MAT 685: C++ for Mathematicians

The Extended Euclidean Algorithm

John Perry

University of Southern Mississippi

Spring 2017

Outline

1. Background
2. Implementation
   - Passing by reference
   - Overloading
   - Implementation
3. Summary

Outline

1 Background

2 Implementation
- Passing by reference
- Overloading
- Implementation

3 Summary

Bézout's Identity

Theorem (Bézout's Identity)

Let $a, b \in \mathbb{Z}$. We can find $x, y \in \mathbb{Z}$ such that

$$\gcd(a,b) = ax + by \ .$$

Moreover, $\gcd(a,b)$ is the smallest positive integer of the form $ax + by$ where $x, y \in \mathbb{Z}$.

Bézout's Identity

Theorem (Bézout's Identity)

Let $a, b \in \mathbb{Z}$. We can find $x, y \in \mathbb{Z}$ such that

$$\gcd(a,b) = ax + by\ .$$

Moreover, $\gcd(a,b)$ is the smallest positive integer of the form $ax+by$ where $x, y \in \mathbb{Z}$.

Examples

- $\gcd(24,-16) = 8 = 24 \times 1 + (-16) \times 1$
- $\gcd(24,-15) = 3 = 24 \times 2 + \times (-15) \times 3$

Extended Euclidean Algorithm

Recall

If $c = a \bmod b$ then $\gcd(a,b) = \gcd(b,c)$.

Extended Euclidean Algorithm

Recall

If $c = a \bmod b$ then $\gcd(a,b) = \gcd(b,c)$.

Modify recursive Euclidean Algorithm:

given

- $a, b \in \mathbb{Z}$ — arguments
- $c = a \bmod b$ — from recursion
- $u, v \in \mathbb{Z}$ s.t. $\gcd(b,c) = bu + cv$ — from recursion

Extended Euclidean Algorithm

Recall

If $c = a \bmod b$ then $\gcd(a,b) = \gcd(b,c)$.

Modify recursive Euclidean Algorithm:

given

- $a, b \in \mathbb{Z}$ — arguments
- $c = a \bmod b$ — from recursion
- $u, v \in \mathbb{Z}$ s.t. $\gcd(b,c) = bu + cv$ — from recursion

choose q: $a = qb + c$ — compute

rewrite: $\mathbf{c = a - qb}$ — theory

Extended Euclidean Algorithm

Recall

If $c = a \bmod b$ then $\gcd(a,b) = \gcd(b,c)$.

Modify recursive Euclidean Algorithm:

given

- $a, b \in \mathbb{Z}$ — arguments
- $c = a \bmod b$ — from recursion
- $u, v \in \mathbb{Z}$ s.t. $\gcd(b,c) = bu + cv$ — from recursion

choose q: $a = qb + c$ — compute

rewrite: $\mathbf{c = a - qb}$ — theory

substitute: $\gcd(b,c) = bu + (\mathbf{a - qb})v$ — theory

rewrite: $\gcd(b,c) = av + b(u - qv)$ — theory

Extended Euclidean Algorithm

Recall

If $c = a \bmod b$ then $\gcd(a,b) = \gcd(b,c)$.

Modify recursive Euclidean Algorithm:

given

- $a, b \in \mathbb{Z}$ — arguments
- $c = a \bmod b$ — from recursion
- $u, v \in \mathbb{Z}$ s.t. $\gcd(b,c) = bu + cv$ — from recursion

choose q: $a = qb + c$ — compute

rewrite: $\mathbf{c = a - qb}$ — theory

substitute: $\gcd(b,c) = bu + (\mathbf{a - qb})v$ — theory

rewrite: $\gcd(b,c) = av + b(u - qv)$ — theory

substitute: $\gcd(a,b) = av + b(u - qv)$ — theory

Extended Euclidean Algorithm

Recall

If $c = a \bmod b$ then $\gcd(a,b) = \gcd(b,c)$.

Modify recursive Euclidean Algorithm:

given

- $a, b \in \mathbb{Z}$ — arguments
- $c = a \bmod b$ — from recursion
- $u, v \in \mathbb{Z}$ s.t. $\gcd(b,c) = bu + cv$ — from recursion

choose q: $a = qb + c$ — compute

rewrite: $\mathbf{c = a - qb}$ — theory

substitute: $\gcd(b,c) = bu + (\mathbf{a - qb})v$ — theory

rewrite: $\gcd(b,c) = av + b(u - qv)$ — theory

substitute: $\gcd(a,b) = av + b(u - qv)$ — theory

return $\gcd(a,b)$, $x = v$, $y = u - qv$ — compute

Example

Compute gcd(26, −14):

