

Programming in C
Quick Start!

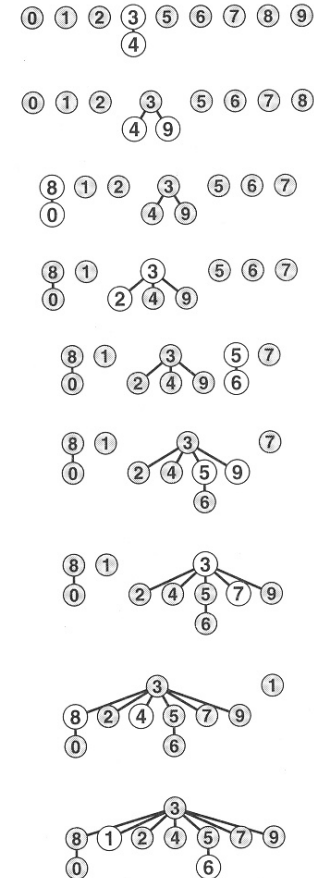
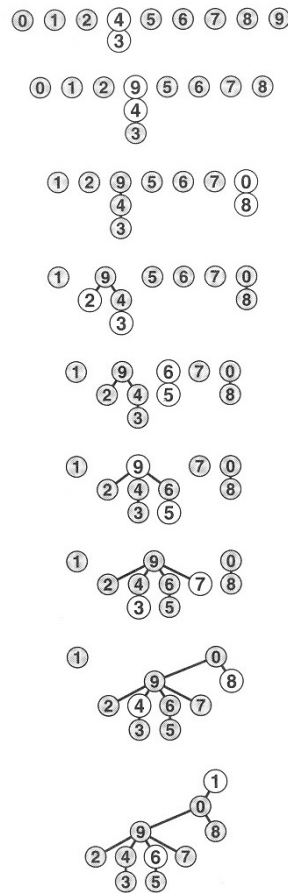
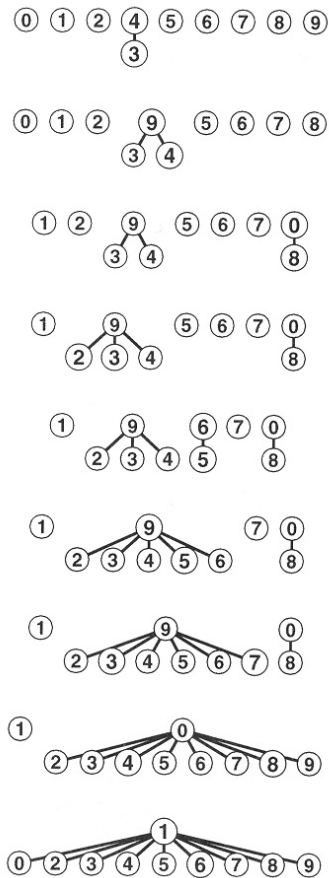
Biostatistics 615/815
Lecture 2

Last Lecture

- Describe 3 algorithms for tackling the connectivity problem
 - Quick Find
 - Quick Union
 - Weighted Quick Union

Pictorial Comparison

Quick Find Quick Union Weighted



Quick Find in C

```
// Data Initialization
for (i = 0; i < N; i++)
    a[i] = i;

// Loop through connections
while (scanf(" %d %d", &p, &q) == 2)
{
    // Check that input is within bounds
    if (p < 0 || p >= N || q < 0 || q >= N) continue;

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

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

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

Quick Union in C

```
// To start, place each element in its own group
for (i = 0; i < N; i++)
    a[i] = i;

// Loop through connections
while (scanf(" %d %d", &p, &q) == 2)
{
    // Check that input is within bounds
    if (p < 0 || p >= N || q < 0 || q >= N) continue;

    // 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
    a[i] = j;

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

Weighted Quick Union in C

```
// Initialize groupings and weights
for (i = 0; i < N; i++)
    weight[i] = 1, a[i] = i;

// Loop through connections
while (scanf(" %d %d", &p, &q) == 2)
{
    // Check that input is within bounds
    if (p < 0 || p >= N || q < 0 || q >= N) continue;

    // 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);
}
```

This Week

- **Basics of Programming in C**
 - General organization of C programs
 - C function libraries
- **How to compile and debug C programs**
 - On Windows, with Visual Studio
 - On Unix (and Macs!), with GCC / GDB

