

EN47/COMP9

Exploring Computer Science

Lecture 6

Variables and Functions

Outline

Variables

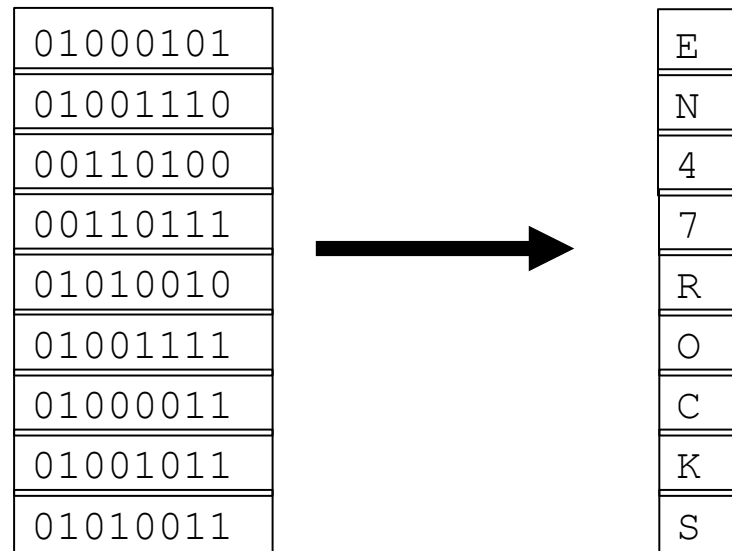
Functions

Lots of examples...

