CS 32 Spring 2021

Week 1 Discussion 1F

TA: Manoj Reddy

LA: Katherine Zhang

Credit: Prof. Carey Nachenberg

Survey

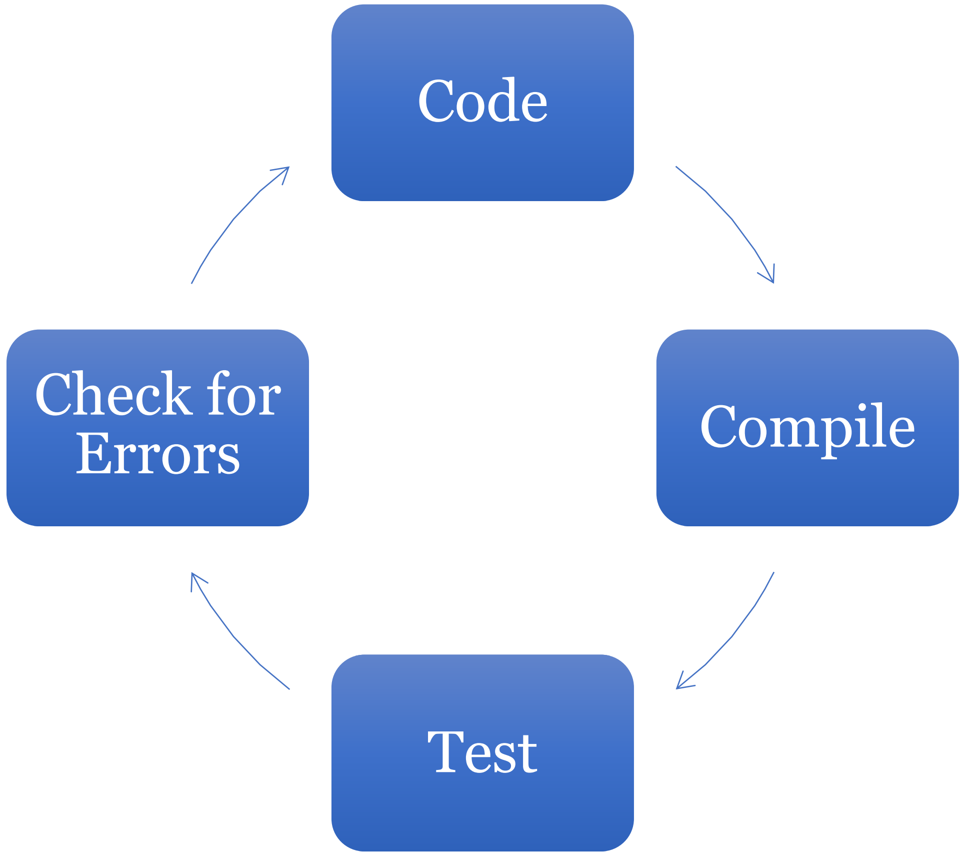
- Year in School
- Major
- Taken CS 31 last quarter (Winter 2021)
- Programming experience in High School (any language)
- Experience with C++

CS32 Outline

- CS 31 taught fundamentals of programming in C++
  - Basics, conditionals, loops etc.
- Data structures:
  - LinkedLists
  - Trees
  - HashTables
- Algorithms:
  - Recursion
  - Sorting
  - Algorithmic Efficiency
- Object-oriented programming:
  - Inheritance
  - Polymorphism
  - Templates, Iterators

