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

- Introduction to C/C++ (non-OO -> procedural)
  - C/C++ Program Structure
  - Statements
    - Simple (type-safe) I/O operations (*Section 2.4*)
    - Control flow (*Section 2.6*)
  - Notions about
    - Source Files
    - Preprocessing (*Section 2.2*)
    - Compilation and Linking Processes

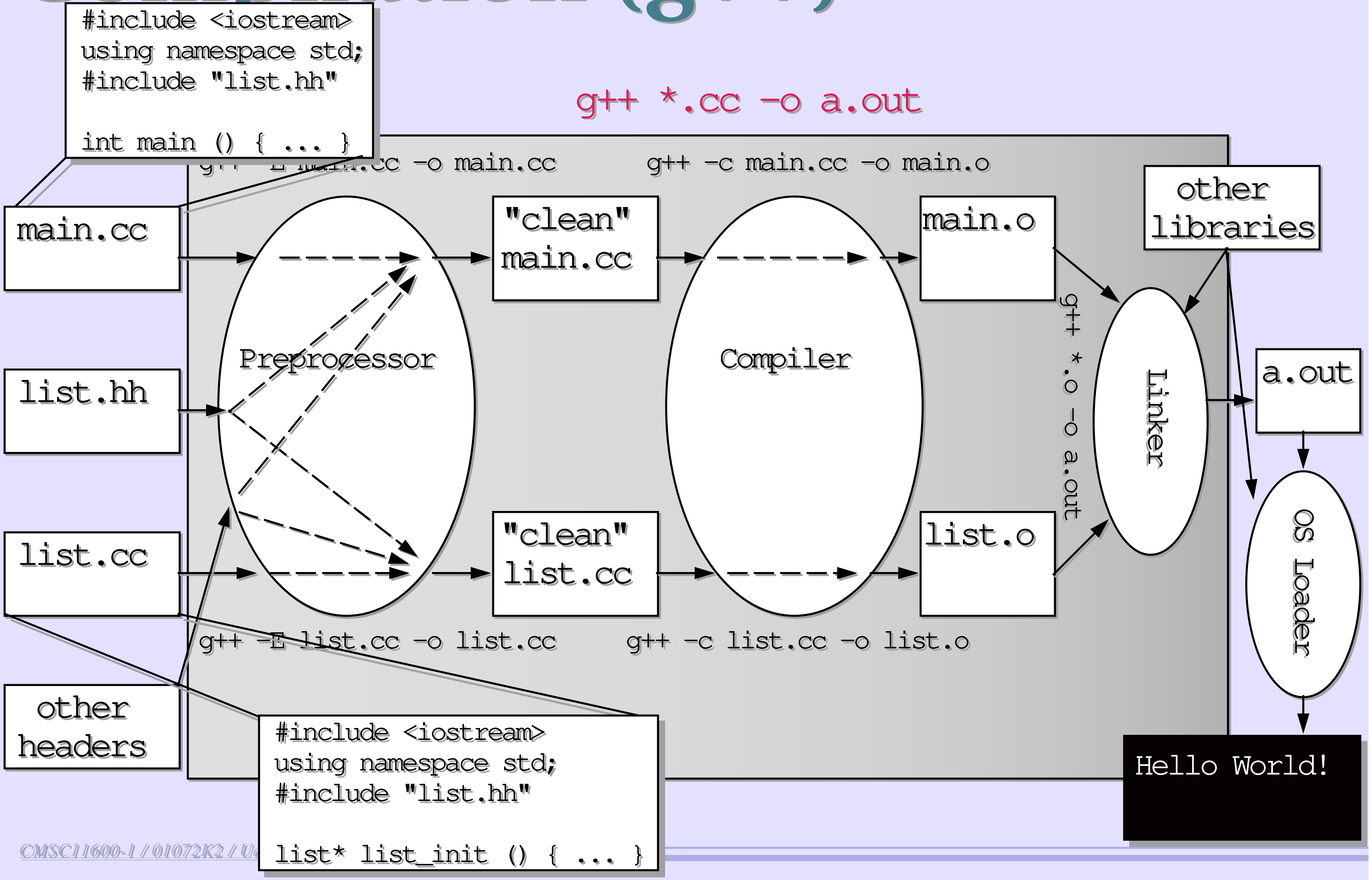
2 corrections

- Compiler parsing strategies
- declaration_list and statement_list inside compound_statements

Example (e3.cc)

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

int main () {
    int i, prevTerm = 0, crtTerm = 1, nextTerm;

#ifdef DEBUG
    int term; cout << "Input N: "; cin >> term;
#else
#  define term 10
#endif /* DEBUG */

    for (i = 3; i <= term; i++) {
          nextTerm = prevTerm + crtTerm; prevTerm = crtTerm; crtTerm = nextTerm;