Computers store bits

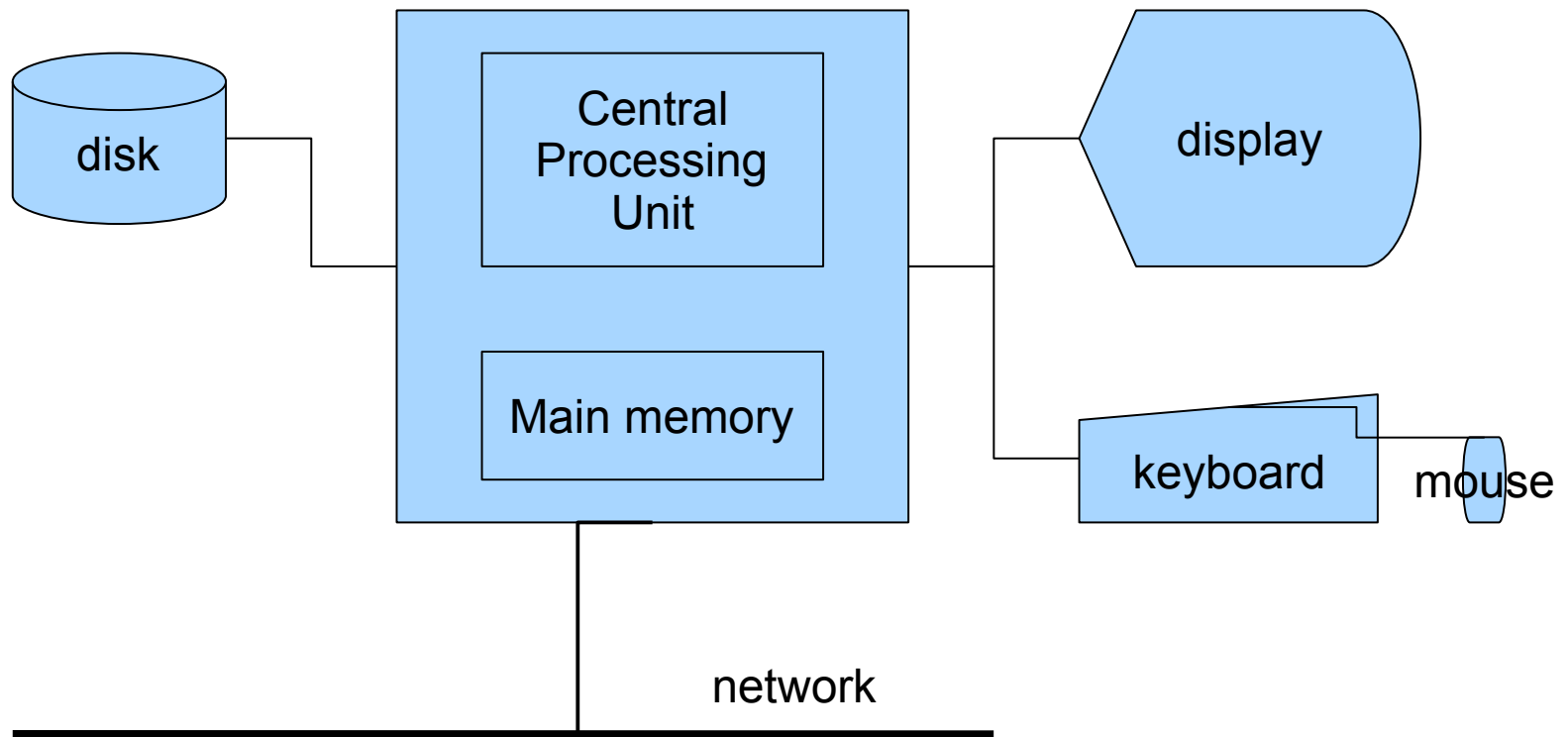
A **bit** is a binary digit: a 0 or a 1

- Any data can be represented by enough bits
- Bits are easy to represent in hardware
- Bits are an incredible pain to deal with...



The information in the bits depends on how we (and the computer) look at them!

Review: Computer Organization



Memory

Memory is a collection of locations

Each location is a group of bits

To make use of these we need:

- A way of interpreting a location: **type**
- A way to reference a location of interest: **name**

Type example

1 **byte** = 8 bits

For example: 01001111

- As a character: 'O'
 - Extra credit: See ASCII (American Standard Code for Information Interchange) code online
- As an unsigned integer: 79
 - Extra credit: Read about binary counting

Variables and types in C++

Variable declaration:

```
<type> <name>;
```

```
int count;
```

```
double gasPrice;
```

```
char bannerChar;
```

count



(int)

gasPrice



(double)

bannerChar



(char)

Initialization values

Variable declaration + initialization:

```
<type> <name> = <value>;
```

```
int count = 12;
```

```
double gasPrice = 3.79;
```

```
char bannerChar = '*';
```

count

12

(int)

gasPrice

3.79

(double)

bannerChar

*

(char)

Scope of a variable

A variable exists in the block in which it is defined

- Does not exist outside!

Local variables

- within { } (a block)
- e.g. within a loop or within a function

Global variables (bad, see below...)

Local variables in a function

Variables local to a function:

- Have meaning only in the function
- Created when function is called
- Cease to exist when the function returns

Function parameters are also local

A function with local variables

```
double DiskArea (double r) {  
    double rsquared, area;  
  
    rsquared = r*r;  
    area = 3.1416 * rsquared;  
  
    return area;  
}
```

Example 1: Functions

```
#include <iostream>
using namespace std;

void first(){
    cout << "Now in first." << endl;
}

void second(){
    cout << "Now in second." << endl;
}

int main(){
    cout << "Starting main function." << endl;
    first();
    second();
    cout << "Exiting main function." << endl;
    return 0;
}
```

Example 1: Functions

```
#include <iostream>
using namespace std;

void first(){
    cout << "Now in first." << endl;
}

void second(){
    cout << "Now in second." << endl;
}

int main(){
    cout << "Starting main function." << endl;
    first();
    second();
    cout << "Exiting main function." << endl;
    return 0;
}
```

Output:

```
Starting main function.
Now in first.
Now in second.
Exiting main function.
```

Example 2: Functions

```
#include <iostream>
using namespace std;

void second(){
    cout << "Now in second." << endl;
}

void first(){
    cout << "Now in first." << endl;
    second();
    cout << "Exiting first." << endl;
}

int main(){
    cout << "Starting main function." << endl;
    first();
    cout << "Exiting main function." << endl;
    return 0;
}
```