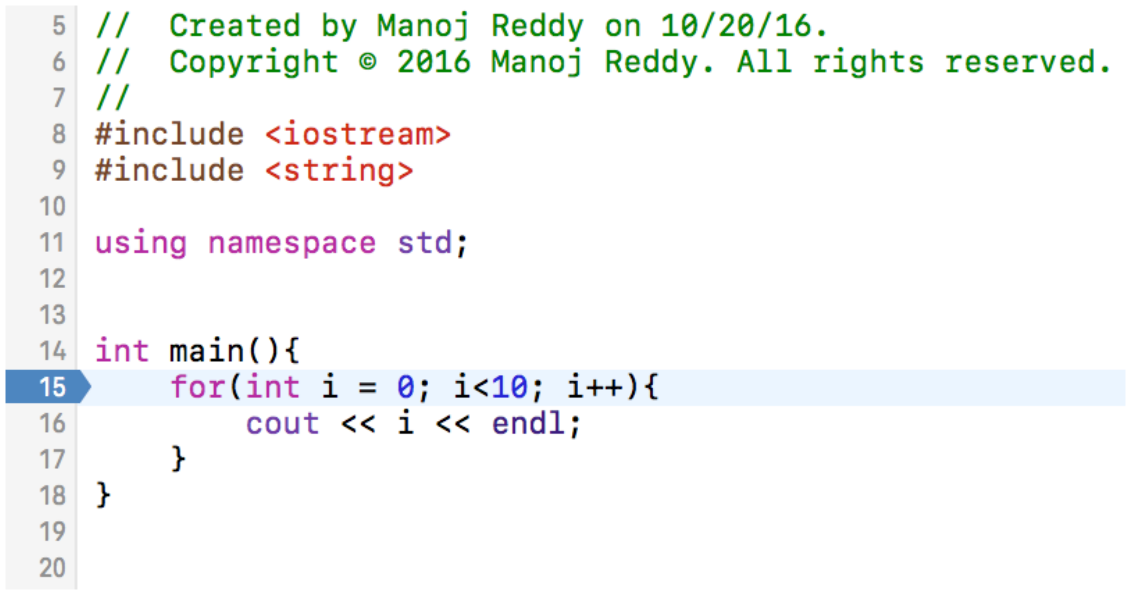
Debugging

- Workflow
- Bug = Error
- Debug = Getting rid of the error
- Time-consuming process if program is not designed properly
- Debugger is a tool that helps “debug” a program
- Xcode has a in-built debugger