`gcd(26,-14)` `a=26, b=-14`

- $26 \geq 0$ so no change to `a`
- $-14 < 0$ so `b` reassigned to 14
- `a` and `b` not `0`
- `c = 26 % 14 = 12`
- return `gcd(14,12)`

Example

Compute gcd(26, −14):

```
gcd(26,-14)
```

```
 gcd(14,12) a=14, b=12
```

- $14 \geq 0$ so no change to `a`
- $12 \geq 0$ so no change to `b`
- `a` and `b` not `0`
- `c = 14 % 12 = 2`
- return `gcd(12,2)`

Example

Compute gcd(26, −14):

```
gcd(26,-14)
 gcd(14,12)
  gcd(12,2) a=12, b=2
```

- $12 \geq 0$ so no change to `a`
- $2 \geq 0$ so no change to `b`
- `a` and `b` not `0`
- `c = 12 % 2 = 0`
- return `gcd(2,0)`

Example

Compute $\gcd(26,-14)$:

```
gcd(26,-14)
 gcd(14,12)
  gcd(12,2)
   gcd(2,0) a=2, b=0
```

- $2 \geq 0$ so no change to `a`
- $0 \geq 0$ so no change to `b`
- **but** $b = 0$

$\therefore$ `return` $a = 2$

…cascades back up: $\gcd(26,-14) = 2$

Example

Compute $\gcd(26, -14)$:

```
gcd(26,-14)
 gcd(14,12)
gcd(12,2)=2
 gcd(2,0)=2
```

$2 = 2 \times 1 + 0 \times 1$

return 2, 1, 1

Example

Compute $\gcd(26,-14)$:

```
gcd(26,-14)
 gcd(14,12)
gcd(12,2)=2
```

received $2, 1, 1$

$2 = 2 \times 1 + 0 \times 1$

$0 = 12 - 2 \times 6$

$2 = 2 \times 1 + (12 - 2 \times 6) \times 1$

$2 = 2 \times (-5) + 12 \times 1$

return $2, 1, -5$

Example

Compute $\gcd(26, -14)$:

```
gcd(26,-14)
```

```
gcd(14,12)=2
```

received $2, 1, -5$

$2 = 2 \times (-5) + 12 \times 1$

$\mathbf{2 = 14 - 12 \times 1}$

$2 = (14 - 12 \times 1) \times (-5) + 12 \times 1$

$2 = 14 \times (-5) + 12 \times 6$

return $2, -5, 6$

Example

Compute $\gcd(26, -14)$:

```
gcd(26,-14)=2
```

received $2, -5, 6$

$$2 = 14 \times (-5) + 12 \times 6$$

$$\mathbf{12 = 26 - 14 \times 1}$$

$$2 = 14 \times (-5) + (\mathbf{26 - 14 \times 1}) \times 6$$

$$2 = 14 \times (-11) + 26 \times 6$$

$$2 = -14 \times 11 + 26 \times 6$$

return $2, 6, 11$

Outline

1 Background

2 Implementation
- Passing by reference
- Overloading
- Implementation

3 Summary

Outline

1. Background
2. Implementation
   - Passing by reference
   - Overloading
   - Implementation
3. Summary

Return 3 items?

We need `gcd()` to return $\gcd(a, b)$, x, and y

Return 3 items?

We need `gcd()` to return $\gcd(a, b)$, x, and y

Problem

C++ procedures return only **one** item

Solution

Modify arguments

Modify arguments?

To “return” more than one item, modify procedure’s arguments

Modify arguments?

To "return" more than one item, modify procedure's arguments

Problem

C++ procedures *copy* arguments by default

- Can change *copy* of argument, but…
- …doesn't change original!

Solution

Pass by "reference"

Pass by reference?

Listing 1: template to pass by reference

```
return_type proc_name(type1 & arg1, ...) {
  statement1;
  statement2;
        ...
}
```

Notice the difference?

Pass by reference?

Listing 2: template to pass by reference

```
return_type proc_name(type1 & arg1, ...) {
  statement1;
  statement2;
        ...
}
```

Notice the difference?

& pass argument by reference

- argument not copied
- procedure receives actual object
- hides some "pointer magic"
  (if you don't know what pointers are, don't worry about it)

C++ Interface

Listing 3: add to recursive `gcd.hpp`

```
/**
  Calculate the greatest common divisor
  of the first two integers, as well as
  the Bezout coefficients.
  @param a integer for gcd
  @param b integer for gcd
  @param x Bezout coefficient (modified)
  @param y Bezout coefficient (modified)
  @return gcd(a,b)
*/
long gcd(long a, long b, long & x, long & y);
```

Outline

1. Background
2. Implementation
   - Passing by reference
   - Overloading
   - Implementation
3. Summary

An objection

```
long gcd(long, long);
long gcd(long, long, long &, long &);
```

Can we do this?

