Last Lecture

- Describe 3 algorithms for tackling the connectivity problem
  - Quick Find
  - Quick Union
  - Weighted Quick Union
Pictorial Comparison
Quick Find   Quick Union   Weighted
```c
// Data Initialization
for (i = 0; i < N; i++)
    a[i] = i;

// Loop through connections
while (read_connection(p, q))
{
    // Check that input is within bounds
    if (p < 0 || p >= N || q < 0 || q >= N) continue;

    if (a[p] == a[q]) continue; // FIND operation

    set = a[p];                   // UNION operation
    for (i = 0; i < N; i++)
        if (a[i] == set)
            a[i] = a[q];

    print_connection(p, q);
    unique_connections++;
}
```
Quick Union in C

// Loop through connections
while (read_connection(p, q))
{
    // Check that input is within bounds
    if (p < 0 || p >= N || q < 0 || q >= N) continue;

    // FIND operation
    i = a[p];
    while (a[i] != i)
    {
        i = a[i];
    }

    j = a[q];
    while (a[j] != j)
    {
        j = a[j];
    }

    if (i == j) continue;

    // UNION operation
    a[i] = j;

    print_connection(p, q);
    unique_connections++;
}
Weighted Quick Union in C

```c
for (i = 0; i < N; i++)        // Initialize weights
    weight[i] = 1;

while (read_connection(p, q))   // Loop through connections
{
    // Check that input is within bounds
    if (p < 0 || p >= N || q < 0 || q >= N) continue;

    i = a[p];               // FIND operation
    while (a[i] != i)        //
        i = a[i];           // Identify group to which items p and q
    j = a[q];               // belong, by tracing up their respective
    while (a[j] != j)        // chains
        j = a[j];           //
    if (i == j) continue;   //

    if (weight[i] < weight[j])       // UNION operation
        { a[i] = j; weight[j] += weight[i]; }  //
    else                                // Link smaller group to
        { a[j] = i; weight[i] += weight[j]; }  // the larger one ...  

print_connection(p, q);
unique_connections++;
}
```
Today

- How to compile and debug C programs
  - On Windows, with Visual Studio
  - On Unix (and Macs!), with GCC / GDB

- Basics of Programming in C
  - General organization of C programs
  - C function libraries
Brief History of C

- C was developed by Dennis Ritchie at Bell Labs (1969 – 72)
  - Support the new UNIX operating system
  - Successor to B and BCPL
- Strongly typed language
- Dynamic memory allocation
- User defined data structures
The Modern C/C++ Languages

- High level languages
  - Relatively easy to understand

- Portable language
  - Work on desktop computers, mainframes and mobile phones

- Computers execute much more detailed, "lower-level" instructions …

- … compilers perform the necessary translation.
Anatomy of C Program

- A collection of functions
  - Receive a set of parameters
  - Declare local variables
  - Carry out processing
  - Return a value

- main() function
  - Called to start the program
A C function definition

```c
# Each function has a type
# Each function argument has a type
# Each local variable has a type

type function(argument_list)
{
    variable_declarations;

    statements;
}
```
C libraries

- Most programs are not built from scratch

- Rely on pre-existing collections of functions
  - Standard C Library, C++ Templates, KDE/Qt, GNU Scientific Library…

- Header (.h) files describe functions in these collections
  - Accessed through `#include` statements
/* C code is stored in .c or .cpp files */

#include <stdio.h>

int main()
{
    printf("Hello, I am a program ...
");
    printf("... And I don't do much.
");

    return 0;
}
Now I have a program ... ... how do I run it?
In a UNIX Environment…

- Many of you will end up running your programs in a Unix or Linux server

- GNU tools are widely available and quite popular
  - Start with a text editor, type in your code, then use …
  - GCC / G++ to compile code
  - GDB to debug and test code
  - GPROF to collect performance metrics

- … this approach has seriously steep learning curve.

- I recommend a more friendly setting …
- … write and debug code in a Windows PC or a Mac.
In a Windows PC or Mac

- Integrated toolsets exist combine editing, compiling, debugging and extensive documentation

- Microsoft Visual Studio / Visual C++
  - Discounted versions available through the University

- Turbo C++ Explorer
  - Free version of Borland C++ Builder
  - Available from [www.turboexplorer.com](http://www.turboexplorer.com)

- Xcode
  - Included in Mac OS X (may have to find original DVD)
Getting Started: Create a New Project

- The first step is to create a new project.
- Throughout the course, we will be focusing on data manipulation and analysis using console applications.
- So, we’ll use a “Console Application” template for our projects…
File | New ...

C++ Builder

Visual Studio
Very Basic C Program

/* C code is stored in .c or .cpp files */

#include <stdio.h>

int main()
{
    printf("Hello, I am a program ...\n");
    printf("... And I don’t do much.\n");

    return 0;
}
Running the Program ...