Example 2: Functions

```
#include <iostream>
using namespace std;

void second(){
    cout << "Now in second." << endl;
}

void first(){
    cout << "Now in first." << endl;
    second();
    cout << "Exiting first." << endl;
}

int main(){
    cout << "Starting main function." << endl;
    first();
    cout << "Exiting main function." << endl;
    return 0;
}
```

Output:

```
Starting main function.
Now in first.
Now in second.
Exiting first.
Exiting main function.
```

Example 3: Variables

```
#include <iostream>
using namespace std;

void first(){
    int area=1;
    cout << "Now in first. Area is " << area << "." << endl;
}

void second(){
    int area = 2;
    cout << "Now in second. Area is " << area << "." << endl;
}

int main(){
    int area =0;
    cout << "Starting main function. Area is " << area << "." << endl;
    first();
    second();
    cout << "Exiting main function. Area is " << area << "." << endl;
    return 0;
}
```

Example 3: Variables

```
#include <iostream>
using namespace std;

void first(){
    int area=1;
    cout << "Now in first. Area is " << area << "." << endl;
}

void second(){
    int area = 2;
    cout << "Now in second. Area is " << area << "." << endl;
}

int main(){
    int area =0;
    cout << "Starting main function. Area is " << area << "." << endl;
    first();
    second();
    cout << "Exiting main function. Area is " << area << "." << endl;
    return 0;
}
```

Output:

```
Starting main function. Area is 0.
Now in first. Area is 1.
Now in second. Area is 2.
Exiting main function. Area is 0.
```

Example 4:

Passing and returning values

```
#include <iostream>
using namespace std;

int AreaOfSquare(int x){
    int area = x * x;
    cout << "\tIn AreaOfSquare. Area is " << area << "." << endl;
    return area;
}

int main(){
    int area =0;
    int MyVariable =0;

    cout << "1. main function. Area is " << area << "." << endl << endl;

    AreaOfSquare (10);
    cout << "2. main function. Area is " << area << "." << endl << endl;

    MyVariable = AreaOfSquare (9);
    cout << "3. main function. Area is " << MyVariable << "." << endl << endl;

    area = AreaOfSquare (11);
    cout << "4. main function. Area is " << area << "." << endl << endl;
    return 0;
}
```

Example 4:

Passing and returning values

Output:

1. main function. Area is 0.

 In AreaOfSquare. area is 100.

2. main function. Area is 0.

 In AreaOfSquare. area is 81.

3. main function. Area is 81.

 In AreaOfSquare. area is 121.

4. main function. Area is 121.

Example 5: Arguments are copied

```
#include <iostream>
using namespace std;

void MyFunction(int x){
    cout << "\tA. MyFunction: x is " << x << endl;
    x = 5;
    cout << "\tB. MyFunction x is " << x << endl;
    return;
}

int main(){
    int x = 10;
    cout << "1. main function. x is " << x << "." << endl;

    MyFunction (x);
    cout << "2. main function. x is " << x << "." << endl;

    return 0;
}
```

Example 5: Arguments are copied

```
#include <iostream>
using namespace std;

void MyFunction(int x){
    cout << "\tA. MyFunction: x is " << x << endl;
    x = 5;
    cout << "\tB. MyFunction x is " << x << endl;
    return;
}

int main(){
    int x = 10;
    cout << "1. main function. x is " << x << "." << endl;

    MyFunction (x);
    cout << "2. main function. x is " << x << "." << endl;

    return 0;
}
```

Output:

```
1. main function. x is 10.
   A. MyFunction. x is 10.
   B. MyFunction. x is 5.
2. main function. x is 10.
```

Global variables

Variables declared outside of any function

- Exist in every function defined in the file

Sometimes inevitable

But in most cases, poor programming style!

