

Discussion 01

Intro to C++

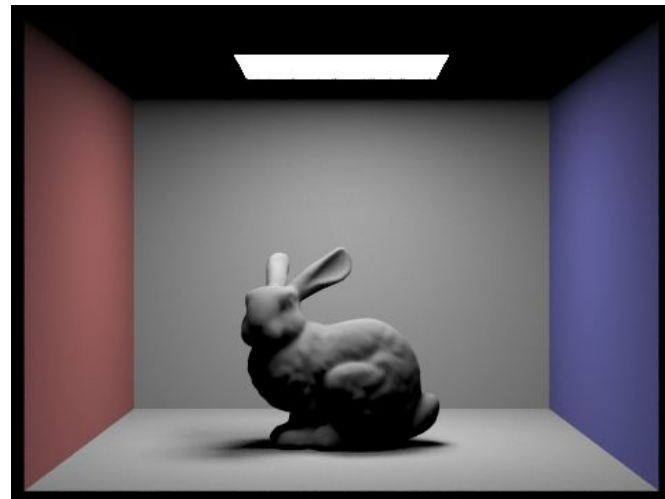
Computer Graphics and Imaging
UC Berkeley CS 184

Why C++?

- Graphics is computation-heavy → want to optimize for efficiency.
 - e.g. a 1000x1000 px image needs to run some operation 1,000,000 times.
- C++ is much faster than other languages, such as Java and Python.
- C++ offers finer control of computer resources.
- C++ is object-oriented.
- C++ is statically typed.
 - Have to build C++ code before running it.
- Used frequently in games & animation software.

C++ in CS 184

- Homeworks are primarily in C++.
- We do not require expertise on the details of C++.
- Exams test for understanding of graphics concepts, not C++ syntax.



What does this code do?

```
#include <iostream>
using namespace std;
int main()
{
    cout << "Enter a positive integer: ";
    int n;
    cin >> n;

    int k = 1;
    for (int i = 1; i <= n; i++) {
        k *= i;
    }

    cout << "The number " << n << " turns into " << k << endl;
    return 0;
}
```

What does this code do?

```
#include <iostream>
using namespace std;
int main()
{
    cout << "Enter a positive integer: ";
    int n;
    cin >> n;

    int k = 1;
    for (int i = 1; i <= n; i++) {
        k *= i;
    }

    cout << "The number " << n << " turns into " << k << endl;
    return 0;
}
```

- Syntax is very similar to Java.
- `#include <iostream>` is a library import.
- Must declare type when initializing variables.

Namespaces, Classes, and Objects

Namespaces

- I create “x”, and you create “x”. Later on, a code asks for “x”, but which one does it use?
- To solve this: C++ has namespaces.
- Namespaces provide additional scope for variables, functions, and classes.
- For example:
 - Two different classes called **Base**.
 - If they are in different namespaces (**A** vs. **B**), they won't conflict with each other.
 - The code tells them apart by specifying **A :: Base** and **B :: Base**.

Namespaces Example

```
#include <iostream>

namespace hello {
    int x = 5; // variable inside namespace
}

int main() {
    int x = 123; // local variable
    std::cout << x << std::endl; // prints "123"
    std::cout << hello::x << std::endl;
    // prints "5"
}
```

We can declare two x variables! One inside the namespace and one outside.

To refer to a namespace variable:

hello::variable

Namespaces Example

```
#include <iostream>

namespace n {
    int x = 5;
}
namespace n2 {
    int x = 4;
}

using namespace n;

int main() {
    std::cout << x << std::endl;
    // Prints 5
}
```

Alternatively, use the `using` keyword.

Namespaces Example

```
#include <iostream>

namespace n {
    int x = 5;
}
namespace n2 {
    int x = 4;
}

using namespace n;
using namespace n2;

int main() {
    std::cout << x << std::endl;
    // which x is it???
}
```

Alternatively, use the `using` keyword.

Careful: It could cause ambiguities if you're using multiple namespaces with the same variable names.

