McGill

COMP 208 – Computers in Engineering

# COMP 208
# Computers in Engineering

Lecture 13

Jun Wang
School of Computer Science
McGill University
Fall 2007

McGill

COMP 208 – Computers in Engineering

# FORTRAN to C

***Nathan Friedman***

***Fall, 2007***

# C

- A general-purpose language
- Powerful for system-level programming
- Small
- C++, Java are derived from C
- Examples:
  - UNIX / Linux
  - The Python language is implemented with C
  - Compilers
  - Databases
  - Networking
  - Graphics

# Remember our first Fortran program?

```
PROGRAM hello
  IMPLICIT NONE
  !This is my first program

  WRITE (*,*) "Hello World!"

END PROGRAM hello
```

# Concepts We Saw

- Blocks of code
  - Bracketed by keywords such as

    ```
    PROGRAM…END PROGRAM,
    FUNCTION...END FUNCTION,
    DO…END DO
    ```

- Program Block
  - A special block identifying the program
- Comments
  - From “!” to end of line

# Concepts We Saw

- Statements
  - Each statement begins on a new line
    - assignment statement
    - READ statement
    - WRITE statement
    - IF statement
    - DO statement
    - CALL statement
  - WRITE statement is part of language
  - Format of output is determined by compiler but can be specified by programmer

# Here's the C version

```
#include <stdio.h>

void main()
{
  /* This is my first program */

  printf("Hello World!\n");
}
```

# Hello World: a comparison

```
PROGRAM hello
  IMPLICIT NONE
  ! First program

  WRITE (*,*) "Hello World!"
END PROGRAM hello
```

```
#include <stdio.h>

/* first program */
void main()
{
  printf ("Hello World!\n");
}
```

- No IMPLICIT NONE. C variables must be explicitly declared
- Statements are terminated with ;

