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”, 75);
  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;
}

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

```
2
3
```

```
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);
}
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);
}
```

Before:
- `a`: 3
- `b`: 3

After:
- `a`: 2
- `b`: 3
Call By Value

```cpp
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>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>
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;
}
```
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);
}

int[] 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;
}
```
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);
}
```

Call By Reference

```c
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;
}
```

```java
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;
}
```

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

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

Before calling `swap(a, b)`, the values of `a` and `b` are 2 and 3, respectively. After calling `swap(a, b)`, the values of `a` and `b` become 3 and 2, respectively.
void swap(int &a, int &b)
{
    int temp = a;
    a = b;
    b = temp;
}

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

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

• To some of you, we imagine that some of C++’s syntax and structure may be pretty foreign, to say the least.
  – In particular, some people have never worked (heavily) with OO before.
  – This is because there’s a whole different way of thinking about programming tasks in OO.
Object-Oriented Programming

- Object-orientation is quite different.
  - As we’ve seen already, part of its design is to enforce the organization of data into logical, conceptual units within the system.
  - Each object keeps its data private (ideally) and seeks to enforce constraints to keep itself in a proper form.
Object-Orientation

• Object-orientation is quite different.
  – Work gets done by objects interacting with other objects.
    • As such, the exact flow of execution in the program may not be easy to track.
  – Object orientation aims to avoid making anything truly global.
    • Java doesn’t even allow “truly” global variables.
    • C++ allows them.