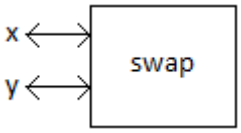
Passing Arguments by Value and by Reference

I. The general problem

A. Sometimes it might be useful if a function could modify the function arguments

```
int a = 2, b = 3;
Swap(a, b);

// a and b have swapped values
cout << a << b;   // 3 and 2
```



B. Suppose this function was written like so:

```
void Swap(int x, int y)
{
        int temp = x;
        x = y;
        y = temp;
}
```

1. Although this function does swap the values of `x` and `y`, the values of `a` and `b` in the function call would remain unchanged

```
int a = 2, b = 3;
Swap(a, b);

// a and b have NOT swapped values!
cout << a << b;   // 2 and 3
```

2. That's because the arguments are being passed by value

II. Pass by value

A. By default, arguments are passed by value to a function which means a copy is made of the argument

B. Any changes made to a parameter are only changing the copy, not the actual argument

C. It's like handing someone a photocopy of a paper you wrote; any changes the person makes to the photocopy doesn't change the original paper

III. Pass by reference

A. To make the `Swap` function work, we need to put `&` in front of the parameter names so they become **reference parameters**

```
void Swap(int &x;, int &y;)
{
        int temp = x;
        x = y;
        y = temp;
}
```

B. This means the parameter is just another name for the argument

C. Therefore any changes made to the reference parameter will also change the argument (modifying `x` will change `a`, and modifying `y` will change `b`)

D. It's like handing someone the original paper you wrote; any changes the person makes to the original will affect the original!

E. When using pass by reference, a variable must be used as the argument

```
// Won't work because literals cannot change value or be passed by reference!
Swap(2, 3);

// This works because variables are used as arguments
Swap(a, b);
```

IV. Illustration showing how RAM is affected

A. Example

```
void AddOne(int x, int &y;)
{
        x++;
        y++;
}

void main()
{
        int a = 2, b = 2;
        AddOne(a, b);

        cout << a << b;   // 2 and 3
}
```

B. RAM before the call to `AddOne`:

2	2
main: a	main: b

C. RAM during the call to `AddOne`:

2	2	2
main: a	main: b AddOne: y	AddOne: x

D. RAM after adding one to `x` and `y` in `AddOne`:

2	**3**	**3**
main: a	main: b AddOne: y	AddOne: x

E. RAM after the call to `AddOne`:

2	3
main: a	main: b

F. Note how `a`'s value did not change because it was passed by value, but `b`'s value changed because it was passed by reference

V. Example of using pass-by-reference in a function in the Pepsi Challenge

```
// This function adds one to coke_votes if the incoming vote is C or c,
// or it adds one to pepsi_votes if the vote is P or p
```

```
void CountVote(char vote, int &coke;_votes, int &pepsi;_votes)
{
        vote = toupper(vote);
        if (vote == 'C')
                coke_votes++;
        else if (vote == 'P')
                pepsi_votes++;
}

void main()
{
        int coke_votes = 0, pepsi_votes = 0;
        char vote;

        cout << "Enter a vote for Coke or Pepsi (C or P): ";
        cin >> vote;

        CountVote(vote, coke_votes, pepsi_votes);
}
```