CS11600-1
Intro to Programming II

Instructor:
Catalin Dumitrescu <cldumitr@cs.uchicago.edu>

TAs:
Irina Matveeva <matveeva@cs.uchicago.edu>
Zhimin Xie <zxie@cs.uchicago.edu>

Last Time

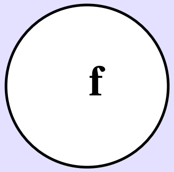
- Types (Section 2.1)
- Expressions (Section 2.5)
- Functions (Section 3.1)

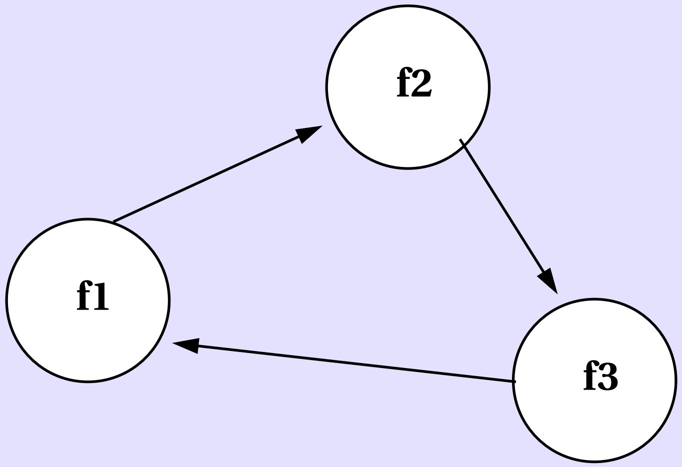
Today

- Recursivity
- More about Pointers
- Dynamic Allocation of Memory

Recursivity

- Method (expression) that
  - calls itself (immediate recursion)
  - is in a potential cycle of function calls





More about

- Decomposition into smaller problems of same type
- Necessity of a base case
- Base case must be reached

Recursivity (2)

- Fibonnacci sequence:
  - fib (1) = 0
  - fib (2) = 1
  - fib (N) = fib (N-1) + fib (N-2)

Example (e7.cc)

```
int fib_rec (int i) {
  if (i <= 1)
    return i;
  else
    return fib_rec (i - 1) + fib_rec (i - 2);
}

int fib_iter (int i) {
  int a = 1, b = 0;
  for (j = 0; j < i; j++)
    {
      c = a + b;
      b = a; a = c;
    }
  return a;
}
```

Recursivity (3)

- K out of N
  - $C(n,0) = 1$
  - $C(n,n) = 1$
  - $C(n,k) = C(n-1,k-1)+C(n-1,k)$

Recursivity (4)

- gcd (a, b) = b *if b* | *a*

- gcd (a, b) = gcd (b, a%b)

Example (e8.cc)

```
int gcd_rec (int a, int b)
{
  if (a % b == 0) return b;
  return gcd_rec (b, a % b);
}

int gcd_iter (int a, int b)
{
  while (a % b)
    {
      int c = a % b;
      a = b; b = c;
    }
  return b;
}
```

Recursivity Support

Efficiency

- Produce clean solutions
- Valuable when no simple iterative sol. is possible
- Introduce bookkeeping overhead

Inline Functions

- The benefits of defining of a function are:
  - it is generally easier to read a call with a meaningful name
  - it is easier to change one single occurence
  - semantic is uniform
  - code can be reused

- Drawback: making functions for small sets of instructions will slow the code

- Possible Solutions:
  - inline functions: *inline int min (x, y) { return (x > y) ? y : x; }*
  - macrodefinitions: *#define min(x,y) ((x) > (y) ? (y) : (x))*

extern "C"

- Used to instruct the compiler about a function written and compiled in C

Example: *extern "C" int WinMain (...) { ... }*

Scopes and Lifetime

- Every name in C++ must refer to a unique entity, but a name is not used only once

- Solution -- using contexts
  - global and local scopes
  - namespace scopes
  - class scopes

Global and Local Scopes

- Local scope: a portion of program contained within a function

```
int main ()
{
   int a = 20;
   ...
}
```

- Global Scope: a function declaration introduces ain the global scope a new symbol

```
int fib_rec (int i)
{ ... }
```

Other Scopes

- **Namespace Scopes**

Example:

Named namespaces:

```
namespace std {
    ostream cout;
};
using namespace std;   cout ;   or   std::cout;
```

Unamed namespaces (never span multiple files):

```
namespace {
    int a;
};
```

