• By default, most types in C++ will be treated as if they were direct values.
  – The following code will handle both variables “by value.”

```cpp
int i = 4;
Person p("Harrison Ford", 72);
```

• Note the form of the constructor call here – it’s for a “by value” class instance.
By default, most types in C++ will be treated as if they were direct values.

- Noteworthy exception: arrays are automatically pointers to the actual storage.
- The following code is valid:

```cpp
int i[] = {1, 2, 3};
int* iArr = i; //Arrays ARE ptrs.
```
Working with Data in C++

• Conversely, any type in C++ can be referred to via a pointer.
  – The following code will handle both variables “by reference.”

```cpp
int *i = new int(4);
Person *p =
  new Person(“Harrison Ford”, 72);
```

• The ‘*’ symbol denotes that the variable stores a pointer to that type.
Working with Data in C++

• If a pointer is not presently referring to any active object, it should *always* be set to “null” – the address zero (0).

```cpp
int *i = 0;
Person *p = 0;
Person *q; // WARNING: q is not
            // pointing to null!
```
Working with Data in C++

• A pointer can be obtained for any value – including pointers!
  – This is done with the & operator.

Person p("Harrison Ford", 72);
Person *pPtr = &p;
Person **pPtrPtr = &pPtr;
// Yes, pointers to pointers
// are completely legal.
Function Calls

• In C++, each function may specify the manner by which its parameters are received.
  – The type declaration of the parameter determines whether the data is passed “by value” or “by reference.”
    • Value types are said to be passed “call by value”.
    • On the other hand, reference types are said to be passed “call by reference.”
void swap(int a, int b)
{
    int temp = a;
    a = b;
    b = temp;
}

void main()
{
    int a = 2;
    int b = 3;
    swap(a, b);
}
Call By Value

```c
void swap(int a, int b) {
    int temp = a;
    a = b;
    b = temp;
}

void main() {
    int a = 2;
    int b = 3;
    swap(a, b);
}
```

Call By Value

```c
void swap(int a, int b) {
    int temp = a;
    a = b;
    b = temp;
}

do {
    int a = 2;
    int b = 3;
    swap(a, b);
}
```

```
a b
2 3
```

```
a b
2 3
```
void swap(int a, int b) {
    int temp = a;
    a = b;
    b = temp;
}

void main() {
    int a = 2;
    int b = 3;
    swap(a, b);
}
void swap(int a, int b)
{
    int temp = a;
    a = b;
    b = temp;
}

void main()
{
    int a = 2;
    int b = 3;
    swap(a, b);
}
void swap(int a, int b)
{
    int temp = a;
    a = b;
    b = temp;
}

void main()
{
    int a = 2;
    int b = 3;
    swap(a, b);
}
Call By Value

```c
void swap(int a, int b)
{
    int temp = a;
    a = b;
    b = temp;
}

void main()
{
    int a = 2;
    int b = 3;
    swap(a, b);
}
```

```
<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td></td>
<td>3</td>
</tr>
</tbody>
</table>
```
void swap(int* ints, int i_1, int i_2)
{
    int temp =
    ints[i_1];
    ints[i_1] =
    ints[i_2];
    ints[i_2] = temp;
}

public void main()
{
    int[] ints =
    {1, 2, 3, 4};

    swap(ints, 0, 3);
}

ints

1 2 3 4
void swap(int* ints, int i_1, int i_2) {
    int temp = ints[i_1];
    ints[i_1] = ints[i_2];
    ints[i_2] = temp;
}

public void main() {
    int[] ints = {1, 2, 3, 4};
    swap(ints, 0, 3);
}
Call By Reference

```java
void swap(int* ints, int i_1, int i_2)
{
    int temp =
        ints[i_1];
    ints[i_1] =
        ints[i_2];
    ints[i_2] = temp;
}

public void main()
{
    int[] ints =
        {1, 2, 3, 4};
    swap(ints, 0, 3);
}
```

```java
void swap(int* ints, int i_1, int i_2)
{
    int temp =
    ints[i_1];
    ints[i_1] =
    ints[i_2];
    ints[i_2] = temp;
}

public void main()
{
    int[] ints =
    {1, 2, 3, 4};
    swap(ints, 0, 3);
}
```
Call By Reference

```java
public void main()
{
    int[] ints = {1, 2, 3, 4};
    swap(ints, 0, 3);
}

void swap(int* ints, int i_1, int i_2)
{
    int temp =
        ints[i_1];
    ints[i_1] =
        ints[i_2];
    ints[i_2] = temp;
}
```
Call By Reference

```java
void swap(int* ints, int i_1, int i_2)
{
    int temp = ints[i_1];
    ints[i_1] = ints[i_2];
    ints[i_2] = temp;
}

public void main()
{
    int[] ints = {1, 2, 3, 4};
    swap(ints, 0, 3);
}
```

Diagram:
```
<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>2</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
Function Calls

• Additionally, using the & operator (instead of a *) will make that parameter call-by-reference.
  – It will hide the obtained address, but still work with and alter the same object/variable.
void swap(int &a, int &b)
{
    int temp = a;
    a = b;
    b = temp;
}

void main()
{
    int a = 2;
    int b = 3;
    swap(a, b);
}
Call By Reference (2)

```c
void swap(int &a, int &b)
{
    int temp = a;
    a = b;
    b = temp;
}

void main()
{
    int a = 2;
    int b = 3;
    swap(a, b);
}
```

```plaintext
<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>
```
void swap(int &a, int &b)
{
    int temp = a;
    a = b;
    b = temp;
}

void main()
{
    int a = 2;
    int b = 3;
    swap(a, b);
}
void swap(int &a, int &b) {
    int temp = a;
    a = b;
    b = temp;
}

void main() {
    int a = 2;
    int b = 3;
    swap(a, b);
}
void swap(int &a, int &b) {
    int temp = a;
    a = b;
    b = temp;
}

void main() {
    int a = 2;
    int b = 3;
    swap(a, b);
}
Call By Reference (2)

```c
void swap(int &a, int &b)
{
    int temp = a;
    a = b;
    b = temp;
}

void main()
{
    int a = 2;
    int b = 3;
    swap(a, b);
}
```

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>temp</th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>
Call By Reference (2)

```c
void swap(int &a, int &b) {
    int temp = a;
    a = b;
    b = temp;
}

void main() {
    int a = 2;
    int b = 3;
    swap(a, b);
}
```