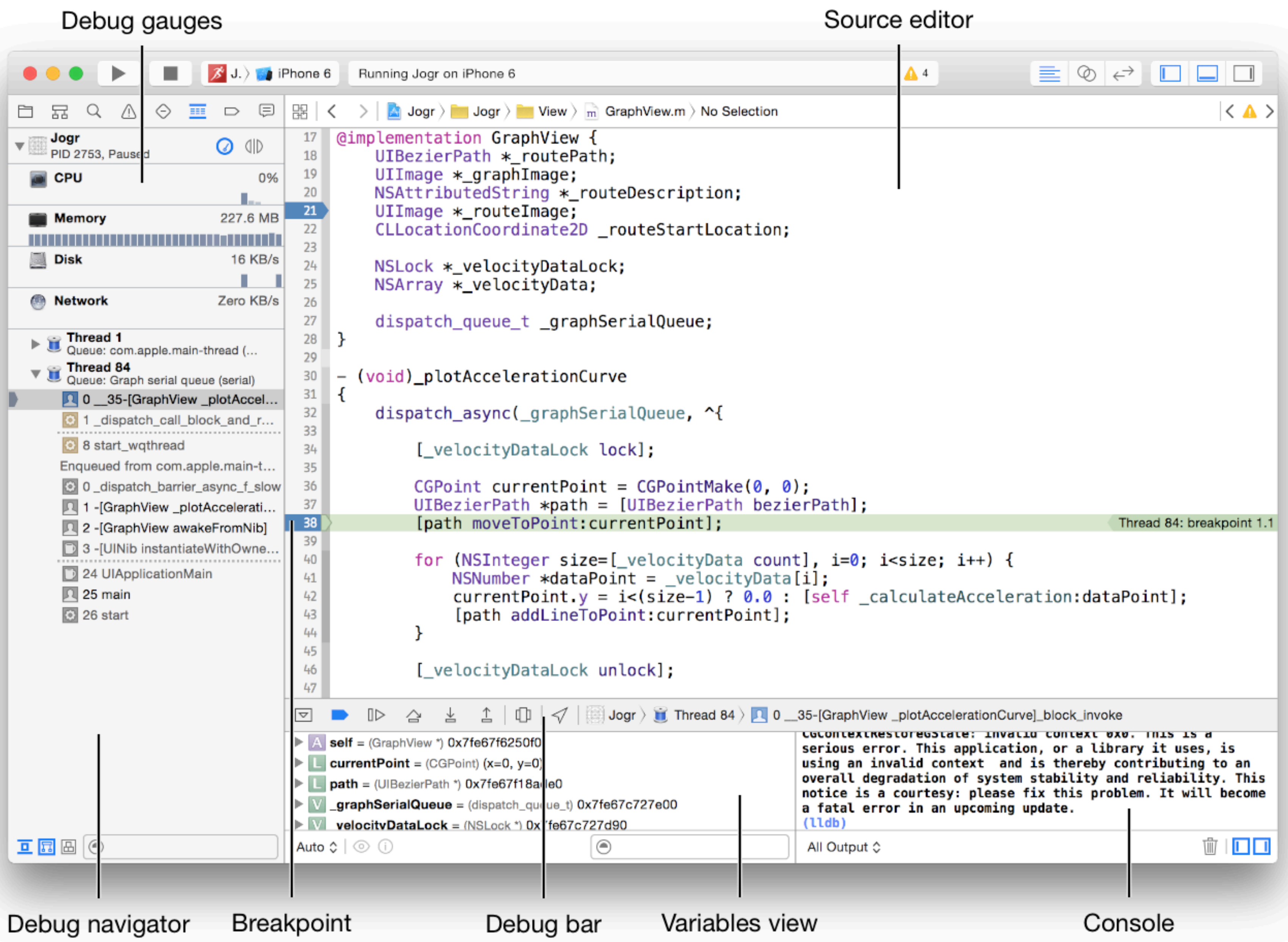


Breakpoint

- Way to intentionally stopping a program execution
- Just click on the line number
- Multiple breakpoints possible
- Breakpoints can be enabled or disabled
- Breakpoint navigator
  - Manage breakpoints



```
//  Created by Manoj Reddy on 10/20/16.
//  Copyright © 2016 Manoj Reddy. All rights reserved.
//
#include <iostream>
#include <string>

using namespace std;


int main(){
    for(int i = 0; i<10; i++){
        cout << i << endl;
    }
}