Similar bad practice in Python:

```
from module_1 import *
from module_2 import *
```

Namespaces Example

```
#include <iostream>

namespace n {
    int x = 5;
}
namespace n2 {
    int x = 4;
}

using namespace n;
using namespace n2;

int main() {
    std::cout << x << std::endl;
    // which x is it????
}
```

The compiler will tell you something's wrong!

```
main.cpp: In function 'int main()':
main.cpp:14:14: error: reference to 'x' is ambiguous
   14 | std::cout << x << std::endl;
      |               ^
main.cpp:7:5: note: candidates are: 'int n2::x'
   7 | int x = 4;
      |     ^
main.cpp:4:5: note:                  'int n::x'
   4 | int x = 5;
      |     ^
```

Classes & Objects

- All the usual OOP concepts exist in C++:
 - Objects, classes, abstraction, inheritance, polymorphism, etc.
 - Just with slightly new syntax!
- All methods & attributes of a C++ class are private, unless explicitly declared public.
- C++ also has structs → like classes, but all visibilities public by default.

```
struct Student {  
    std::string student_name;  
    int age;  
    bool enrolled_in_cs184;  
};
```

Classes and Objects

```
#include <iostream>

class Rectangle {
private:
    int width;
    int height; // private variables
public:
    void set_values (int,int);
    int area();
    // functions to declare
};

void Rectangle::set_values(int x, int y) {
    width = x;
    height = y;
}

int Rectangle::area() {
    return width * height;
}
```

```
int main() {
    Rectangle rect;
    rect.set_values (3,4);
    std::cout << "Area: " << rect.area();
    return 0;
}
```

// What is the output?

Classes and Objects

```
#include <iostream>

class Rectangle {
private:
    int width;
    int height; // private variables
public:
    void set_values (int,int);
    int area();
    // functions to declare
};

void Rectangle::set_values(int x, int y) {
    width = x;
    height = y;
}

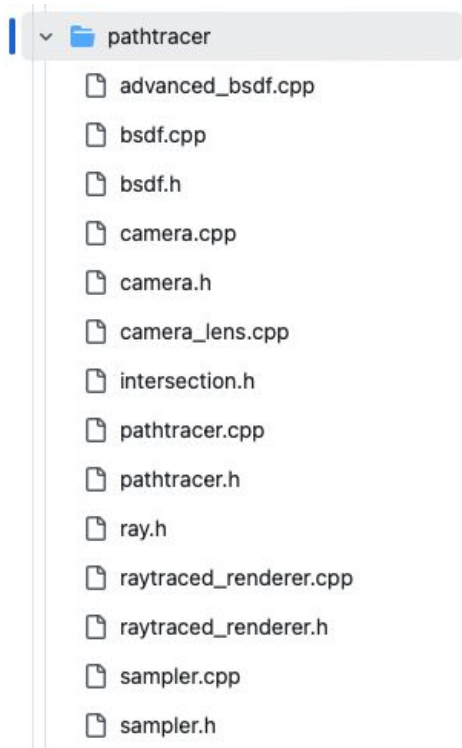
int Rectangle::area() {
    return width * height;
}
```

```
int main() {
    Rectangle rect;
    rect.set_values (3,4);
    std::cout << "Area: " << rect.area();
    return 0;
}
```

```
// What is the output?  
12
```

Header Files

- You might see a .cpp and .h file with the same name in your assignments:
 - .cpp contains what is actually *compiled and run*.
 - Code, logic, algorithms, instructions.
 - e.g. function **bodies**.
 - .h is a “header” file meant to be *imported*.
 - Class definitions, data structures, interfaces.
 - e.g. function **declarations**.
 - They are like a “table of contents”.
 - Many .cpp files can `#import` the same .h file!



.cpp vs. .h

```
// Rectangle.h:
```

```
class Rectangle {  
    private:  
        int width;  
        int height;  
    public:  
        void set_values(int,int);  
        int area();  
};
```

```
// Rectangle.cpp:
```

```
#include "Rectangle.h"  
void Rectangle::set_values(int x, int y) {  
    width = x;  
    height = y;  
}  
int Rectangle::area() {  
    return width * height;  
}
```

```
// We write out the explicit code for the  
method in the .cpp file!
```

.cpp vs. .h

```
// Rectangle.h:
```

```
class Rectangle {  
    private:  
        int width;  
        int height;  
    public:  
        void set_values(int,int);  
        int area();  
};
```

```
// main.cpp:
```

```
#include "Rectangle.h"
```