#ifdef DEBUG
      cout << "For i = " << i << " => t = " << t << endl;
#endif /* DEBUG */
     }
    cout << "The " << term << " term in the sequence is: " << nextTerm << endl;
    return 0;
}
```

Compilation (g++)



Today

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

Variables

- **Used to manipulate memory regions using names**

- **Elements (C/C++ specifics):**
  - ***types, name, value***
  - ***lvalue, rvalue***

Example:

```
char  *helpMessage = "  Help Message  ";
int      prevTerm = 0, crtTerm = 1, nextTerm = 1;
```

Variables (2)

- **Qualifiers: const, volatile, extern, register, static**

- **Stages:**
  - ***declaration, definition***

Example:

int prevTerm = 0, crtTerm = 1, nextTerm = 1;

extern int prevTerm, crtTerm, nextTerm;

Types

- **Built-in** (primitive types: int, float, double, char, ...)
- **C++ Standard Library Provided Types** (strings, vector, ...)
  - Extend Basic Language
- **Programmer Defined Types** (class)

Built-in

- void, bool, int, float, double, char (page 29 / "Navigating C++")

- enum, struct, union (page 34, 107 / "Navigating C++")
  - enum fewColors { RED, GREEN, BLUE } ;
  - struct Person { string firstName; string lastName; } ;
  - union Role { struct Manager; struct Programmer; } ;

- typedef
  - typedef *type* aliasName;

Arrays

- **One-dimensional or multidimensional arrays**
  type name[size] = { init_list };
  type name[size1][size2]...[sizeN] = { { ... { ... }, { ... }, ..., { ... } ... }}

- **Usage:**
  name[el]
  name[el1][el2]...[elN]

- **Relation Pointers** (***Hold adresses of other objects - indirect manipulations***)

Example (e4.cc)

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

int main () {
   int dim1 = 2, dim2 = 2;

   // Read matrix dimensions
   cout << "Enter the number of rows: ";  cin >> dim1;
   cout << "Enter the number of columns: ";  cin >> dim2;

   // 2-dimenstions matrix definition
   int M[dim1][dim2];

   // Double cycle to read each element
   for (int i = 0; i < dim1; i++)
      for (int j = 0; j < dim2; j++) {
       cout << "M[" << i << "][" << j << "] = ";   cin >> M[i][j];
      }
   return 0;
}
```

Example (e4.cc)

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

int main () {
   int dim1 = 2, dim2 = 2;

   // Read matrix dimensions
   cout << "Enter the number of rows: ";  cin >> dim1;
   cout << "Enter the number of columns: ";  cin >> dim2;

   // 2-dimenstions matrix definition
   int M[dim1 * dim2];

   // Double cycle to read each elements
   for (int i = 0; i < dim1; i++)
      for (int j = 0; j < dim2; j++) {
       cout << "M[" << i << "][" << j << "] = "; cin >> *(M+i*dim1+j);
      }
   return 0;
}
```

Extensions

- Standard Template Library
  - string (string msg = " Hello! ";)
  - vector (vector<int> vec[100];)
  - pair (pair <string, string> author ("James", "Joyce");)

- Other user defined and distributed with the compiler

Programmer Defined

- *class*
- *enum*
- *struct*
- *union*

Example (e5.cc)

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

// New defined type using class
class myType {
public:
  int age;
};

// New type using struct
struct myType_st {
  int age;
};

// New type using union
union myType_un {
  int ID;
  char Name[20];
};
```

Example (e5.cc / 2)

```
int main () {
  char *pchrMessage = " Hello World ! ";
  string strMessage = " Hello World ! ";

  // User program memory management
  cout << "string type message: " << pchrMessage << endl;

  // Advantage using string type: automatic memory management
  strMessage += "Happy to see you again !";
  cout << "char pointer message: " << strMessage << endl;

  // Play with programmer defined types
  myType var; cin >> var.age;
  struct myType_st var2; cin >> var2.age;
  union myType_un var3; cin >> var3.ID;

  return 0;
}
```

Expressions

- Composed of one of more operands and operators

Example:

5

" Hello "

20 + 4 * a

- Possible classification of operators:
  - Arithmetic
  - Logic & Relational
  - Assignment
  - Other: increment, decrement, memory management, ...

