Programming in C
Quick Start!

Biostatistics 615
Lecture 4
Last Lecture ...

- Analysis of Algorithms
- Empirical Analysis
- Mathematical Analysis
  - Big-Oh notation
Today

- Basics of programming in C
  - Syntax of C programs
  - C function library
- Representation of numbers in a computer ...
Very Short 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
Data Types in C

- Integer data types
  - int, long
- Floating point data types
  - float, double
- Character types
  - char
- Pointers
  - int *, double *, char *
- User defined types
  - struct (short for structure)
- Void type
Integers

For most purposes the int type will do

- *unsigned int* for strictly positive quantities

Typically, store up to 31 or 63 digits

- in base 2
- plus one digit for sign
- range -2.1 to 2.1 billion (32 bit)
Floating point

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

- Limited range and precision
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 declaration

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

#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;
}
Another C Function

```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;
}
```
Programming Constructs in C

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

- C statements can be grouped with `{ }`
- Each compound statement starts with optional local variable declarations
- Individual statements separated by “;”
if ... else ...

```plaintext
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.
/* 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 */
int 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

// Search through each item
for (i = start; i <= stop; i++)
    if (value == a[i])
        return i;
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>
Arithmetic Limits...

- `<float.h>`
  - DBL_MIN
  - DBL_MAX
  - DBL_EPSILON

- `<limits.h>`
  - INT_MIN
  - INT_MAX
Example ...

```c
#include <float.h>
#include <limits.h>
#include <stdio.h>

int main()
{
    printf("The largest double is: \%e\n", DBL_MAX);
    printf("The smallest double is: \%e\n", DBL_MIN);
    printf("The precision of a double is: \%e\n\n", DBL_EPSILON);

    printf("The largest integer is: \%d\n", INT_MAX);
    printf("The smallest integer is: \%d\n", INT_MIN);

    return 0;
}
```
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
Example ...

```c
#include <math.h>
#include <stdio.h>

int main()
{
    double N;

    printf("Table of running times as a function of N\n\n\n");

    printf("%10s %10s %10s %10s %10s\n", "", "O(N)", "O(N log N)", "O(N*N)", "O(e^N)");
    for (N = 1; N < 20; N *= 2)
        printf("%9.0fN %10.1f %10.1f %10.1f %10.1f\n", N, N, N * log(N) / log(2), N*N, exp(N));

    return 0;
}
```
## Example Output

**Table of running times as a function of N**

<table>
<thead>
<tr>
<th>N</th>
<th>O(N)</th>
<th>O(N log N)</th>
<th>O(N*N)</th>
<th>O(e^N)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1N</td>
<td>1.0</td>
<td>0.0</td>
<td>1.0</td>
<td>2.7</td>
</tr>
<tr>
<td>2N</td>
<td>2.0</td>
<td>2.0</td>
<td>4.0</td>
<td>7.4</td>
</tr>
<tr>
<td>4N</td>
<td>4.0</td>
<td>8.0</td>
<td>16.0</td>
<td>54.6</td>
</tr>
<tr>
<td>8N</td>
<td>8.0</td>
<td>24.0</td>
<td>64.0</td>
<td>2981.0</td>
</tr>
<tr>
<td>16N</td>
<td>16.0</td>
<td>64.0</td>
<td>256.0</td>
<td>8886110.5</td>
</tr>
</tbody>
</table>
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
  - Arguments are output in order as fields are encountered
    - %[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 controls text is stored in arguments
  - Input converted as fields are encountered
    - %type
**scanf fields**

- **Types**
  - “s” for strings
  - “d” for **int** variables
  - “ld” for **long** variables
  - “f” for **float** variables
  - “lf” for **double** variables

- Addresses of variables should follow format string
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));
}
```
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 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);
}
```
Today ...

- Brief introduction to C
- Some useful C library functions
- Introduced concept of floating point precision