Computers in Engineering COMP 208

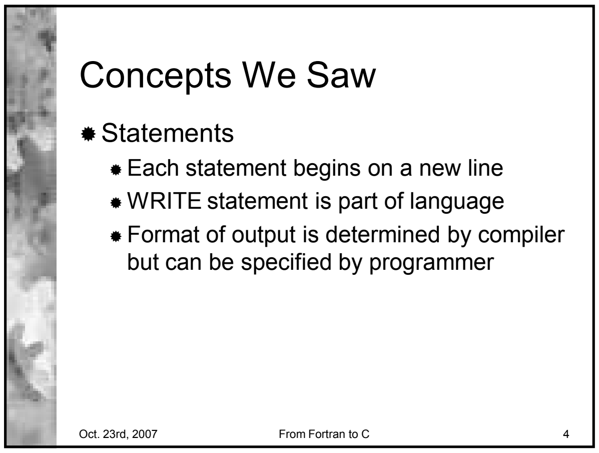
Moving From Fortran to C
Michael A. Hawker

Remember our first Fortran program?

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

  WRITE (*,*) "Hello, World!"

END PROGRAM hello
```

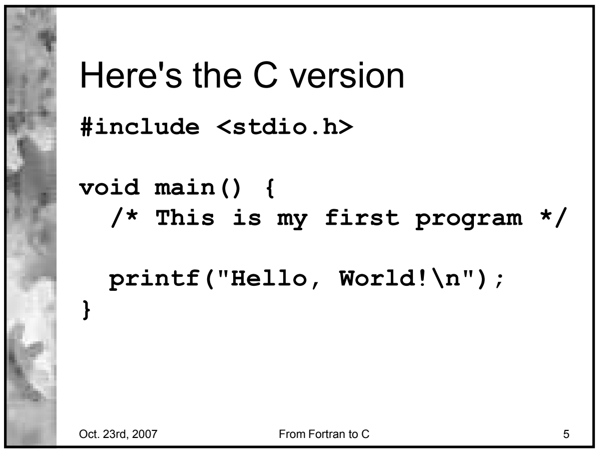
Concepts We Saw

- Blocks of code
  - Bracketed by keywords such as PROGRAM…END PROGRAM, DO…END DO
- Program Block
  - A special block identifying the program
- Comments
  - From “!” to end of line



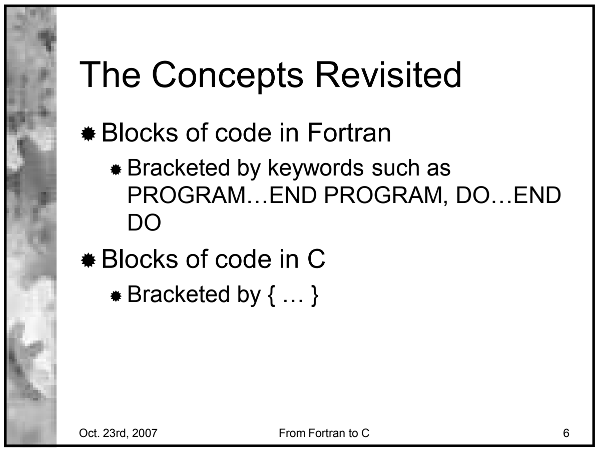
Concepts We Saw

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



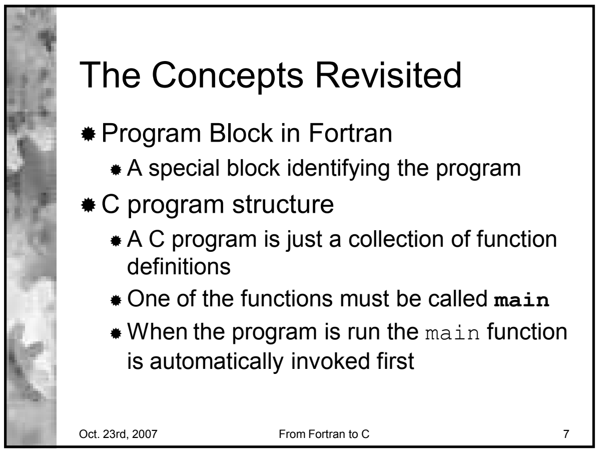
Here's the C version

