    int a = 45, b = 35;
    cout << "Before Swap\n";
    cout << "a = " << a << " b = " << b << "\n";

    swap(&a, &b);

    cout << "After Swap with pass by pointer\n";
    cout << "a = " << a << " b = " << b << "\n";
}
```



Output:

```
Before Swap
a = 45 b = 35
After Swap with pass by pointer
a = 35 b = 45
```

Difference in Reference variable and pointer variable

1. A pointer can be re-assigned while reference cannot, and must be assigned at initialization only.
2. Pointer can be assigned NULL directly, whereas reference cannot.
3. Pointers can iterate over an array, we can use ++ to go to the next item that a pointer is pointing to.
4. A pointer is a variable that holds a memory address. A reference has the same memory address as the item it references.
5. A pointer needs to be dereferenced with * to access the memory location it points to, whereas a reference can be used directly.

```
int main()
{
   int x = 5;
   int y = 6;


   int *p;
   p =  &x;
   p = &y;                    // 1. Pointer reintialization allowed
   int &r = x;
   // &r = y;                 // 1. Compile Error
   r = y;                    // 1. x value becomes 6

   p = NULL;
   // &r = NULL;              // 2. Compile Error

   p++;                      // 3. Points to next memory location
   r++;                      // 3. x values becomes 7

   cout << &p << " " << &x << endl;    // 4. Different address
   cout << &r << " " << &x << endl;    // 4. Same address

   cout << p << endl;       // 6. Prints the address
   cout << r << endl;       // 6. Print the value of x

   return 0;
}
```