Arithmetic

^,

*,

/,

%,

+,

-,

Logic & Relational

!,
<,
<=,
>,
>=,
==,
!=,
&&,
||

- Result is a bool

Assignment

- Provides a new value (change a state): lvalue = rvalue
  - int ival;
  - ival = 23;

- Compound assignment operators:
  - +=, -=, *=, /=, %=, <<=, >>=, &=, ^=, |=

Others

- **increment & decrement (>> *and* <<)**
- **? : (*3 == 5 ? true : false*)**
- **sizeof (*sizeof (int)*)**
- **new & delete (*dynamic memory allocation*)**
- **comma (*series of expressions*)**
- **Bitwise (~, <<, >>, &, ^, |, &=, ^=, |=)**
- **Bitset (*bitset<32> bitvec; bitvec.any ();*)**

Precedence

- scopes ::
- selectors . -> [], ()
- postfix increment ++ --
- conversion const_cast dynamic_cast, static_cast
- sizeof
- prefix increment ++ --
- bit operations
- new, delete
- arithmetic operations

Functions

- **A step for group program statements**

- **Declaration:**

  type function_name (type arg1, type arg2, ..., type argN);

- **Definition:**

  type function_name (type arg1, type arg2, ..., type argN)
  compound_statement

Example (e6.cc)

```
#include <iostream>
using namespace std;
// Additional function declaration
void help (int argn, char *argv[]);
void about ();
// Main function
int main (int argn, char *argv[]) {
  // Version
  if ((argn == 2) && !strcmp (argv[1], "-v")) { about (); return -1; }
  // Correct invocation
  if ((argn != 4) || (!strcmp (argv[1], "-h"))) { help (argn, argv); return -1; }
  // Other processings
  return 0;
}
// Help function
void help (int argn, char *argv[]) {
  cout << "Usage: " << argv[0] << " {-h | -a | three arguments} " << endl;
}
// About function
void about () {
  cout << " Simple test program for cs11600" << endl;
}
```

Homework

- Due date: January 10, 8:00pm
  - Problem 1/ Page 80 (Navigating C++)
  - Problem 3 / Page 80 (Navigating C++)
  - Write a C++ program that displays the Fibonnaci sequence up to the 100th term in reverse order

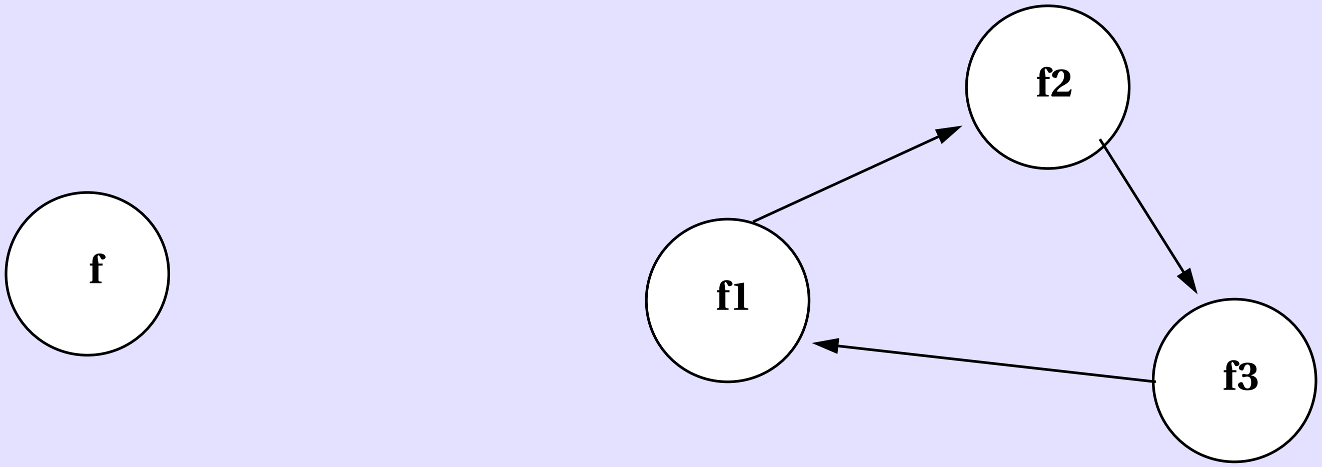
Next Time

- Recursivity
- More about Pointers
- Dynamic Allocation of Memory

Function Support

Recursivity

- a method (expression) which
  - calls itself (immediate recursion)
  - is in a potential cycle of function calls



Recursivity (2)

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

Recursivity (3)

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

- **iterative solution ?**

Recursivity (4)

- **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 (e7.cc)

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

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;
}
int main () {
  int A, B;
  // Read input
  cout << "Enter two integers: ";
  cin >> A >> B;
  // Write result
  cout << "gcd (" << A << ", " << B << ") = " << gcd_iter (A, B) << endl;
  return 0;
```

Efficiency

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

Recursivity Support