```
#include <stdio.h>

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

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



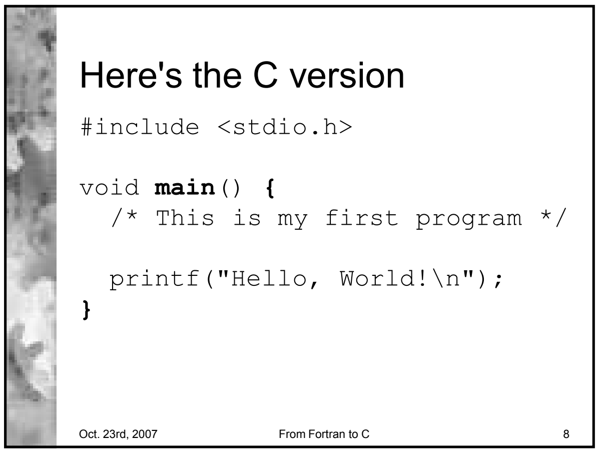
The Concepts Revisited

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



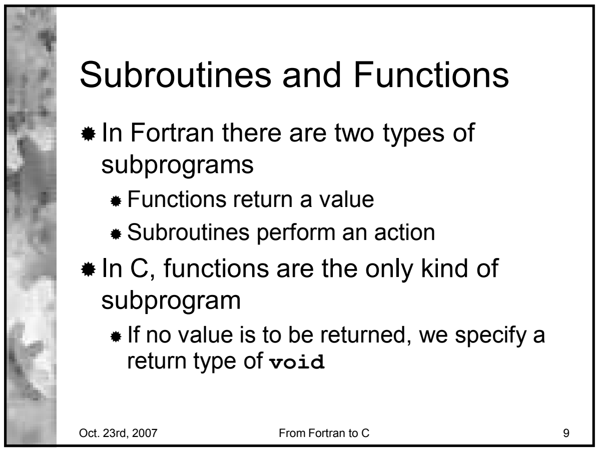
The Concepts Revisited

- Program Block 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



Here's the C version

```
#include <stdio.h>

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

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



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
  - Enclosed by /* … */
  - Can appear anywhere in the program (even in the middle of other code on the same line)
  - Comments preceded by // include everything to the end of the line

Here's the C version

```
#include <stdio.h>

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

  printf("Hello, World!\n");
  // It has comments in it
}
```

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 others

Free Format Code

- 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 Concepts Revisited

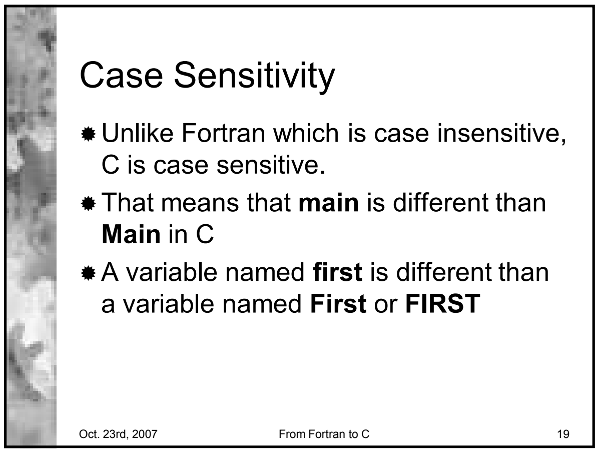
- Fortran actions
  - WRITE statement is part of language
- C actions
  - Functions that return values or perform actions such as I/O are not an intrinsic part of C
  - C has many libraries that contain groups of related functions (such as I/O operations)
  - To access these libraries we must tell the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

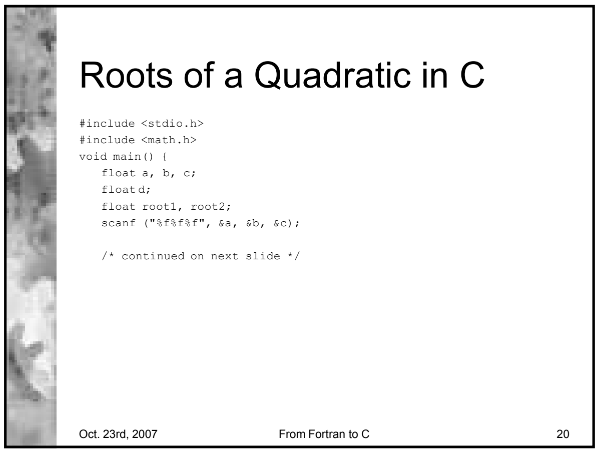
Here's the C version