- e.g. avoid taking the time to use parameters

Example: Bad use of variables

Do not use this code!

```
#include <iostream>
using namespace std;

int x = 10;

void MyFunction(int x){
    cout << "\tA. MyFunction: x is " << x << endl;
    x = 5;
    cout << "\tB. MyFunction x is " << x << endl;
    return;
}

int main(){
    cout << "1. main function. x is " << x << "." << endl;
    MyFunction (x);
    cout << "2. main function. x is " << x << "." << endl;
    return 0;
}
```

Summary

Parameters and variables declared in a function *are local to* the function:

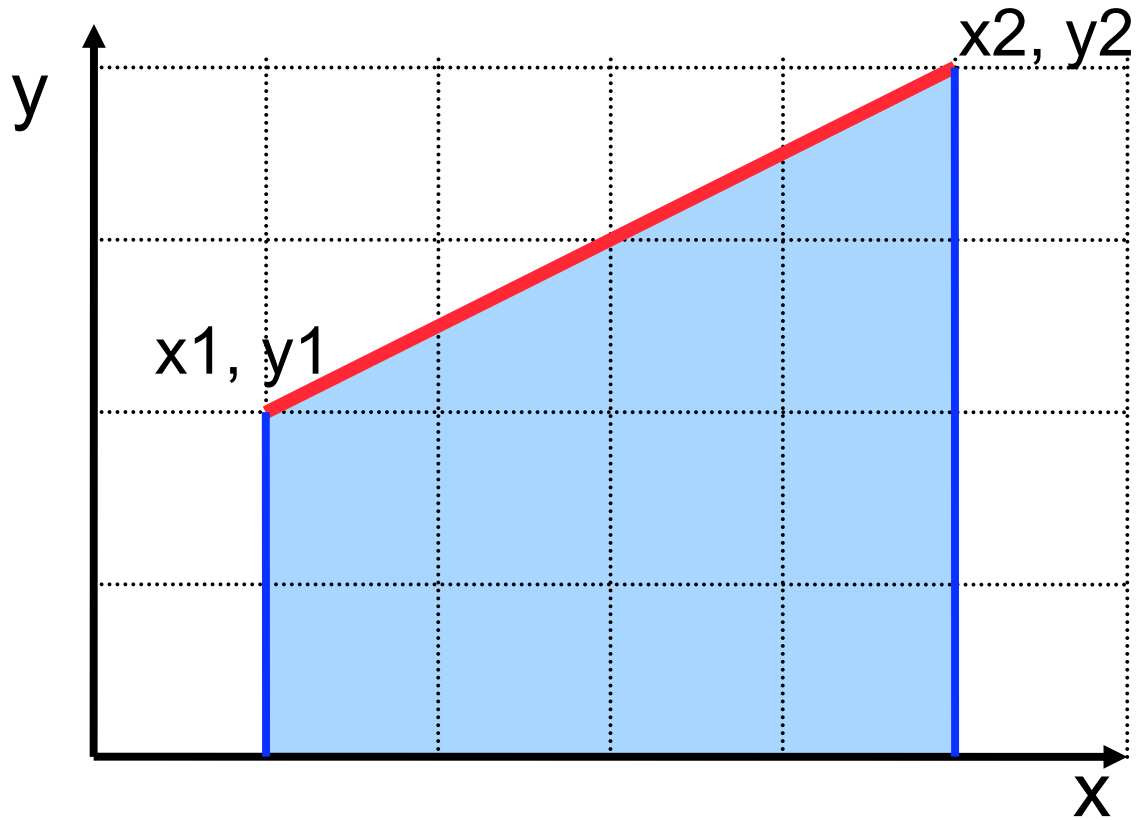
- Cannot be accessed (used) by other functions (except by being passed as arguments or return values)

Local variables are **allocated** (created) on function entry, **de-allocated** (destroyed) on function return.

