

# COL100 Lecture 8

Note Title

16-08-2018

## Reviews:

for loops ; break  
nested for loops  
examples

iteration

main() // caller

foo() // callee

(0) 5 . . . . . 1  
(1) 4 . . . . . 2  
      . . . . . 3  
      . . . . . 2  
      . . . . . 1  
      . . . . . 5  
      . . . . . 4  
      . . . . . 3  
      . . . . . 2

methods :  
"void" type

type name () // declaration  
statements ;

name() ; // call

```

void printGreeting () { string name;
    cout << "Hello " << name << endl; }
}
//undefined variable name

int main () {
    string name = "";
    while (name != "") {
        printGreeting();
        name = getLine ("next name? ");
    }
    return 0;
}

```

```
void printX()
```

```
{
```

```
    cout << "X has value " << X << endl;
```

```
}
```

```
int main()
```

```
{
```

```
    int x = 2;
```


```
    printX();
```

```
    return 0;
```

```
}
```

# Scope

```
int x = 5;  
if (some-condition) {  
    int y = 10;  
}  
→ cout << x+y << endl; // undegined variable y
```



code block

When a program exits a code block,  
all variables declared inside that block  
go away!

if (some-condition)  
<sup>int</sup> <sup>int</sup> y = 10;  
cout << y << endl;

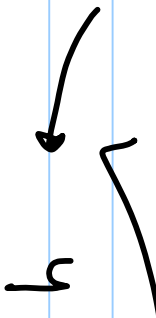
while (condition)  
→ statement;

for (...)  
statement;

int y = 10

if (somecondition) {  
... y = 4;

}

cout << y << endl; 

The scope of a variable refers to the section of code where a variable can be accessed.

- Scope starts where the variable is declared
- Scope ends at the termination of the code block in which the variable was declared
- A code block is a chunk of code between { } braces.

Lifetime of a variable  $\rightarrow$  same as scope

{

cout << n;

// UNDEF

int x = 10;

cout << x;

// ✓

}

cout << x;

// UNDEF



if {

double v = 8; // comes to lge here

if { (condition)

v = 4;

// Some other code

cost of 1

// ....

You can't have two variables of the same name in the same code block

```
{
```

```
int v = 10;
```

```
double v = 3.0; // X  
int v = 4; // X
```

```
}
```

```
if (some_conditin)
    int w = 10;
```

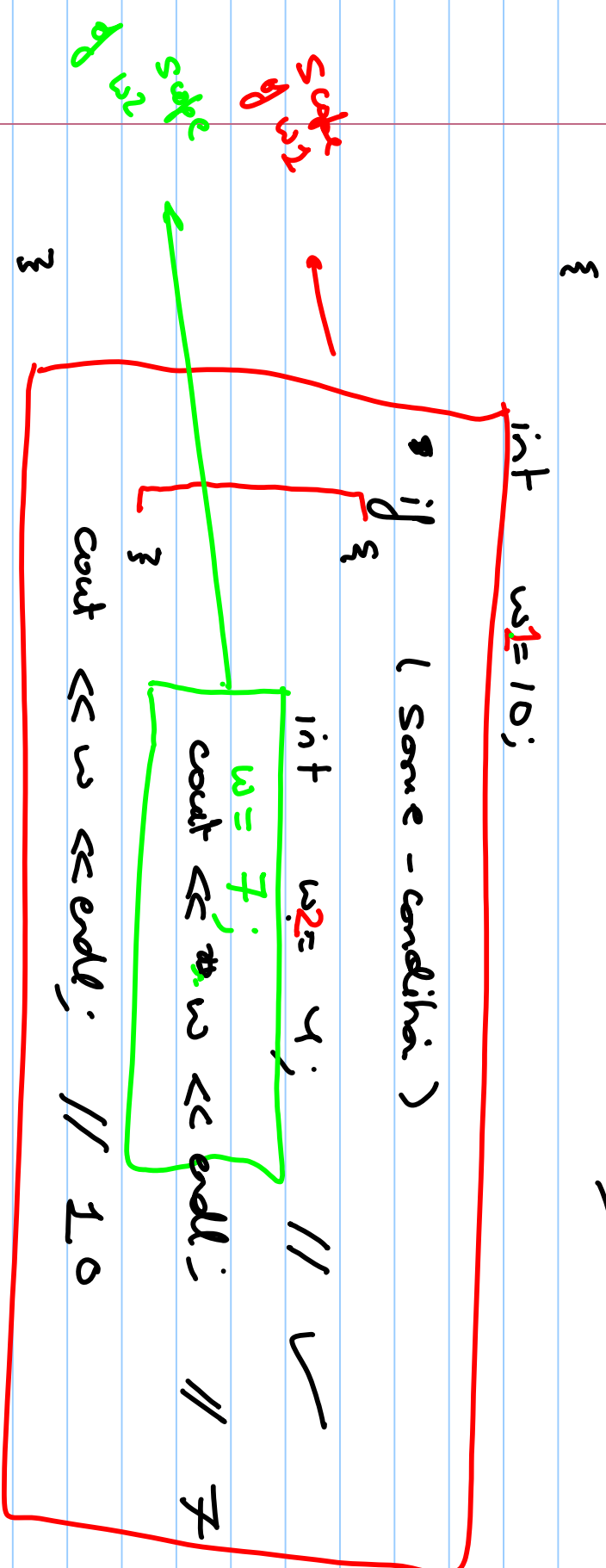
}

```
if (condition)
    int w = 4; ✓
```