```

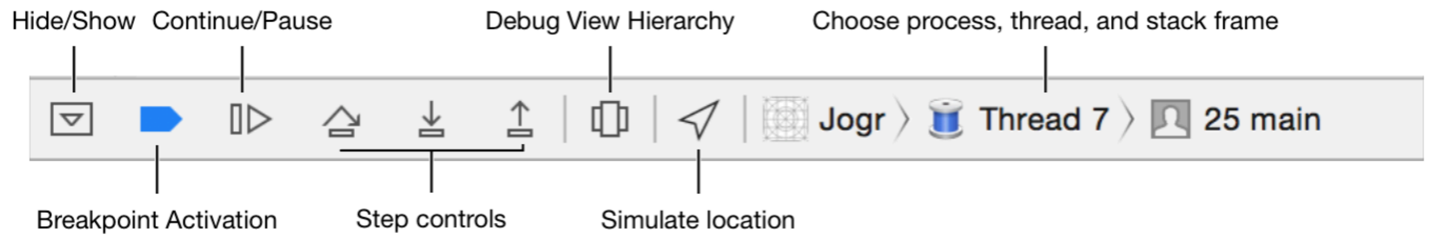

Debug gauges
Source editor
Debug navigator
Breakpoint
Debug bar
Variables view
Console
Running Jogr on iPhone 6
Jogr
PID 2753, Paused
CPU 0%
Memory 227.6 MB
Disk 16 KB/s
Network Zero KB/s
Thread 1
Queue: com.apple.main-thread (...
Thread 84
Queue: Graph serial queue (serial)
0 __35-[GraphView _plotAccel...
1 _dispatch_call_block_and_r...
8 start_wqthread
Enqueued from com.apple.main-t...
0 _dispatch_barrier_async_f_slow
1 -[GraphView _plotAccelerati...
2 -[GraphView awakeFromNib]
3 -[UINib instantiateWithOwne...
24 UIApplicationMain
25 main
26 start
Jogr › Jogr › View › GraphView.m › No Selection
Thread 84: breakpoint 1.1
Jogr › Thread 84 › 0 __35-[GraphView _plotAccelerationCurve]_block_invoke
Auto
All Output

```
@implementation GraphView {
    UIBezierPath *_routePath;
    UIImage *_graphImage;
    NSAttributedString *_routeDescription;
    UIImage *_routeImage;
    CLLocationCoordinate2D _routeStartLocation;

    NSLock *_velocityDataLock;
    NSArray *_velocityData;

    dispatch_queue_t _graphSerialQueue;
}

- (void)_plotAccelerationCurve
{
    dispatch_async(_graphSerialQueue, ^{

        [_velocityDataLock lock];

        CGPoint currentPoint = CGPointMake(0, 0);
        UIBezierPath *path = [UIBezierPath bezierPath];
        [path moveToPoint:currentPoint];

        for (NSInteger size=[_velocityData count], i=0; i<size; i++) {
            NSNumber *dataPoint = _velocityData[i];
            currentPoint.y = i<(size-1) ? 0.0 : [self _calculateAcceleration:dataPoint];
            [path addLineToPoint:currentPoint];
        }

        [_velocityDataLock unlock];
```

```
self = (GraphView *) 0x7fe67f6250f0
currentPoint = (CGPoint) (x=0, y=0)
path = (UIBezierPath *) 0x7fe67f18a1e0
_graphSerialQueue = (dispatch_queue_t) 0x7fe67c727e00
_velocityDataLock = (NSLock *) 0x7fe67c727d90
```

```
CGContextRestoreGState: invalid context 0x0. This is a
serious error. This application, or a library it uses, is
using an invalid context  and is thereby contributing to an
overall degradation of system stability and reliability. This
notice is a courtesy: please fix this problem. It will become
a fatal error in an upcoming update.
(lldb)
```

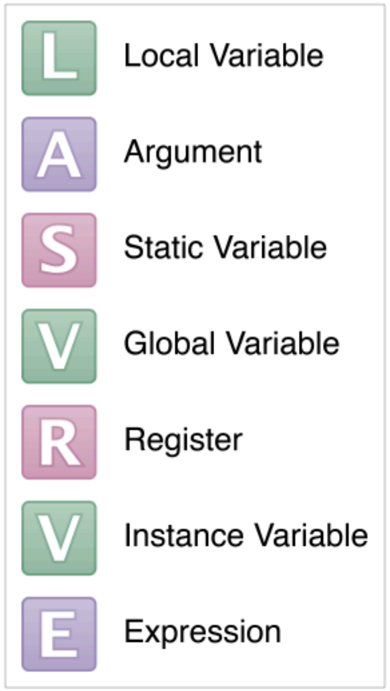
Trace



- Step over
  - Execute the current line of code and, if the current line is a function or method, step out to the next line in the current file
- Step in
  - Execute the current line of code and, if the current line is a routine, jump to the first line of that routine
- Step out
  - Complete the current routine and step to the next routine or back to the calling routine
- Continue/Pause
  - Keep running until next breakpoint statement