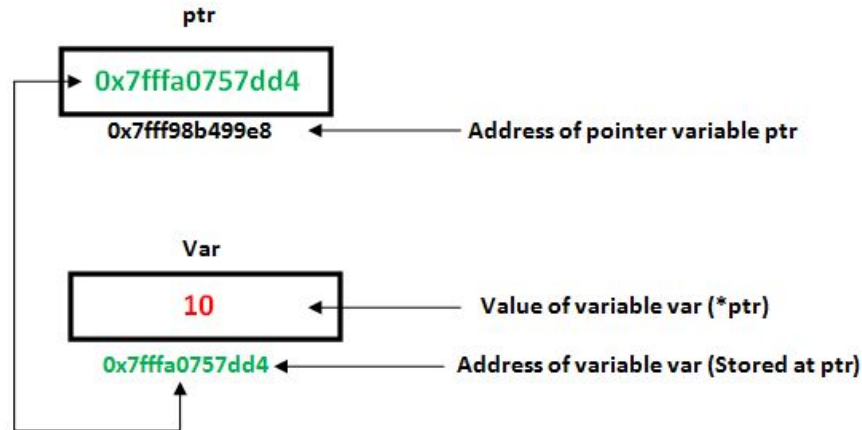
```
int main() {  
    Rectangle rect;  
    rect.set_values (3,4);  
    std::cout << "Area: " << rect.area();  
    return 0;  
}
```

Memory Allocation

Different from Java but very similar to C!

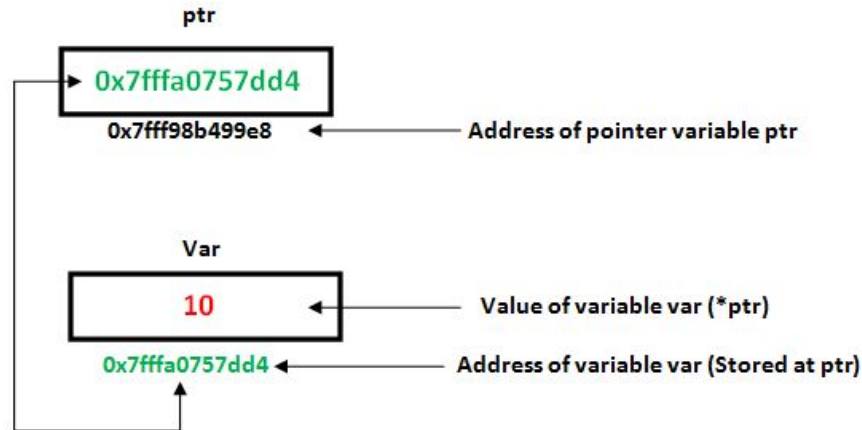
Pointers and Addresses

- A variable in C++ is assigned an address in computer memory.
- That's where the variable's value is stored!



Pointers and Addresses

- A variable in C++ is assigned an address in computer memory.
- That's where the variable's value is stored!
- A pointer is a variable whose value is the address of another variable.



Pointers and Addresses

- It must have a type.
- Initialized using an asterisk *

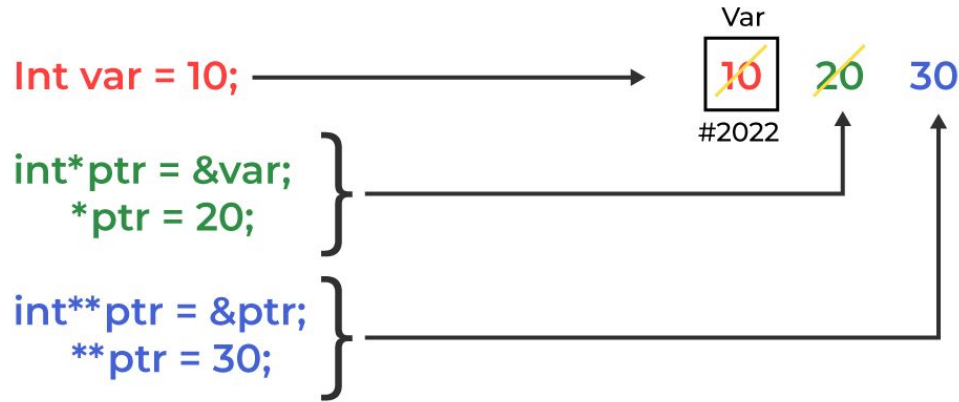
```
int *x;  
    // pointer variable is named x  
    // stores the address of an int in  
    memory
```



Reference & Dereference Operators

- Reference operator (&) → gives the address occupied by a variable.