C++ Builder

Visual Studio
Errors?

- Compiler will list problems in your code and you should be able to click on each error message to identify problem ...

- ... sometimes a single problem can lead to many error messages.
Making our program more... “Useful”... “Complicated”...
Variables in C

- Must be declared before use
- Each variable has a specific type
  - integer
  - floating point
  - character
- Names are case-sensitive
Another C Program

#include <stdio.h>

int Multiply(int x, int y)
{
    int product = x * y;

    return product;
}

int main()
{
    int x = 2;

    printf("%d * %d = %d\n", x, x, Multiply(x, x));

    return 0;
}
Executing Code Line by Line...
(TIP: place mouse over variables to see their contents)

C++ Builder

Visual Studio
Basic Data Types in C

- **Integer data types**
  - `int`, `short`, `long`

- **Floating point data types**
  - `float`, `double`

- **Character types**
  - `char` (which is also an integer!)

- Pointers and user-defined types are also available
Integers

- For most purposes the \texttt{int} type will do
  - \texttt{unsigned int} for strictly positive quantities
  - \texttt{long long} data type for storing large integers

- Typically, store up to 31 or 63 digits
  - in base 2
  - plus one digit for sign
  - range is about -2.1 to 2.1 billion (32 bit)
Counting digits in an int ...

```c
int count_integer_bits()
{
    int bits = 0;
    int integer = 1;

    while (integer != 0)
    {
        bits++;
        integer = integer * 2;
    }

    return bits;
}
```
Floating point numbers

- Stored as exponent, mantissa and sign
  - Representation varies between machines

- Limited range and precision
Floating point data

- Stored in exponential notation
  - In base 2

- Has limited accuracy
  - Computing two similar quantities and evaluating their difference can be especially inaccurate

- Greater range than integer data
  - Exact for small integers
Measuring accuracy of a double

/* Calculate precision of double */
double precision()
{
    double e = 1.0, temp;

    do {
        e = e * 0.5;
        temp = 1.0 + e;
    } while (temp > 1.0);

    return e * 2.0;
}
Arithmetic has limited precision

- You just saw examples where:
  - $a \times 2 == 0$ and $a > 0$...
  - $a + b == a$ and $b > 0$...

- In some applications, these limits of arithmetic precision can make a lot of difference!

- This is a feature of most computer applications
Flow Control Statements...

- Allow programs to make choices based on input or results of previous operations

- Most interesting programs will depend on these...

- Examples ...
  - `if ... else ...
  - `do ... while ...
  - `while ...
  - `for ...
if ... else ...

if (expression)
  statement1;
else
  statement2;

- When expression is true (or nonzero) statement1 is executed; otherwise statement2 is executed.
Example

```c
void Compare(int a, int b)
{
    if (a == b)
        printf(“Values Match!\n”);
    else
        printf(“Values are different!\n”);
}
```
do … while …

do

statement;

while (expression);

- statement is executed until expression evaluates to false (or zero).
- statement is executed at least once.
Example: Measuring Precision for a `double`

```c
/* Calculate precision of double */
double precision()
{
  double e = 1.0, temp;

  do {
    e = e * 0.5;
    temp = 1.0 + e;
  } while (temp > 1.0);

  return e * 2.0;
}
```
while ...

while (expression)
  statement;

- statement is executed while expression evaluates to true.
- statement may never be executed.
Example: Counting digits in an int ...

```c
int count_integer_bits()
{
    int bits = 0;
    int integer = 1;

    while (integer != 0)
    {
        bits++;
        integer = integer * 2;
    }

    return bits;
}
```
for

```c
for ( initialization; condition; increment )
    statement;
```

- **Executes** initialization.

- **While condition is true:**
  - Execute statement.
  - Evaluate increment.

- statement **may never be executed.**
Example:
Searching for value in a list...

```c
int search(int a[], int value, int start, int stop)
{
    // Variable declarations
    int i;

    // Search through each item
    for (i = start; i <= stop; i++)
    {
        if (value == a[i])
        {
            return i;
        }
    }

    // Search failed
    return -1;
}
```
break and continue

- **continue**
  - Re-evaluates loop condition.
  - If not finished, start a new cycle.