- **Class Scopes**

Pointer Types

- Holds the address of another object, allowing for indirect manipulation of that object
- Every pointer has an associated type

Example:

```
int a = 6, *ptr_a = &a, **pptr_a = &ptr_a;
cout << a << *ptr_a << endl;
**pptr_a = 7;
cout << a << *ptr_a << endl;
cout << &a << endl;
```

- **const and pointers:**

```
const int* a; // pointer to a const object
int *const a; // constant pointer
```

Homework

- **Due date: January 17, 8:00pm**

- Problem 0. Problem 7 / Page 81 (Navigating C++) -- nothing to hand in
- Problem 1. Write two functions (ten2tes and tes2ten) for translating large
  - numbers represented as arrays of characters from base 10 to base 2, 8 or 16
  - and, respectively from 2, 8 or 16 to 10.

```
char* ten2tes (char* bten, int base);
char* tes2ten (char* tes);
```

Example:

10 (10) = xA (16) = 012 (8) = b1010 (2)

20 (10) = x14 (16) = 024 (8) = b10100 (2)

Problem 2. Consider the following definition of a person:

```
typedef struct person_st {
  char *fname; char *lname; int   age; } person;
```

- Write a program that reads from input or a file information about several persons
- and saves everything in another file ordered by age.
- Problem 3. Problem 1 / Page 164 (Navigating C++)

Next Time

- More about structures and unions
- More about I/O operations
- Other non-OO elements

Reference Types

- **Introduce an alias, an alternative name for an object (?)**
- **Allow indirect manipulation without the necessity of a ptr**

Example:

```
int a = 6, &ref_a = a;
cout << a << ref_a << endl;
```

- **Note: once created a reference cannot be changed**

Arrays and Pointers

C and C++:

```
int vec[4] = { 1, 2, 3, 4 };
vec  == &vec[0]
*vec = vec[0]
vec + 1 = &vec[1]
*(vec + 1) = vec[1]
```

Note for C++:

```
#include <vector>
vector <int> ivec(10);
ivec[1] = ivec[2];
if (ivec.empty ()) { ... }
```

Example (e9.cc)

```
#include <iostream>

void ia_print (int* pbegin, int* pend)
{
   while (pbegin != pend)
      { cout << *pbegin << " ' "; ++pbegin; }
}

int main ()
{
   int ia[4] = { 0, 1, 2, 3 };
   ia_print (ia, ia+4);
}
```

Pointers to Functions

```
int (*pf) (int, int);   (!= int *pf (int a, int b); )

int max (int a, int b) { ... }
int min (int a, int b) { ... }

int main ()
{
  if (<cond>)
        pf = min;
  else
        pf = max;
  // Other operations
  pf (3, 4)
}

int (*apf[2]) (int, int) = { min, max };
```

Example (e10.cc)

```
#include <iostream>
int min (int*, int);
int (*pf) (int*, int) = min;
const int size = 5;
int ia[size] = { 7, 8, 9, 2, 5};
int main (){
    cout << min (ia, size) << pf (ia, size) << endl;
    return 0;
}
int min (int* ia, int sz) {
    int minVal = ia[0];
    for (int ix = 1; ix < sz; ++ix)
        if (minVal > ia[ix]) minVal = ia[ix];
    return minVal;
}
```

Function Support

- **Argument Passing**
- **Returning Value**

main (): Handling Line Opt

```
./a.out -d -f input.txt

int main () { ... }

int main (int argn) { ... }

int main (int argn, char *argv[]) { ... }

    argv[0] = "./a.out"
    argv[1] = "-d"
    argv[2] = "-f"
    argv[3] = "input.txt"
```

Dynamic Memory

- Lifetime of a *static* declared object is defined during execution

- Sometime is necessary to define objects with sizes and lifetime controlled by programmer (Ex: reading a string from stdin)

- new / delete (C++)

- malloc / free (C - lower level manipulation of memory)

Example (e11.cc)

Array Allocation

- **Used also for allocating memory for arrays**

Example:

```
// A single object with value 1024
int *obj = new int (1024);
// An array
int *vec = new int [1024];
int (*mat)[1024] = new int [4] [1024];

const int *pci = new const int (1024);
```

Example (e12.cc)

```
#include <iostream>

int*[] alloc_mat (int dim1, int dim2) {
    int (*mat)[dim1] = new [dim1][dim2];
    return mat;
}

int main ()
{
    int (*mat1)[10] = alloc_mat (10, 10);
    int (*mat2)[20] = alloc_mat(20,20);
    delete []mat1;
    delete []mat2;
}
```