Variables and Stack

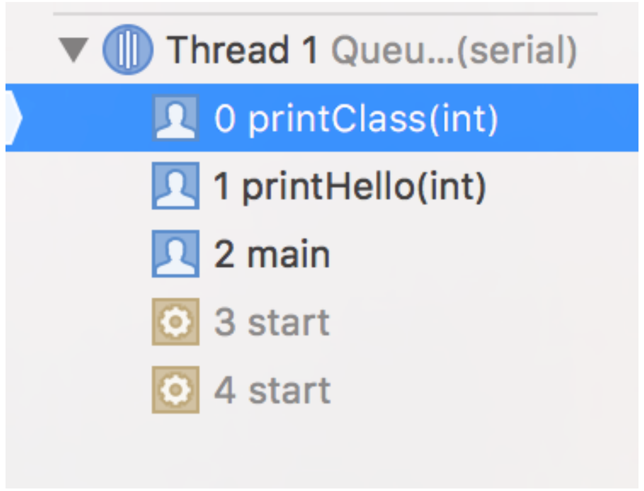
- Variable values can be viewed in the lower left box



L	Local Variable
A	Argument
S	Static Variable
V	Global Variable
R	Register
V	Instance Variable
E	Expression

- Very useful for debugging

- Function call stack can be viewed on the left hand column



```
Thread 1 Queu...(serial)
  0 printClass(int)
  1 printHello(int)
  2 main
  3 start
  4 start
```

- main -> printHello -> printClass

Pseudocode

- A more effective means of communicating an algorithm than a narrative paragraph
- Pseudo = “not real”
- Consider a function that returns the average length of the words in a string
    - How to implement it ???

Code

```
...
int totLength = 0;
int nWords = 0;
for (size_t pos = 0; ; )
{
      // find start of word
    while (pos != s.size()  &&  ! isalpha(s[pos]))
        pos++;

      // if no word, break
    if (pos == s.size())
        break;

    size_t start = pos;

      // find end of word
    do
    {
        pos++;
    } while (pos != s.size()  &&  isalpha(s[pos]));

    totLength += pos - start;
    nWords++;
}
if (nWords == 0)
    cout << "There are no words in the string" << endl;
else
    cout << "The average word length is "
         << static_cast(totLength) / nWords << endl;
...
```

Suitable pseudocode

```
...
repeatedly:
    find start of next word
    if no remaining words,
        break
    find end of word
    add word length to running total
    increment number of words
write average length, or that there were no words
...
```

Not-suitable pseudocode

```
...
set total length to 0
set number of words to 0
repeatedly:
    while current character is not alphabetic
        go on to next character
    if no remaining words,
        break
    save start position of the word
    while current character is alphabetic
        go on to next character
    add word length to total length
    increment number of words
if there were no words,
    write the no word message
else
    write the average length of the words
...
```

- Too detailed to give a clear understanding of what's going on
- Just restates almost every line of code

Not-Suitable pseudocode

First, set the total length to 0 and the number of words to 0. Then go into a loop. Inside the loop, first start a loop that checks every character until it finds a letter. If there was no letter, break out of the outer loop. Otherwise, save the start position of the word. Then start another loop that checks every character looking for a non-letter marking the end of the word. Add the length of the word to the total length and increment the number of words. When the outer loop is all done, if the number of words is 0, write the no word message, otherwise, write the average length.

- Practically useless
- Too detailed and completely hides the structure of the code

Recap: Classes & Constructors

Constructors

- A special member function that automatically initializes every new variable you create of a class
  - By default, they may have random values
  - No return value
- Example: Person class
  - name, age
- Can take variable number of arguments
- Person friends[10];
  - Default constructor is run on every element in the array
- Can define outside the class, similar to other functions after declaration
  - <ClassName>::<methodName>(<Argument List>){}

```
class Person{
public:
  …
private:
  string name;
  int age;
};
```

Destructor

- The job of the destructor is to de-initialize or destruct a class variable when it goes away.
- Syntax:
  - ~Person(){ … }
- Useful in following cases:
  - Dynamic memory allocation (new and delete)
  - Opens a disk file
  - Connects to another computer over the network

Class Composition

- Class composition is when a class contains one or more member variables that are objects
- Order of construction:
  - Member variables are constructed in order
  - Then the current class constructor is executed
- Order of destruction:
  - The current class destructor is executed first
  - The member variables are destructed in the reverse order
- Summary:
  - Constructor: Inside -> Outside (In-order)
  - Destructor: Outside -> Inside (Reverse Order)