Already defined `gcd()` in `gcd.hpp`; can confuse computer?

An objection

```
long gcd(long, long);
long gcd(long, long, long &, long &);
```

Can we do this?

Already defined `gcd()` in `gcd.hpp`; can confuse computer?

- **Overloading**
  - key feature of **object-oriented languages**
  - Simula (1967)
- Many languages **do not allow this**
  - identifiers must be unique
  - common in older languages
    - Fortran, C, Pascal, …

Overloading

- Same name, different procedures
- Also called **polymorphism**
- C++ allows w/different **signature**, determined by:
  - number of arguments
  - argument types

Overloading

- Same name, different procedures
- Also called **polymorphism**
- C++ allows w/different **signature**, determined by:
  - number of arguments
  - argument types

```
long gcd(long, long);
long gcd(long, long, long &, long &);
```

- first `gcd()` takes two arguments
- second `gcd()` takes four arguments

Outline

1. Background
2. Implementation
   - Passing by reference
   - Overloading
   - Implementation
3. Summary

C++ Implementation

Listing 4: add to gcd_recursive.cpp (p. 1)

```
long gcd(long a, long b, long & x, long & y) {
  // Make sure a, b nonnegative
  bool a_neg = false;
  bool b_neg = false;
  if (a < 0) {
    a = -a;
    a_neg = true;
  }
  if (b < 0) {
    b = -b;
    b_neg = true;
  }

  // if a, b both zero print error, return 0
  if ( (a == 0) and (b == 0) ) {
    cerr << "WARNING: gcd called with two zeros.\n";
    return 0;
  }
```

C++ Implementation

Listing 5: add to gcd_recursive.cpp (p. 2)

```
  if (b == 0) {
    x = 1; y = 0;
    return a;
  }
  if (a == 0) {
    x = 0; y = 1;
    return b;
  }

  long c = a % b;
  long q = a / b;

  long d = gcd(b, c, x, y);
  long new_x = y;
  long new_y = x - q*y;
  x = new_x;
  y = new_y;
  if (a_neg) x = -x;
  if (b_neg) y = -y;
  return d;
}
```

A basic test program

Listing 6: `test_xgcd.cpp` (place in same folder)

```
#include <iostream>
using std::cin; using std::cout; using std::endl;
#include "gcd.hpp"

int main() {
  long a, b, x, y;
  cout << "Enter two numbers: " << endl;
  cin >> a >> b;
  cout << "gcd(" << a << "," << b << ") = ";
  cout << gcd(a,b,x,y) << " = ";
  cout << a << '*' << x;
  cout << " + " << b << '*' << y << endl;
}
```

Building from command line, running

```
$ clang++ -c gcd_recursive.cpp
$ clang++ -o test_gcd gcd_recursive.o test_xgcd.cpp
```

Building from command line, running

```
$ clang++ -c gcd_recursive.cpp
$ clang++ -o test_gcd gcd_recursive.o test_xgcd.cpp
```

Usage:

```
$ ./test_xgcd
Enter two numbers:
26 14
gcd(26,14) = 2 = 26*-1 + 14*2
$ ./test_xgcd
Enter two numbers:
26 -14
gcd(26,-14) = 2 = 26*-1 + -14*-2
```

Careful with references, however!

If we change the lines...

```
cout << gcd(a,b,x,y) << " = ";
cout << a << '*' << x;
```

...to...

```
cout << gcd(a,b,x,y) << " = " << a
       << '*' << x;
```

...we get a slightly wrong result!

```
$ ./test_xgcd
Enter two numbers:
26 14
gcd(26,14) = 2 = 26*0 + 14*2
```

Why?

Careful with references, however!

If we change the lines...

```
cout << gcd(a,b,x,y) << " = ";
cout << a << '*' << x;
```

...to...

```
cout << gcd(a,b,x,y) << " = " << a
     << '*' << x;
```

...we get a slightly wrong result!

```
$ ./test_xgcd
Enter two numbers:
26 14
gcd(26,14) = 2 = 26*0 + 14*2
```

Why?

Answer

Compiler substitutes x before evaluating `gcd(a,b,x,y)`

Outline

1 Background

2 Implementation
- Passing by reference
- Overloading
- Implementation

3 Summary

Summary

- Math stuff
  - Extended Euclidean Algorithm
- Programming stuff
  - pass by reference
  - overloading

Homework

pp. 49-50 #3.1, 3.7, 3.9