Functions

For : COP 3330.
Object oriented Programming (Using C++)
http://www.compgeom.com/~piyush/teach/3330

Piyush Kumar

Functions in C++

- Declarations vs Definitions
- Inline Functions
- Class Member functions
- Overloaded Functions
- Pointers to functions
- Recursive functions

What you should know?

Gcd Example
1071,1029
1029, 42
42, 21
21, 0

- Defining a function

Non-reference parameter.

```
// return the greatest common divisor
int gcd(int v1, int v2)
{
    while (v2) {
        int temp = v2;
        v2 = v1 % v2;
        v1 = temp;
    }
    return v1;
}
```

Function body is a scope

Another scope. temp is a local variable

function gcd(a, b)
if b = 0 **return** a
else return gcd(b, a **mod** b)

Calling a function?

```
#include <iostream>

using std::cout;
using std::endl;
using std::cin;

int main()
{
    // get values from standard input
    cout << "Enter two values: \n";
    int i, j;
    cin >> i >> j;

    // call gcd on arguments i and j
    // and print their greatest common divisor
    cout << "gcd: " << gcd(i, j) << endl;
    return 0;
}
```

Function return types

```
// missing return type
Test(double v1, double v2){ /* … */ }

int *foo_bar(void){ /* … */ }

void process ( void ) { /* … */ }

int manip(int v1, v2) { /* … */ }       //
error

int manip(int v1, int v2) { /* … */ }   // ok
```

Parameter Type-Checking

```
gcd("hello", "world");
gcd(24312);
gcd(42,10,0);

gcd(3.14, 6.29); // ok?

// Statically typed
language
```

Pointer Parameters

```
#include <iostream>
#include <vector>
using std::vector;
using std::endl;
using std::cout;

void reset(int *ip)
{
    *ip = 0;  // changes the value of the object to which ip points
    ip = 0;   // changes only the local value of ip; the argument is unchanged
}

int main()
{
    int i = 42;
    int *p = &i;

    cout << "i: " << *p << '\n';   // prints i: 42
    reset(p);                      // changes *p but not p
    cout << "i: " << *p << endl;   // ok: prints i: 0
    return 0;
}
```

Const parameters

- Fcn(const int i) {}…
  - Fcn can read but not write to i.

Reference Parameters

```
// vec is potentially large, so copying vec might be expensive
// use a const reference to avoid the copy
void print(const vector<int> &vec)
{
    for (vector<int>::const_iterator it = vec.begin();
                                     it != vec.end(); ++it) {
        if (it != vec.begin()) cout << " ";
        cout << *it;
    }
    cout << endl;
}

int main()
{
    vector<int> vec(42);
    print(vec.begin(), vec.end()); // Defined on next slide
    print(vec);
}
```

Can be used to return information

Passing Iterators

```
// pass iterators to the first and one past the last element to print
void print(vector<int>::const_iterator beg,
           vector<int>::const_iterator end)
{
    while (beg != end) {
        cout << *beg++;
        if (beg != end) cout << " ";  // no space after last element
    }
    cout << endl;
}
```

Const references

```
// When you don't what the function to modify the value associated with
// the reference

bool isShorter(const string &s1, const string &s2){
       return s1.size() < s2.size();
}
```

Passing reference to a pointer

```
// swap values of two pointers to int
void ptrswap(int *&v1, int *&v2)
{
    int *tmp = v2;
    v2 = v1;
    v1 = tmp;
}


// v1 and v2 are the pointers passed to ptrswap themselves,
// just renamed

// The following link is out of the reach of this course:
// Reading C/C++ declarations: http://www.unixwiz.net/techtips/reading-cdecl.html
```

Array parameters

- void printvalues(const int ia[10]);
  - Parameter treated as “const int *”

No return values?

- void swap (int &v1, int &v2);

Returning from main

```
#include <cstdlib>

int main()
{
    bool some_failure = false;
    if (some_failure)
        return EXIT_FAILURE;
    else
        return EXIT_SUCCESS;
}

// Typical use: exit(EXIT_FAILURE)
```

Returning a reference

```
#include <string>
#include <iostream>

using std::string; using std::cout; using std::endl;

//inline version: find longer of two strings
inline const string &
shorterString(const string &s1, const string &s2)
{
        return s1.size() < s2.size() ? s1 : s2;
}

int main()
{
        string s1("successes"), s2("failure");
        cout << shorterString(s1, s2) << endl;

        return 0;
}
```

Tip 1: Never return a reference to a local object.

Tip 2: Never return a pointer to a local object.

Put these in header files.