Parameters initialized by copying value of argument (“Call-by-value”)

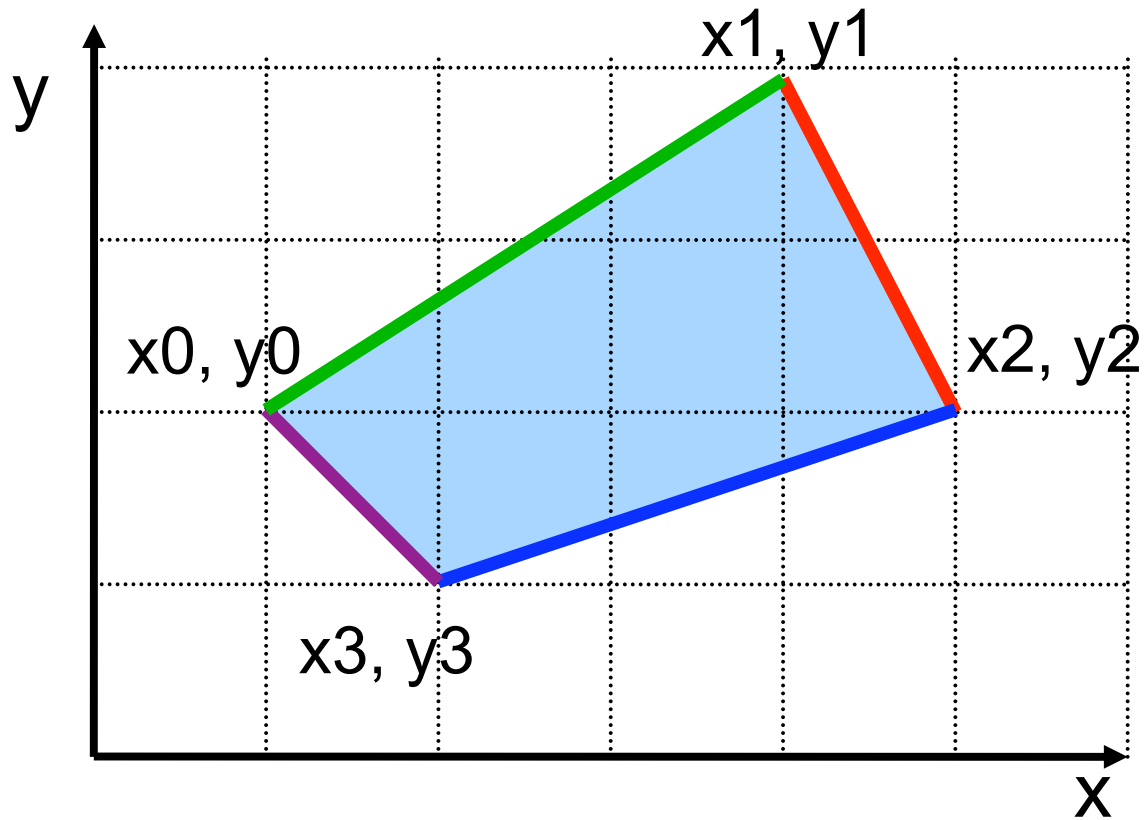
Localize information; reduce interactions.