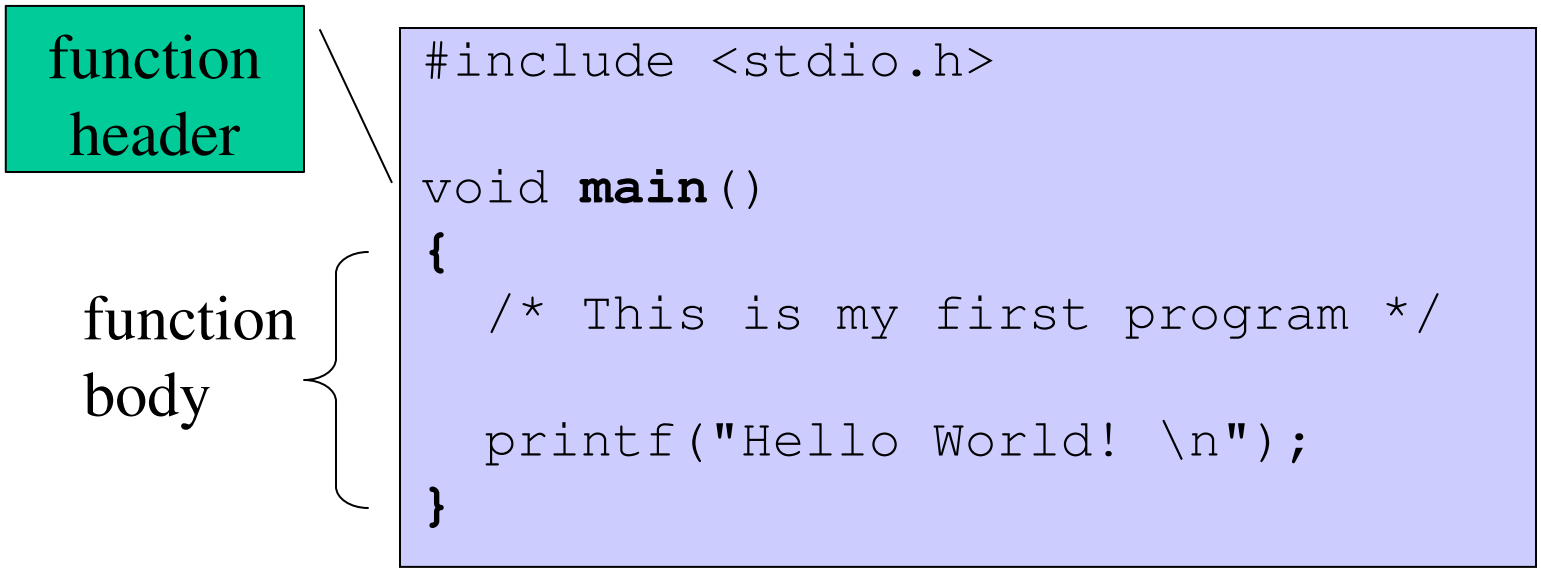
# The Concepts Revisited

- Blocks of code in Fortran
  - Bracketed by keywords such as `PROGRAM…END PROGRAM`, `FUNCTION...END FUNCTION`, `DO…END DO`
- Blocks of code in C
  - Bracketed by { … }

# The Concepts Revisited

- Program Block (main program) in Fortran
  - A special block identifying the program
- C program structure
  - A C program is just a collection of function definitions
  - One of the functions must be called **`main`**
  - When the program is run the `main` function is automatically invoked first - `main` is the starting point of the program

# Here's the C version



# Subroutines and Functions

- In Fortran there are two types of subprograms
  - Functions return a value
  - Subroutines perform an action
- In C, functions are the only kind of subprogram
  - If no value is to be returned, we specify a return type of **`void`**

# Here's the C version

```
#include <stdio.h>

void main()
{
  /* This is my first program */

  printf("Hello World! \n");
}
```

# The Concepts Revisited

- Comments in Fortran
  - From "!" to end of line
- Comments in C: 2 types
  - Block comments: enclosed by /* … */
  - Can appear anywhere in the program (even in the middle of other code on the same line)
  - One-line comments: from / / to the end of the line

# Here's the C version

```
#include <stdio.h>

void main()
{
  /* This is my first C
       program */
  // It has comments in it

  printf("Hello World! \n"); //short comment

  printf /*weird comment */ ("Hello!\n")
}
```

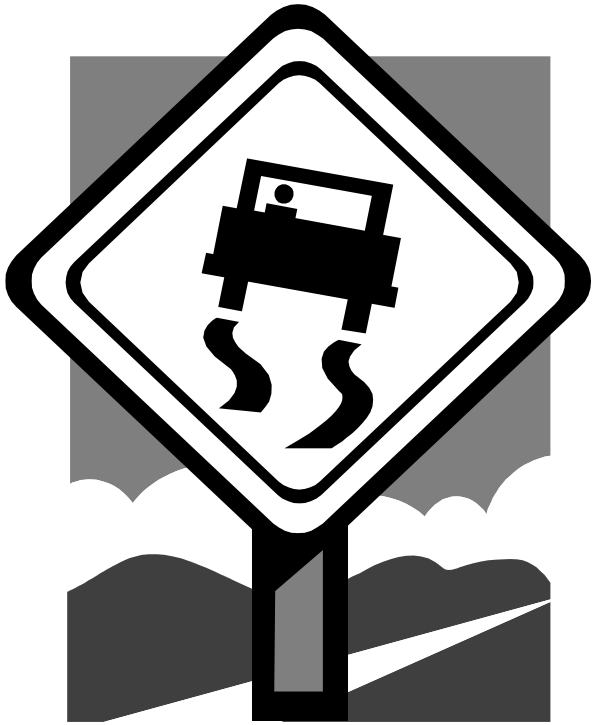
A block comment

A one-line comment

# The Concepts Revisited

- Statements in Fortran
  - Each statement begins on a new line
- Statements in C
  - C is a free format languages
  - Statements can appear anywhere and must be terminated with a ";"
  - A new statement can appear on the same line following the ;
  - The programmer is responsible for making the code readable to self/others

# Free Format Code



# Free Format Code

- C is truly free format
- Free format allows the programmer to write very obscure code
  - Hard for the human reader to understand
  - The compiler doesn't care as long as C syntax rules are obeyed
- Some programmers like to compete as to who can write the most obscure code. See the web site