- What is the output of the program:

```
#include <stdio.h>
#include <iostream>
using namespace std;

class A{
public:
    A(){cout << "C";}
    ~A(){cout << "2";}
};

class B{
public:
    B(){cout << "S";}
    ~B(){cout << "3";}
private:
    A a;
};


int main(){
    B b;
}
```

Initializer Lists

- Constructors can be overloaded
- Initializer lists used to initialize object member variables
- Syntax:
  - “<OuterClass>(<type> <value>…):<member_variable>(<value>…)”
- Can also be used to initialize scalar member variables
- Values need not be constants, can be variables

Copy Construction

```
class Circ
{
public:
  Circ(float x, float y, float r)
  {
    m_x = x; m_y = y; m_rad = r;
  }
  Circ(const Circ & oldVar)
  {
    m_x = oldVar.m_x;
    m_y = oldVar.m_y;
    m_rad = oldVar.m_rad;
  }
  float GetArea(void)
  {
    return(3.14159*m_rad*m_rad);
  }
private:
  float m_x, m_y, m_rad;
};
```

```
int main()
{
  Circ a(1,1,5);

  Circ b(a);


  cout << b.GetArea();
}
```

Copy Constructor
Constructor used to initialize a new variable from an existing variable of the same type

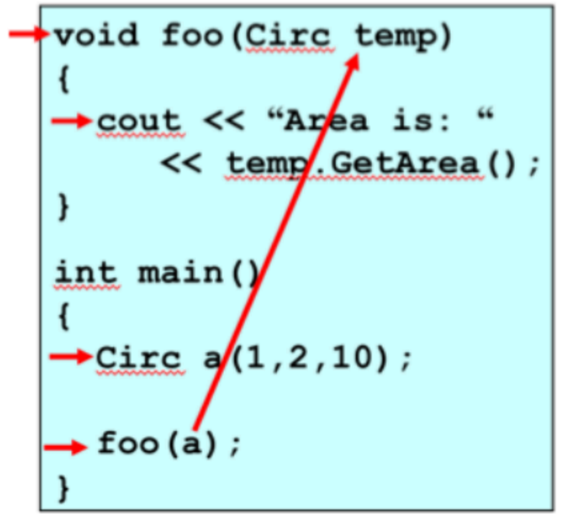
A Copy Constructor is just like a regular constructor.

However, it takes another instance of the same class as a parameter instead of regular values.

Credit: Prof. Nachenberg

Copy Constructor (2)

- Parameter to copy constructor must be const
  - Ensures the parameter variable will not be modified
- Parameter to copy constructor must be a reference (&)
  - Reason: Currently, out of scope!
- Type of parameter must be the same type as the class itself
- Equivalent
  - Circ b(a);
  - Circ b = a;
  - Defines a new variable b and then calls the copy constructor
- Also used when calling by value to a function



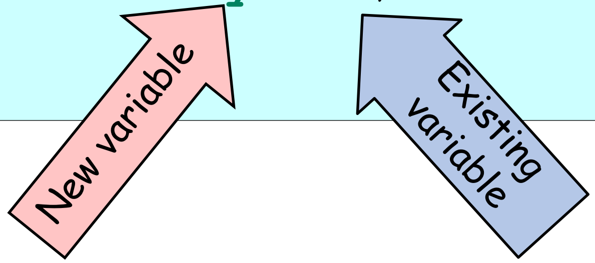
```
void foo(Circ temp)
{
 cout << "Area is: "
      << temp.GetArea();
}

int main()
{
 Circ a(1,2,10);

 foo(a);
}
```

Copy Constructor (3)