Lab 4 Prep



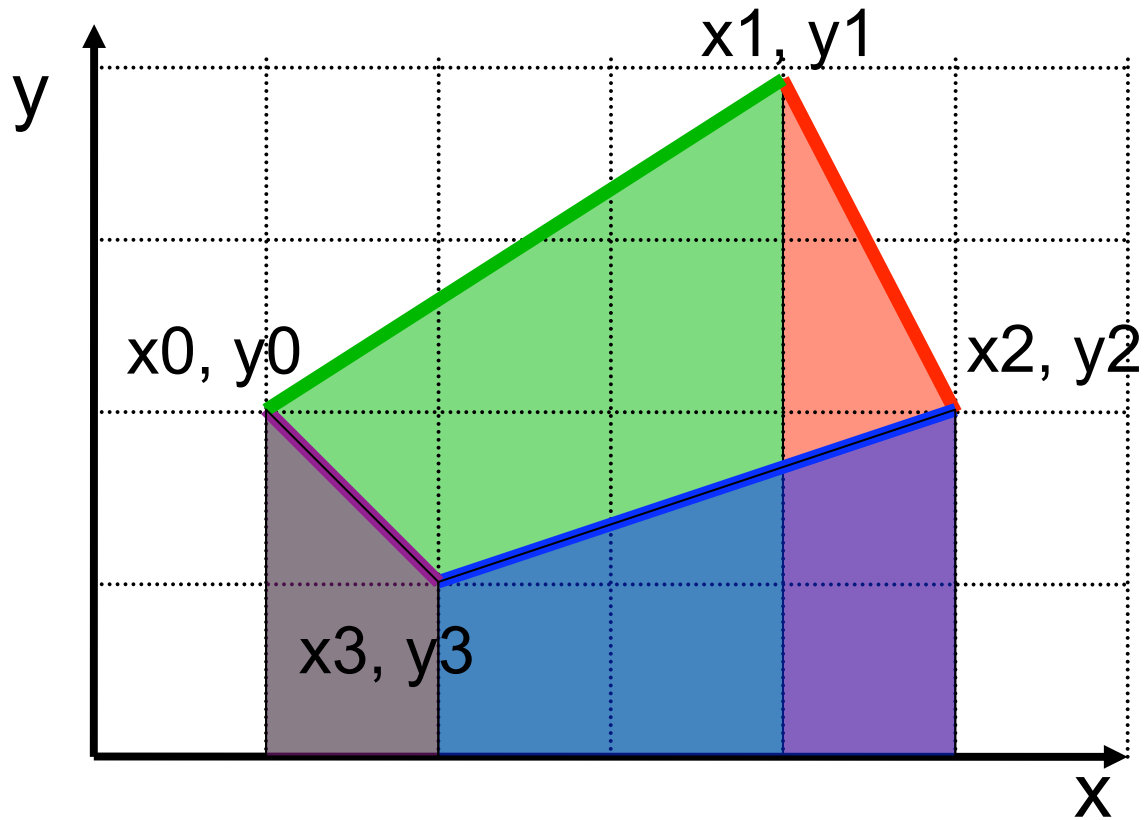
Your function asks to enter the points left to right
What happens if you give the right point first?

Geometry review



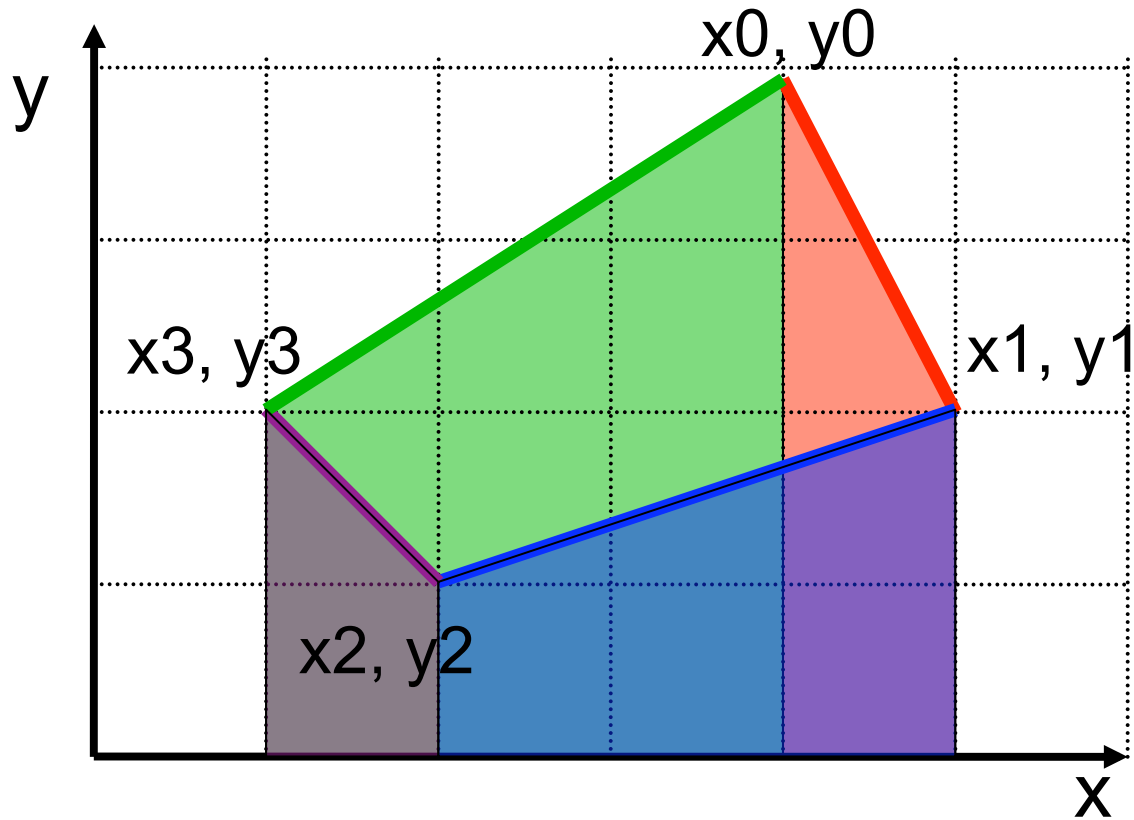
Can we use our old program to compute the area?