  `http://www.ioccc.org/main.html`

# What Does This Do?

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

#define _                              ;double
#define void                           x,x
#define case(break,default)            break[O]:default[O]:
#define switch(bool)                   ;for(;x<bool;
#define do(if,else)                    inIine(else)>int##if?
#define true                           (--void++)
#define false                          (++void--)

char*O=" <60>!?\\\n"_ doubIe[010]_ int0,int1 _ Iong=0 _ inIine(int eIse){int
O1O=!O _ l=!O;for(;O1O<010;++O1O)l+=(O1O[doubIe]*pow(eIse,O1O));return l;}int
main(int booI,char*eIse[]){int I=1,x=-*O;if(eIse){for(;I<010+1;I++)I[doubIe-1]
=booI>I?atof(I[eIse]):!O
   switch(*O)x++)abs(inIine(x))>Iong&&(Iong=abs(inIine(x
)));int1=Iong;main(-*O>>1,0);}else{if(booI<*O>>1){int0=int1;int1=int0-
   2*Iong/0
[O]switch(5[O]))putchar(x-*O?(int0>=inIine(x)&&do(1,x)do(0,true)do(0,false)
case(2,1)do(1,true)do(0,false)6[O]case(-3,6)do(0,false)6[O]-3[O]:do(1,false)
case(5,4)x?booI?0:6[O]:7[O])+*O:8[O]),x++;main(++booI,0);}}}
```

# The "Hello World" program

```
#include
  <stdio.h>
    void
main
   (
)
{ printf (
"Hello World!\n"
)
  ;
}
```

```
#include <stdio.h>
void main ()
{
    printf("Hello World!\n");
}
```

Free-format means any number (>0) of whitespace, tab, or newline characters can be used to separate 2 tokens.

# The Concepts Revisited

- Fortran actions
  - WRITE statement is part of language
- C actions
  - Functions that return values or perform actions such as I/O are not an intrinsic part of C
  - C has many libraries that contain groups of related functions (such as I/O operations)
  - To access these libraries we must tell the compiler using a **`#include`** preprocessor command

# The Concepts Revisited

- I/O in Fortran
  - Format of output is determined by compiler
  - Can be specified by programmer using formats

- I/O in C
  - Functions that perform I/O are found in a library called **`stdio.h`**
  - The most common form is

    ```
    printf("format string", list of expressions);
    ```

  - The format string is required

# Here's the C version

```
#include <stdio.h>

void main()
{
  /* This is my first program */

  printf("Hello World! \n");
}
```

To use a library function, the header file for the function must be specified with `#include`.
Here, the header file for the function `printf` is `stdio.h`.

Header files usually have an extension `.h`.

# Case Sensitivity

Unlike Fortran which is case insensitive, C is case sensitive.

That means that `main` is different than `Main` in C

A variable named `first` is different than a variable named `First` or `FIRST`

# Roots of a Quadratic in C

```
#include <stdio.h>
#include <math.h>
void main() {
   float a, b, c;
   float      d;
   float root1, root2;
   scanf ("%f%f%f", &a, &b, &c);

   /* continued on next slide */
```

```
    if (a == 0.0)
        if (b == 0.0)
            if (c == 0.0)
                printf ("All numbers are roots \n");
            else
                printf ("Unsolvable equation");
        else printf ("This is a linear form, root = %f\n", -c/b);
    else {
        d = b*b - 4.0*a*c ;
        if (d > 0.0) {
            d = sqrt (d);
            root1 = (-b + d)/(2.0 * a) ;
            root2 = (-b - d)/(2.0 * a) ;
            printf ("Roots are %f and %f \n", root1, root2);
        }
        else if (d == 0.0)
            printf ("The repeated root is %f \n", -b/(2.0 * a));
        else {
            printf ("There are no real roots \n");
            printf ("The discriminant is %f \n", d);
        }
    }
}
```

# Roots of a Quadratic in C

```
#include <stdio.h>
#include <math.h>
void main() {
   float a, b, c;
   float       d;
   float root1, root2;
   scanf ("%f%f%f", &a, &b, &c);

   /* continued on next slide */
```

# Declarations

The concept is the same as in Fortran

Declarations tell the compiler:

- To allocate space in memory for a variable
- What "shape" the memory cell should be (i.e. what type of value is to be placed there)
- What name we will use to refer to that cell

Syntax of C declarations