}

# Nested code blocks

variable shadowing



int w;

void foo()

{

int w = 3;

for (int i = 0; i < 5; i++)

{

int w = 4;

...

}

for (int i = 0; i < 2; i++)

{

int w = 5;

...

}

}

void bar()

{

~~int w = 2;~~

foo();

cout << w;

}

undefined variable  
w

{

for ( int x = 1; ... ; )

body int x = 10; X

}

↓

{

int x = 1;

body; int x = 10; X

}

# Revising Sentinel Loops

```
int sum = 0;
int num = getInteger("next?");
while (num != -1)
{
    sum = sum + num;
    num = getInteger("next?");
}
cout << sum;
```

2 1  
3 3

```
int sum = 0;
while (true)
{
    int sum = 0;
    int num = getInteger("next?");
    if (num == -1)
    {
        break;
    }
    sum = sum + num;
    cout << sum;
}
cout << sum;
```

num X

Parameters

CALLER

parameters

```
void printGreeting(string myname)  
{
```

```
    cout << "Hello " << myname << endl >
```

```
    }  
    if main() → CALLER  
    {
```

```
        while (true) {
```

```
            if (name == "") → string name = getline("your name?");  
                break; → printGreeting(name);
```

```
        }  
        return 0;  
    }
```



your name? xyz

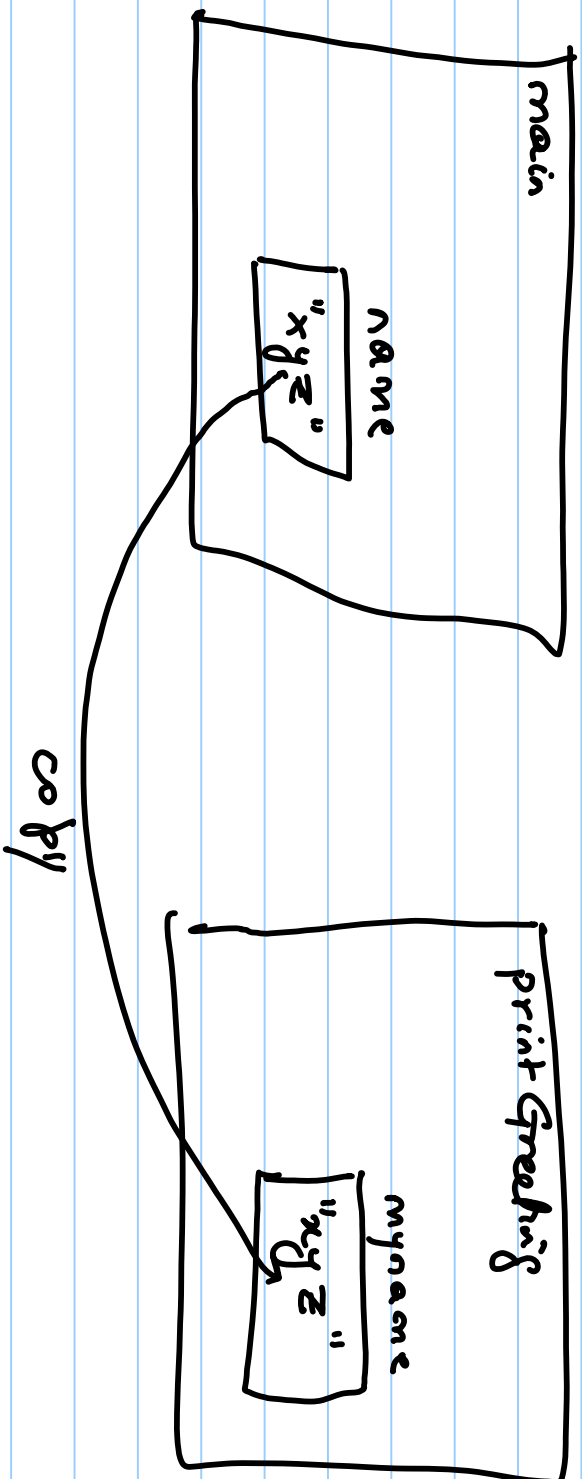
Hello xyz

your name? abc

Hello abc

your name?

<exit>



```
type name ( type name, ...,  
           {  
           statements;  
           }
```

```
name ( value 1, value 2, ..., value n ); // call
```