```
#include <stdio.h>

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

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



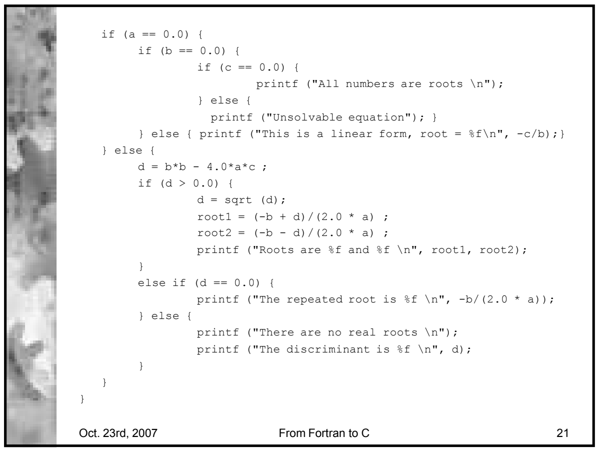
Case Sensitivity

- 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**

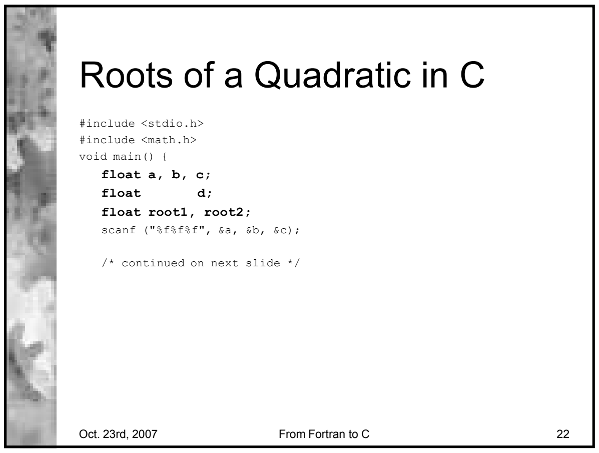


Roots of a Quadratic in C

```
#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 */
```