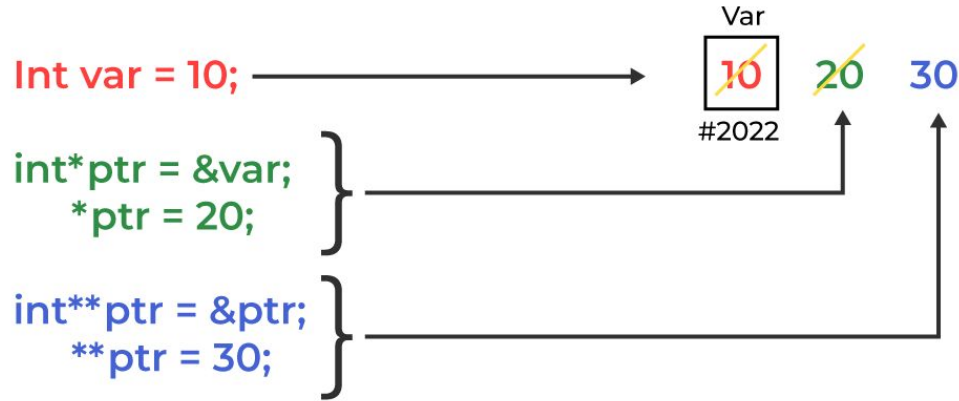
How Pointer Works in C++



Reference & Dereference Operators

- Reference operator (&) → gives the address occupied by a variable.
- Dereference operator (*) → gives value stored at a memory address.

How Pointer Works in C++



Pointers Example

```
// Example program
#include <iostream>
#include <string>

int main()
{
    int *a;
    int x = 3;
    a = &x;
    std::cout << a << std::endl;
    std::cout << *a << std::endl;
    *a = 100;
    std::cout << *a << std::endl;
    std::cout << x << std::endl;
}
```

Pointers Example

```
// Example program
#include <iostream>
#include <string>

int main()
{
    int *a;
    int x = 3;
    a = &x;
    std::cout << a << std::endl;
    std::cout << *a << std::endl;
    *a = 100;
    std::cout << *a << std::endl;
    std::cout << x << std::endl;
}
```

1. Initialize a pointer named a.
2. Initialize a variable named x and set it to 3.

Pointers Example

```
// Example program
#include <iostream>
#include <string>

int main()
{
    int *a;
    int x = 3;
    a = &x;
    std::cout << a << std::endl;
    std::cout << *a << std::endl;
    *a = 100;
    std::cout << *a << std::endl;
    std::cout << x << std::endl;
}
```

1. Initialize a pointer named a.
2. Initialize a variable named x and set it to 3.
3. Set a to the address of x.

Pointers Example

```
// Example program
#include <iostream>
#include <string>

int main()
{
    int *a;
    int x = 3;
    a = &x;
    std::cout << a << std::endl;
    std::cout << *a << std::endl;
    *a = 100;
    std::cout << *a << std::endl;
    std::cout << x << std::endl;
}
```

1. Initialize a pointer named a.
2. Initialize a variable named x and set it to 3.
3. Set a to the address of x.
4. Print a's value (this is an address!).
5. Print the value at the address that a is storing.

Pointers Example

```
// Example program
#include <iostream>
#include <string>

int main()
{
    int *a;
    int x = 3;
    a = &x;
    std::cout << a << std::endl;
    std::cout << *a << std::endl;
    *a = 100;
    std::cout << *a << std::endl;
    std::cout << x << std::endl;
}
```

1. Initialize a pointer named a.
2. Initialize a variable named x and set it to 3.
3. Set a to the address of x.
4. Print a's value (this is an address!).
5. Print the value at the address that a is storing.
6. Reassign the value at the address that a is storing to be 100.

Pointers Example

```
// Example program
#include <iostream>
#include <string>

int main()
{
    int *a;
    int x = 3;
    a = &x;
    std::cout << a << std::endl;
    std::cout << *a << std::endl;
    *a = 100;
    std::cout << *a << std::endl;
    std::cout << x << std::endl;
}
```

1. Initialize a pointer named a.
2. Initialize a variable named x and set it to 3.
3. Set a to the address of x.
4. Print a's value (this is an address!).
5. Print the value at the address that a is storing.
6. Reassign the value at the address that a is storing to be 100.
7. Print the value at the address that a is storing.
8. Print the value of x (which has now changed!).