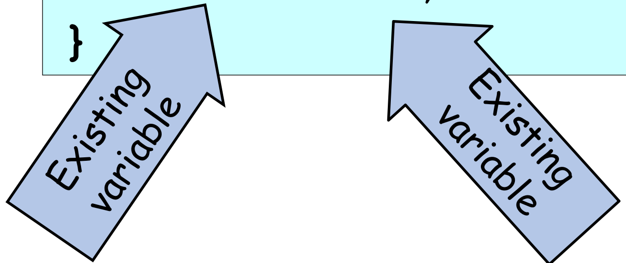
- By default, there exists a copy constructor
  - Copies all member variables from old instance to new instance
  - Known as “shallow copy”
- Important to define a copy constructor when one member variable is a pointer
- Default copy constructor does not work as expected
- Example:
  - Member variable (int *) //integer array
  - Two object variables share the same space in memory

The Assignment Operator

```
int main()
{
    Circ  x(1,2,3);

    Circ  y = x;

}
```



```
int main()
{
    Circ  foo(1,2,3);

    Circ  bar(4,5,6);

    bar = foo;
}
```



Last time we learned how to construct a new variable using the value of an existing variable (via the copy constructor).

Now lets learn how to *change* the value of an existing variable to the value of another variable.

In this example, both foo and bar have been constructed.

Both have had their member variables initialized.

Then we set bar equal to foo.

The Assignment Operator

In this case, the copy constructor is NOT used to copy values from foo to bar.

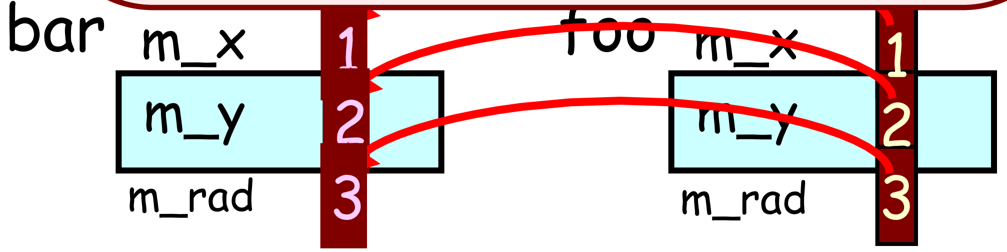
Instead, a special member function called an assignment operator is used

```
int main()
{
  Circ  foo(1,2,3);

  Circ  bar(4,5,6);

  bar = foo;
}
```

Why isn't bar's copy constructor called?

Because bar was already constructed on the line above!

The bar variable already exists and is already initialized, so it doesn't make any sense to re-construct it!



Lets see how to define our own assignment operator.

Credit: Prof. Nachenberg

The Assignment Operator

```
class Circ
{
public:
  Circ(float x, float
  {
     m_x = x; m_y = y; m_ra
  }
  Circ &operator=To(const Circ &src)
  {
    m_x = src.m_x;
    m_y = src.m_y;
    m_rad = src.m_rad;
    return(*this);
  }
  float GetArea(void)
  {
    return(3.14159*m_rad*m_rad);
  }
private:
  float m_x, m_y, m_rad;
};
```

The co
that th
not mo

You MUST pass a *reference* to the source object. This means you have to have the & here!!!

1. The function name is operator=

I'll explain this more in a bit…

tion return
reference to

3. The function returns *this when its done.

Credit: Prof. Nachenberg

So, to summarize…

If you've defined an operator= function in a class…

Then any time you use the equal sign to set an existing variable equal to another…

C++ will call the operator= function of your target variable and pass in the source variable!

Another way to read:

bar = foo;

is:

bar.operator=(foo);

i.e., we're calling bar's operator= member function!

foo
```
class Circ
```

```
cla
{
publ
   Ci
   {

   }
   Ci
   {

   }
   fl
   {

   }
private:
   float m_x, m_y, m_rad;
};
```

d an
for
do

```
C       bar(4,5,6);
```

bar.operator=(foo);

```
  Circ
{
 ...
Circ &operator=(const Circ &src)
 {
   m_x = src.m_x;
   m_y = src.m_y;
   m_rad = src.m_rad;
   return(*this);
 }
 ...
private:
  m_x 4 m_y 5 m_rad 6
}
```