- **break**
  - Stop looping early.
# Some Standard C Libraries

<table>
<thead>
<tr>
<th>Header File</th>
<th>Functionality</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctype.h</td>
<td>Information about characters</td>
</tr>
<tr>
<td>float.h</td>
<td>Information about floating point</td>
</tr>
<tr>
<td>limits.h</td>
<td>Information about integers</td>
</tr>
<tr>
<td>math.h</td>
<td>Common mathematical functions</td>
</tr>
<tr>
<td>stdio.h</td>
<td>Basic input / output functions</td>
</tr>
<tr>
<td>stdlib.h</td>
<td>Kitchen Sink!</td>
</tr>
<tr>
<td>string.h</td>
<td>String manipulation functions</td>
</tr>
<tr>
<td>time.h</td>
<td>Time</td>
</tr>
</tbody>
</table>
**math.h, Mathematical Functions**

- double exp(double x);
  - exponential of x
- double log(double x);
  - natural logarithm of x
- double log10(double x);
  - base-10 logarithm of x
- double pow(double x, double y);
  - x raised to power y

- double sin(double x);
- double cos(double x); ...
  - Standard trigonometric functions

- double sqrt(double x);
  - square root of x
- double ceil(double x);
  - smallest integer not less than x
- double floor(double x);
  - largest integer not greater than x
- double fabs(double x);
  - absolute value of x
Important Library Functions

- `<stdio.h>`
  - Input and output

- `<stdlib.h>`
  - Basic random numbers and memory allocation
Input / Output Functions

- `<stdio.h>`
- Default
  - `int printf(char * format, ...)`;
  - `int scanf(char * format, ...)`;
- File based functions
  - `FILE * fopen(char * filename, char * mode)`;
  - `int fclose(FILE * file)`;
  - `int fprintf(FILE * file, char * format, ...)`;
  - `int fscanf(FILE * file, char * format, ...)`;
printf

- Writes formatted output

- Format string controls how arguments are converted to text
  - Parameters are printed as specified in % fields
    - \%[flags][width][.precision]type
  - Otherwise, string is quoted
printf fields

- **Flags:**
  - “-” to left justify result
  - “+” to show sign in positive numbers

- **Width**
  - Minimum number of characters to print

- **Precision**
  - Number of digits after decimal (for floating point)
  - Maximum number of characters (for strings)

- **Type**
  - “s” for strings
  - “d” for integers, “x” to print hexadecimal integers
  - “f” for floating point, “e” for exponential notation, “g” for automatic
**scanf**

- Reads formatted input

- Format string defines input interpretation
  - Each `%[type]` field is converted and stored

- Arguments should be addresses of variables where input is to be stored
Scanf fields

Field types
- “s” for strings
- “d” for int variables
- “lld” for long long variables
- “f” for float variables
- “lf” for double variables
# Example

```c
#include <stdio.h>

int square(int x)
{
    return x * x;
}

int main()
{
    int number;

    printf("Type a number:");
    scanf("%d", &number);
    printf("The square of %d is %d.\n", number, square(number));

    return 0;
}
```
Opening and closing files

- **FILE * fopen(char * filename, char * type);**
  - Opens file with *filename*
  - If type is “wt”, a text file is opened for writing
  - If type is “rt”, a text file is opened for reading
  - Types “rb” and “wb” are analogous for binary files
  - Returns NULL on failure

- **int fclose(FILE * file);**
  - Closes file
  - Returns 0 on success
# Example

```c
#include <stdio.h>

int sqr(int x)
{
    return x * x;
}

int main()
{
    int number;
    FILE * output;

    printf("Type a number:");
    scanf("%d", &number);

    output = fopen("results.txt", "wt");
    fprintf(output, "The square of %d is %d\n", number, sqr(number));
    fclose(output);

    return 0;
}
```
Basic Random Numbers

- `<stdlib.h>`

- `int rand()`
  - Sample a uniformly distributed random integer between 0 and RAND_MAX

- `void srand(int seed)`
  - Select the sequence of random numbers specified by seed
Weighted Quick Union in C

// Initialize random generator
srand(1234);

// Generate M random connections
while (count++ < M)
{
    // Pick random elements to connect
    p = rand() % N;
    q = rand() % N;

    // FIND operation
    for (i = a[p]; a[i] != i; i = a[i] ) ;
    for (j = a[q]; a[j] != j; j = a[j] ) ;
    if (i == j) continue;

    // UNION operation
    if (weight[i] < weight[j])
        { a[i] = j; weight[j] += weight[i]; } 
    else
        { a[j] = i; weight[i] += weight[j]; }

    printf("%d %d is a new connection\n", p, q);
}
Weighted Quick Union in C

// Initialize random generator
srand(1234);

// Generate M random connections
while (count++ < M)
{
    // This method generates better randomness in many computers
    p = (int) (rand() * 1.0 * N / (RAND_MAX + 1.0));
    q = (int) (rand() * 1.0 * N / (RAND_MAX + 1.0));

    // FIND operation
    for (i = a[p]; a[i] != i; i = a[i]) ;
    for (j = a[q]; a[j] != j; j = a[j]) ;
    if (i == j) continue;

    // UNION operation
    if (weight[i] < weight[j])
        { a[i] = j; weight[j] += weight[i]; }
    else
        { a[j] = i; weight[i] += weight[j]; }

    printf("%d %d is a new connection\n", p, q);
}
Today

- Organization of C programs
- Basic data types
- Standard libraries