-> Operator

```
Rectangle *x = new Rectangle(3, 4);  
int w = x->width;  
// this is the same as (*x).width
```

```
Rectangle x = Rectangle(3, 4);  
int w = x.width;  
x.area();
```

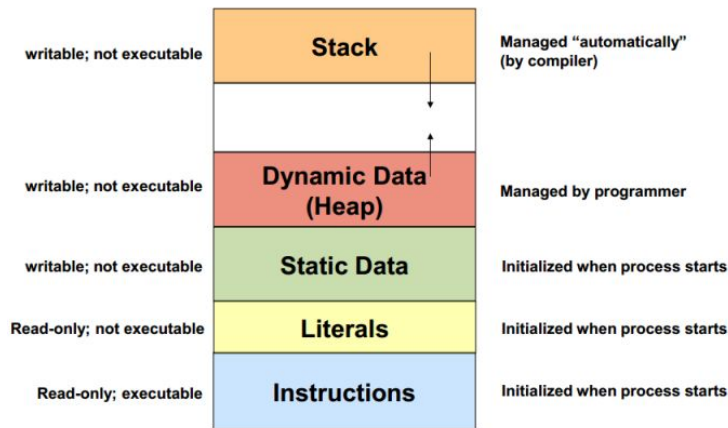
-> Operator

```
Rectangle *x = new Rectangle(3, 4);  
int w = x->width;  
// this is the same as (*x).width  
x->area();  
// also used to call methods
```

```
Rectangle x = Rectangle(3, 4);  
int w = x.width;  
x.area();
```

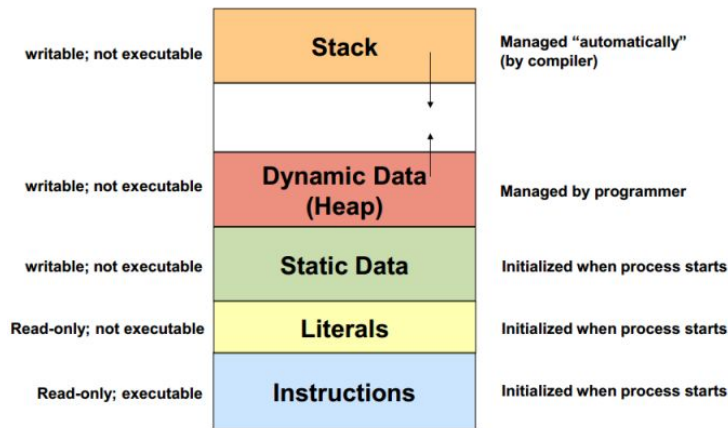
Memory Management

- Memory in your program can be allocated on the stack or the heap:



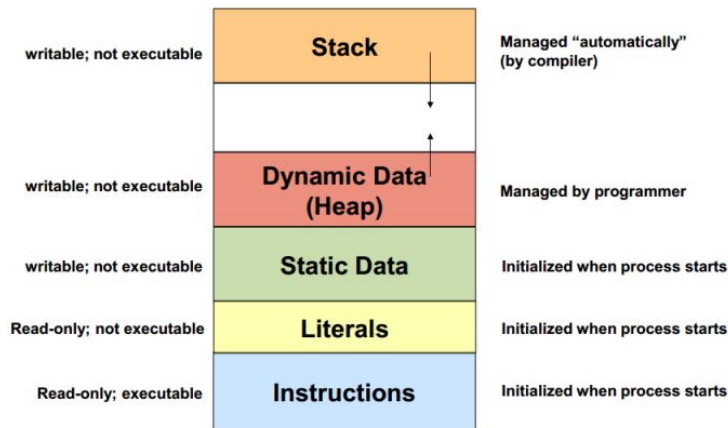
Memory Management

- Memory in your program can be allocated on the stack or the heap:
 - Stack → allocation and deallocation is automatically done.



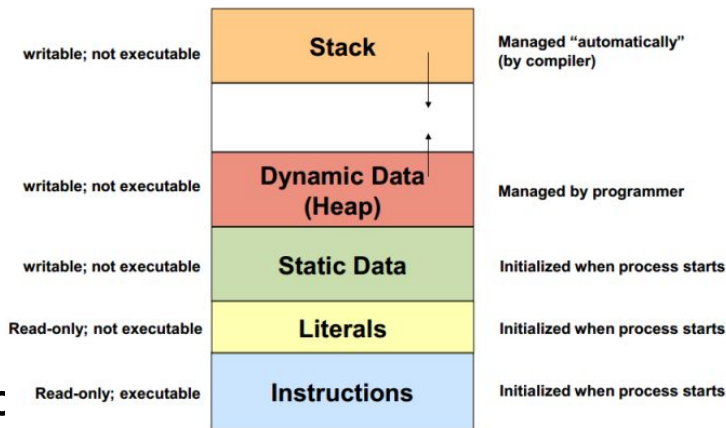
Memory Management

- Memory in your program can be allocated on the stack or the heap:
 - Stack → allocation and deallocation is automatically done.
 - Heap → needs to be done by the programmer manually.