Geometry review



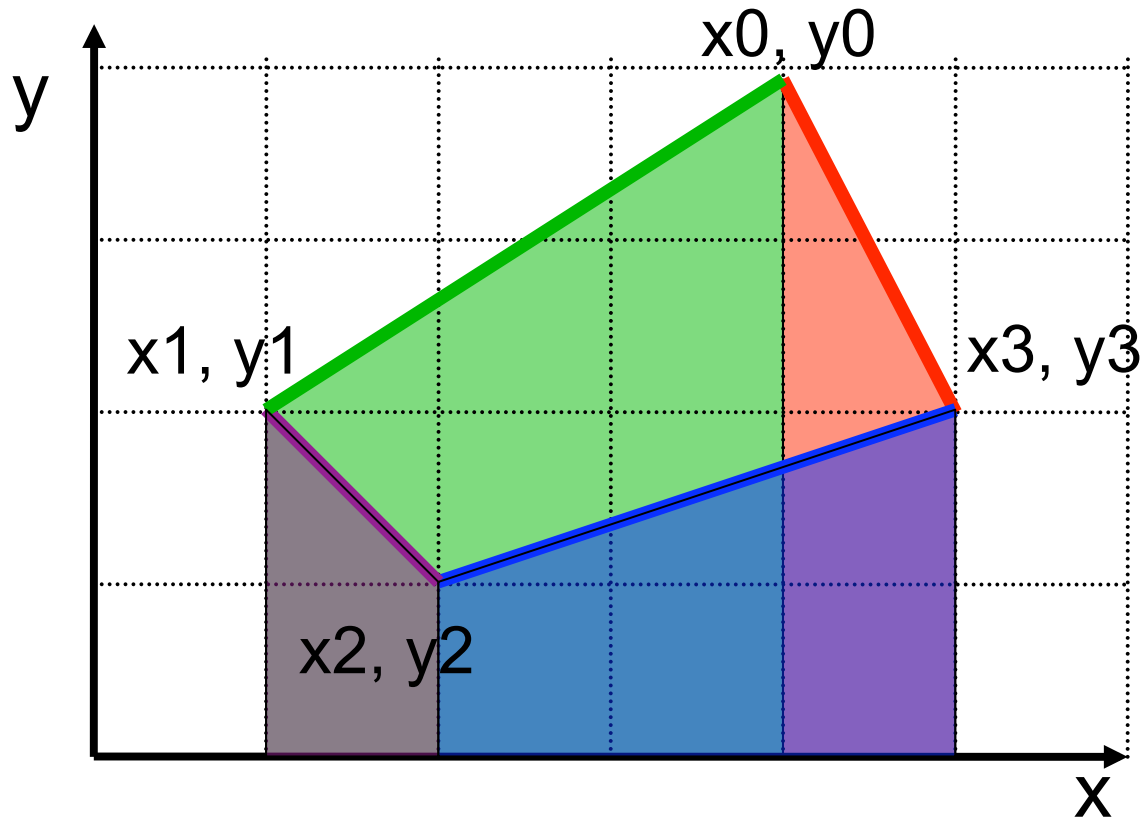
Can we use our old program to compute the area?

Geometry review



Can we re-label the points?

Geometry review



What if we specified the coordinates counter-clockwise?