```
<type> list of variable names;
```

# Variable Types

| Fortran | C |
|---|---|
| `INTEGER` | `int` |
| | `short` |
| | `long` |
| | `unsigned int (unsigned)` |
| | `unsigned short` |
| | `unsigned long` |
| `REAL` | `float` |
| | `double` |
| `CHARACTER` | `char` |
| | `unsigned char` |

# What's Missing?

- Fortran has a Logical type
- C doesn't
- Fortran uses logical expressions in if statements and to control some loops
- What does C do?
- We'll have to wait and see

```
#include <stdio.h>
#include <math.h>
void main() {
   float a, b, c;
   float       d;
   float root1, root2;
   scanf ("%f%f%f", &a, &b, &c);

   /* continued on next slide */
```

```
    if (a == 0.0)
          if (b == 0.0)
                  if (c == 0.0)
                          printf ("All numbers are roots \n");
                  else
                    printf ("Unsolvable equation");
          else printf ("This is a linear form, root = %f\n", -c/b);
    else {
          d = b*b - 4.0*a*c ;
          if (d > 0.0) {
                  d = sqrt (d);
                  root1 = (-b + d)/(2.0 * a) ;
                  root2 = (-b - d)/(2.0 * a) ;
                  printf ("Roots are %f and %f \n", root1, root2);
          }
          else if (d == 0.0)
                  printf ("The repeated root is %f \n", -b/(2.0 * a));
          else {
                  printf ("There are no real roots \n");
                  printf ("The discriminant is %f \n", d);
          }
    }
}
```

# Arithmetic Expressions

Expressions in C are very much like those in Fortran

Basic operations include `+,-,*,/`

The precedence rules are similar

Type requirements and conversions are similar to Fortran

Functions such as `sqrt()` are found in libraries such as `math.h`

# What's Missing

There is no exponentiation operator in C

# What's New?

## C has some other operators:

```
% is the mod operator, x%y is like the
  Fortran function mod(x,y)
```

## Increment(++), decrement(--)

```
i++   After using i, increase i by 1
i--   After using i, decrease i by 1
++i   Before using i, increase i by 1
--i   Before using i, decrease i by 1
```

# increment operator

- increment operator (++)
  - unary operator: adds one to its only operand
  - 2 forms:

| prefix form | postfix form |
|---|---|
| `++counter;` | `counter++;` |

  - equivalent to `counter = counter + 1;`
  - when used on its own, the 2 forms are equivalent

`++counter;` is the same as `counter++;`

Guess how the programming language C++ gets its name

# increment (cont'd)

– when used as part of a statement, the 2 forms behave differently

```
int x = 3, y;
y = ++x;
//x is 4; y is 4
```

**increment x, and then assign the value of x to y**

```
int x = 3, y;
y = x++;
//x is 4; y is 3
```

**assign the value of x to y, and then increment x**

`y = ++x;` is equivalent to

```
++x;
y = x;
```

whereas `y = x++;` is equivalent to

```
y = x;
++x;
```

# decrement

- increment operator (--)
  - unary operator: subtracts one from its only operand
  - 2 forms:

| prefix form | postfix form |
|---|---|
| `--counter;` | `counter--;` |

  - equivalent to `counter = counter - 1;`
  - when used on its own, the 2 forms are equivalent

`--counter;` is the same as `counter--;`

# decrement (cont'd)

– when used as part of a statement, the 2 forms behave differently

```
int x = 3, y;
y = --x;
//x is 2; y is 2
```

**decrement x, and then assign the value of x to y**

```
int x = 3, y;
y = x--;
//x is 2; y is 3
```

**assign the value of x to y, and then decrement x**

`y = --x;` is equivalent to

```
--x;
y = x;
```

whereas `y = x--;` is equivalent to

```
y = x;
--x;
```

# Example

This is not a good example!!
According to the C standard, the order of evaluation of the actual arguments of a function call is not specified.
Programs like this are not portable.

```
#include <stdio.h>
int main() {
   int x, y;
   x = 14;
   y = 5;

   printf(" %i \n %i  \n %i \n",
              ++x - y--, ++x - --y, x++ - y--);
   return 0;
}
```

```
 OUTPUT:
  14
  13
  9
```

Here, the actual arguments were evaluated from right to left.

# Example

The following expression

```
x = ((++z) - (w--)) % 100;
```

is equivalent to

```
z++;    /* or z=z+1 */
X = (z-w) % 100;
w--;   /* or w=w-1; */
```

The increment operators are actually more efficient