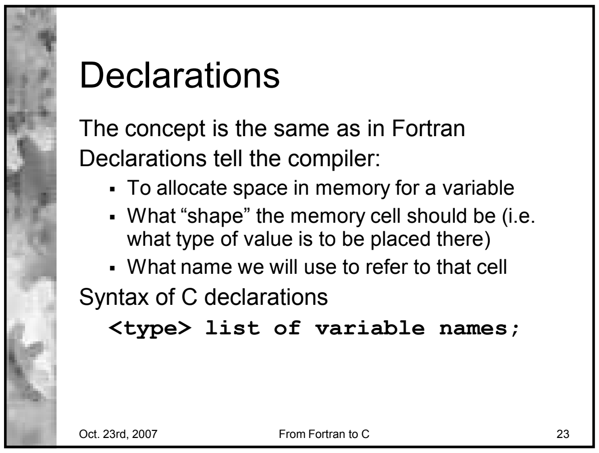


```
   if (a == 0.0) {
        if (b == 0.0) {
                if (c == 0.0) {
                        printf ("All numbers are roots \n");
                } else {
                  printf ("Unsolvable equation"); }
        } else { printf ("This is a linear form, root = %f\n", -c/b);}
   } else {
        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

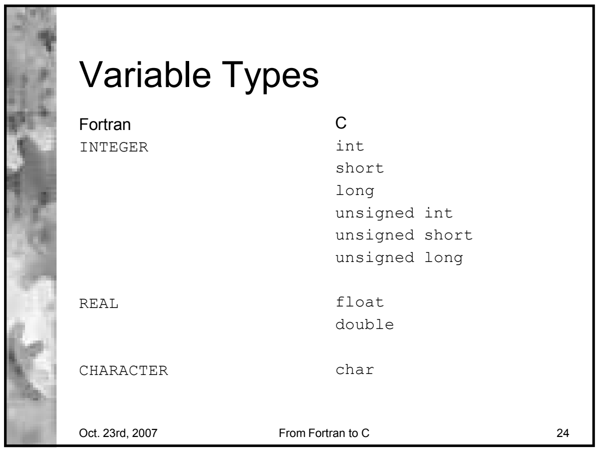
The concept is the same as in Fortran

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

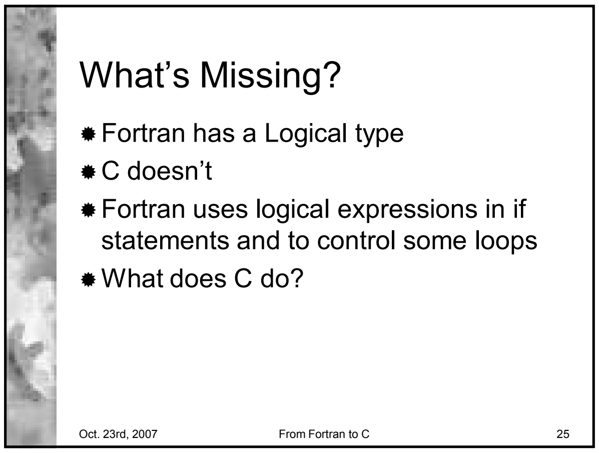
Syntax of C declarations

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



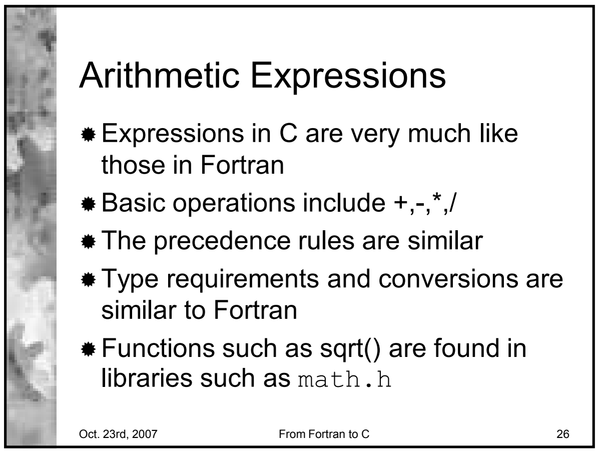
Variable Types

Fortran	C
`INTEGER`	`int`
	`short`
	`long`
	`unsigned int`
	`unsigned short`