Memory Management

- Memory in your program can be allocated on the stack or the heap:
 - Stack → allocation and deallocation is automatically done.
 - Heap → needs to be done by the programmer manually.
- The **new** operator requests memory allocation on the Heap.
- Like malloc in C. (C++ has malloc but it's less user-friendly.)



Stack

```
Rectangle rect;
```

OR

```
Rectangle rect = Rectangle();
```

- No need to delete
- To get attributes/methods, use .
(rect.width)
- If constructor has arguments:

```
Rectangle rect(3, 4);
```

OR

```
Rectangle rect = Rectangle(3, 4);
```

Heap

```
Rectangle *rp = new Rectangle();
```

- Later, must call delete rp; or it will exist until your program ends.
- To get attributes/methods, use ->
(rp->width)
- If constructor has arguments:

```
Rectangle *rp = new Rectangle(3, 4);
```

Passing Arguments

Pass By Value

```
int square_value(int a) {  
    return a * a;  
}  
  
int main() {  
    int x = 2;  
    x = square_value(x);  
}
```

Pass By Value

```
int square_value(int a) {  
    return a * a;  
}  
  
int main() {  
    int x = 2;  
    x = square_value(x); // x is now 4!  
}
```

Pass By Pointer

```
void square_pointer(int *a) {  
    *a = (*a) * (*a);  
}  
  
int main() {  
    int x = 2;  
    // Passing in an address  
    square_pointer(&x);  
}
```

Pass By Pointer

```
void square_pointer(int *a) {  
    *a = (*a) * (*a);  
}  
  
int main() {  
    int x = 2;  
    // Passing in an address  
    square_pointer(&x); // After this line, x = 4  
}
```

Pass By Reference

```
void square_reference(int &a) {  
    a = a * a;  
}  
  
int main() {  
    int x = 2;  
    square_reference(x); // After this line, x = 4  
}
```

Pass By Pointer vs. Reference vs. Value

```
void square_pointer(int *a) {  
    *a = (*a) * (*a);  
}  
  
void square_reference(int &a) {  
    a = a * a;  
}  
  
int square_value(int a) {  
    return a * a;  
}  
  
int main() {  
    int x = 2;  
    square_pointer(&x); // After this line, x = 4  
    square_reference(x); // After this line, x = 16  
    x = square_value(x); // After this line, x = 256  
}
```

Common Built-in Data Structure: vector!

Vectors

- `std::vector` is something you'll use a lot in projects!
- It's an ordered list of items, similar to:
 - Java ArrayList
 - Python List



Vectors Example

```
int main() {  
    // We initialize a vector with an initializer list  
    std::vector<int> nums = {2, 4, 6, 0, 1};  
  
    // Access by index  
    std::cout << "nums[2] is " << nums[2] << std::endl;  
  
    // Index-based iteration  
    for (int i = 0; i < nums.size(); i++) {  
        std::cout << nums[i] << " "; // prints 2 4 6 0 1  
    }  
  
    // Range for loop  
    for (int x : nums) {  
        std::cout << x << " "; // prints 2 4 6 0 1  
    }  
}
```

Range For Loops Warning


```
int main() {  
    // We initialize a vector with an initializer list  
    std::vector<Image> images = std::vector<Image>(5);  
  
    // Range for loop (items are copied here)  
    for (Image image : images) {  
        //do something  
    }  
}
```

Range For Loops Warning

```
int main() {  
    // We initialize a vector with an initializer list  
    std::vector<Image> images = std::vector<Image>(5);  
  
    // Range for loop (over references)  
    for (Image &image : images) {  
        //do something  
    }  
  
    // Alternatively:  
    for (auto it = images.begin(); it != images.end(); ++it) {  
        //do something  
    }  
}
```

Additional Resources

- [C++ Intro](#)
 - And a more detailed [guide](#) linked from that doc
- Ask ChatGPT to translate C++ code into your language of choice.
 - Just remember to review our policy on AI tools!!!
- You can also learn a lot just by poking around existing code in your assignments!

 CS 184/284A

Resources / C++ Intro

C++ Intro

All the projects for this class will primarily be in C++ (language version 11). If you have prior experience with, it adds support for object oriented code like Java such as inheritance, child classes, and overriding parent functions

For questions not answered by this guide, you can ask a TA, post on the forum, or email the instructor.

Header Files

Though C and C++ may look similar, to use the full potential of C++ you need to understand the differences. [exhaustive table of one-to-one mapping of a C header file to its C++ equivalent](#)

Thanks for coming!