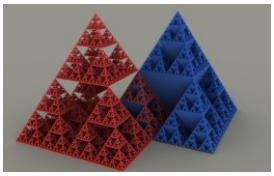
Recursion

- **Recursion** is the process a procedure goes through when one of the steps of the procedure involves rerunning the entire same procedure.
- Fibonacci number sequence:
  - $F(n) = F(n - 1) + F(n - 2)$.

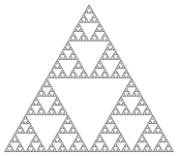
Recursion

```
function factorial(n) {
        if (n <= 1) return 1;
        else return n * factorial(n-1);
}
```

Recursion



Fractals



Recursion

```
template <typename T>
void recSort(T begin, T end){
  int len = distance(begin,end);
  if( len <= 1) return;

  for(int i = 1; i < len; ++i)
    if(*(begin+i) < *(begin))
      swap( *(begin+i) , *(begin) );
  recSort(begin+1,end);
}
```

```
int main() {
  vector<int> vs;
  int N = 10;

  for(int i = 0; i < N; ++i)
    vs.push_back(rand());

  recSort(vs.begin(), vs.end());
  for(int i = 0; i < N; ++i)
    cout << vs[i] << endl;
  return 0;
}
```

Recursion

```
template <typename T>
void recSort(T begin, T end){
  int len = distance(begin,end);
  if( len <= 1) return;

  for(int i = 1; i < len; ++i)
    if(*(begin+i) < *(begin))
      swap( *(begin+i) , *(begin) );

  recSort(begin+1,end);

}
```

Generates

```
void recSort(vector<int>::iterator begin,
             vector<int>::iterator end){
  int len = distance(begin,end);
  if( len <= 1) return;

  for(int i = 1; i < len; ++i)
    if(*(begin+i) < *(begin))
      swap( *(begin+i) , *(begin) );

  recSort(begin+1,end);
}
```

Can you see the recursion in a Sudoku solver?

Recursion

- Another example.

```
void printNum(int n)
{
  if (n >= 10) printNum(n/10);
  print(n%10);
}
```

Another version.

```
void printNumB(int n, int b)
{
  if (n >= b) printNumB(n/b);
  print(n%b);
}
```

Recursion

- Eight Queens problem:
  - Brute force solution: 64^8 = 2^48 = 281,474,976,710,656 possible blind placements of eight queens,

  - The **eight queens puzzle** is based on the problem of putting eight chess queens on an 8×8 chessboard such that none of them is able to capture any other using the standard chess queen's moves.

Recursion

- Invariants:
  - no two pieces can share the same row
  - any solution for n queens on an $n \times m$ board must contain a solution for $n-1$ queens on an $(n-1) \times m$ board
  - proceeding in this way will always keep the queens in order, and generate each solution only once.

Backtracking

- A strategy for guessing at a solution and backing up when an impasse is reached
- Recursion and Backtracking can be used together to solve our problem (and many other problems)

Eight Queens

- An observation that eliminates many arrangements from consideration
  - No queen can reside in a row or a column that contains another queen
    - Now: only 40,320 (8!) arrangements of queens to be checked for attacks along diagonals

Eight Queens

- A recursive algorithm that places a queen in a column
  - Base case
    - If there are no more columns to consider
      - You are finished
  - Recursive step
    - If you successfully place a queen in the current column
      - Consider the next column
    - If you cannot place a queen in the current column
      - You need to backtrack

The Eight Queens Problem

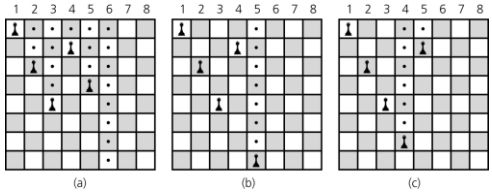
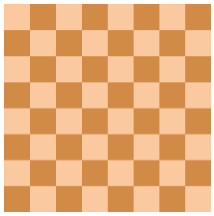


Figure 5.1 a) Five queens that cannot attack each other, but that can attack all of column 6; b) backtracking to column 5 to try another square for the queen; c) backtracking to column 4 to try another square for the queen and then considering column 5 again

Backtracking Animation



Default Arguments

```
int ff(int i = 0);
screen =  screenInit( string::size_type height = 24,
                      string::size_type width = 80,
                      char background = ' ' );

string screen = screenInit();
string screen = screenInit(66);
string screen = screenInit(66,256);
string screen = screenInit(66,256,'#');
string screen = screenInit(,,'?'); // error
string screen = screenInit('?'); // calls ('?',80,' ');
```

You can also call functions to initialize default Arguments. These functions are called when the function is called.

Function overloading

- Functions having same name but different parameter types.
  - Record lookup(const Account&);
  - Record lookup(const Phone&);
  - Record lookup(const Name&);
  - Record lookup(const SS&);