	`unsigned long`
`REAL`	`float`
	`double`
`CHARACTER`	`char`



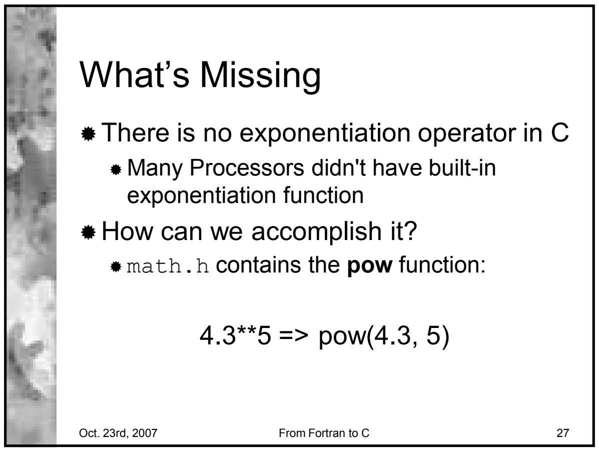
What's Missing?

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



Arithmetic Expressions

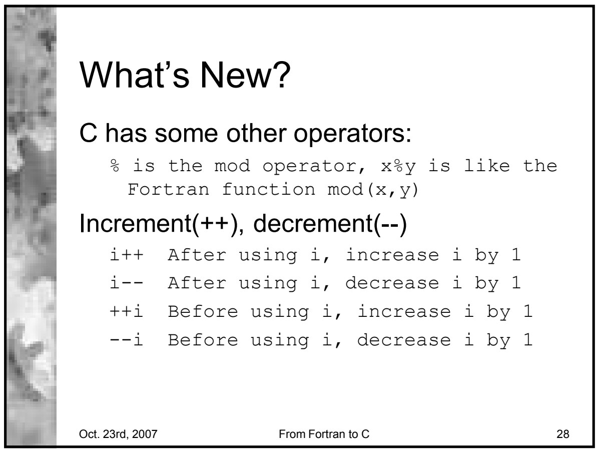
- Expressions in C are very much like those in Fortran
- Basic operations include +,-,*,/
- The precedence rules are similar
- Type requirements and conversions are similar to Fortran
- Functions such as sqrt() are found in libraries such as `math.h`



What's Missing

- There is no exponentiation operator in C
  - Many Processors didn't have built-in exponentiation function
- How can we accomplish it?
  - `math.h` contains the **pow** function:

4.3**5 => pow(4.3, 5)



What’s New?

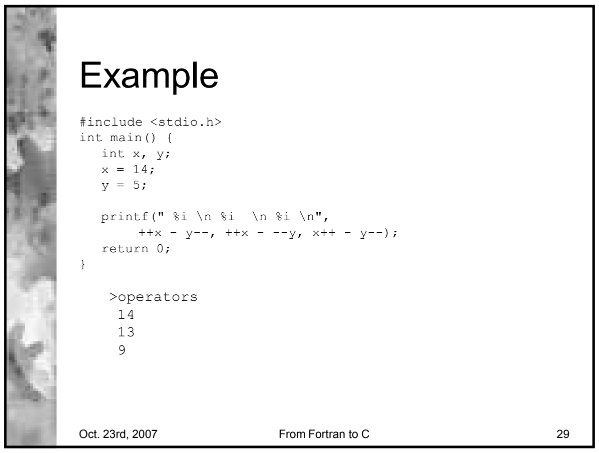
C has some other operators:

```
% 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
```

Oct. 23rd, 2007 From Fortran to C 28

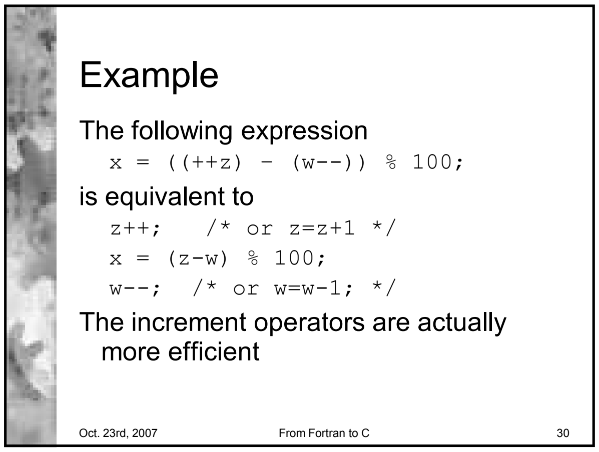


Example

```
#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;
}
```

```
>operators
 14
 13
 9
```

Oct. 23rd, 2007 From Fortran to C 29



Example

The following expression

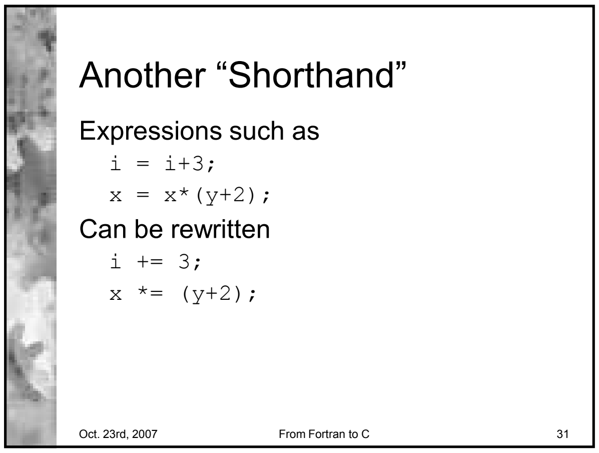
```
x = ((++z) - (w--)) % 100;
```

is equivalent to

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

The increment operators are actually more efficient

Oct. 23rd, 2007 From Fortran to C 30



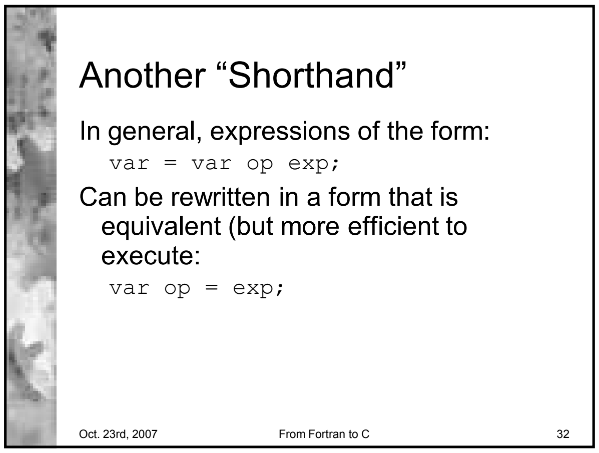
Another “Shorthand”

Expressions such as

```
i = i+3;
x = x*(y+2);
```

Can be rewritten

```
i += 3;
x *= (y+2);
```



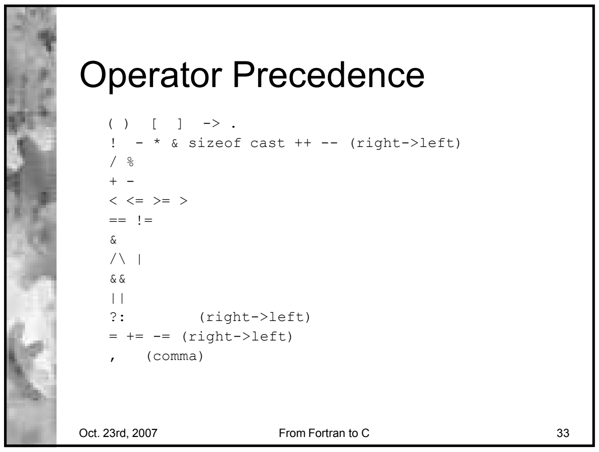
Another “Shorthand”

In general, expressions of the form:

```
var = var op exp;
```

Can be rewritten in a form that is equivalent (but more efficient to execute:

```
var op = exp;
```



Operator Precedence

```
( )  [  ]  -> .
!  - * & sizeof cast ++ -- (right->left)
/ %
+ -
< <= >= >
== !=
&
/\ |
&&
||
?:         (right->left)
= += -= (right->left)
,   (comma)
```

```
    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);
         }
    }
}
```

If Statements in C

Syntax:
```
    if (expression)
      statement_1
    else
      statement_2
```
Other forms:
```
        if (expression)
            statement
```
and
```
        if (expression)
          statement
        else if (expression)
          statement
        . . .
        else statement
```

```
if (expression) {
   statements
} else {
   statements
}

if (expression) {statements}

if (expression) {
   statements
} else if (expression) {
   statements
} else {
   statements
}
```

A Note On Syntax

- Grouping statements inside braces {…} makes them into a block set of statements
  - Easier to read
  - Easier to expand
- There is no marker to end the if statement
- Only a single statement can appear in each clause if no braces used

Wait a Second!

- There is no logical type in C
- This isn't like Fortran
- What is the expression supposed to evaluate to?
- What is the meaning of an if statement?

Semantics of "if" Statements

- Evaluate the expression
  - If it evaluates to any non-zero value, execute the first statement
  - If it evaluates to 0, execute the second statement (after the else)
- Go on to the statement following the if

Still To Come

- We will see how other Fortran features are reflected in C
  - Some are almost the same with syntactic modifications
  - Some have sometimes subtle semantic differences
- Later we will see concepts which do not have direct Fortran counterparts