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 Language

- Portable language
 - C compilers are available for desktop computers, mainframes and mobile phones
- Very efficient
- C++ is the successor to C
 - Simplifies grouping of functions and related data

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

C libraries

- Most programs are not built from scratch
- Rely on pre-existing collections of functions
 - e.g. the Standard C library
- Header (.h) files describe functions in these collections
 - e.g. accessed through **#include** statements

A C function definition

```
type function(argument_list)
{
    variable_declarations;

    statements;
}
```

- Each function has a type
- Each function argument has a type
- Each local variable has a type

A simple C program

```
/* C code is stored in .c or .cpp files */  
  
#include <stdio.h>  
  
int main()  
{  
    printf("Hello, I am a program\n");  
  
    return 0;  
}
```

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;
}
```

Basic Data Types in C

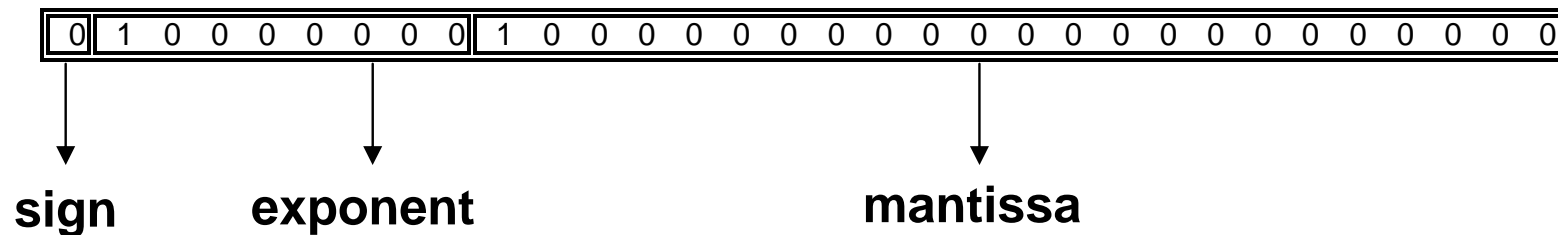
- Integer data types
 - `int`, `long`
- Floating point data types
 - `float`, `double`
- Character types
 - `Char`
- Pointers and user-defined types are also available

Integers

- For most purposes the `int` type will do
 - `unsigned int` for strictly positive quantities
 - `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)

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

Programming Constructs in C

- Function definitions and calls
- Compound statements
- Flow-control
 - `if ... else ...`
 - `do ... while ...`
 - `while ...`
 - `for ...`

Compound Statements

- C statements can be grouped with { }
- Optionally, each compound statement starts with local variable declarations
- Individual statements separated by “;”

if ... else ...

```
if (expression)
    statement1;
else
    statement2;
```

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

Example

```
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

```
/* 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

```
/* Calculate maximum integer */
double maximum_integer()
{
    int a = 2, b = 1, bits = 1;

    while (a > b)
    {
        b = a;
        a = a + a;
        bits++;
    }

    printf("Looks like a %d-bit computer\n", bits);

    return b;
}
```

for

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

- Executes initialization.
- While condition is true:
 - Execute statement.
 - Evaluate increment.
- statement may never be executed.

Example

```
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

Header File	Functionality
<code>ctype.h</code>	Information about characters
<code>float.h</code>	Information about floating point
<code>limits.h</code>	Information about integers
<code>math.h</code>	Common mathematical functions
<code>stdio.h</code>	Basic input / output functions
<code>stdlib.h</code>	Kitchen Sink!
<code>string.h</code>	String manipulation functions
<code>time.h</code>	Time

`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

```
#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

```
#include <stdio.h>

int square(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, square(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

Thursday: Executing C Code

- C is a high level language
 - Relatively easy to understand
- Computer CPUs execute much more detailed, "lower-level" instructions
- A compiler performs the necessary translation...

Working in a UNIX Environment

- **GCC / G++**
 - Compile code
- **GDB**
 - Debug and test code
- **GPROF**
 - Collect performance metrics

GCC

- GCC is a free C compiler
 - GNU C Compiler
- Versions available for
 - Linux
 - Unix
 - Mac
 - Windows
- Developed by Free Software Foundation



Working in a Windows Environment

- Good integrated toolsets exist
- Good options include:
 - Microsoft Visual Studio / Visual C++
 - Discounted version available through the University
 - Turbo C++ Explorer
 - Free C/C++ compiler, www.turboexplorer.com

