McGill

COMP 208 – Computers in Engineering

COMP 208
Computers in Engineering

Lecture 14

Jun Wang
School of Computer Science
McGill University
Fall 2007

Review: basics of C

- C is case sensitive
- 2 types of comments: /* */, //
- A C program is a collection of functions
  - The `main` function is the starting point
  - If a function doesn't return anything, the return type should be `void`
- Blocks of code are enclosed in { }
- Operators: `+ - * / % ++ --`
- I/O operations (scanf/printf) provided by standard library. Header file `stdio.h` must be included

Another "Shorthand"

Expressions such as

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

Can be rewritten

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

Compound assignment

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;
```

This is called compound assignment.

Operator Precedence

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

If Statements in C

Syntax:

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

Each of the statement can be a simple statement or a block statement enclosed in { }

Other forms:

```
if (expression)
  statement
```

```
if (expression1)
  statement
else if (expression2)
  statement
...
else
  statement
```

A Note On Syntax

- There is no marker to end the if statement
- Only a single statement can appear in each clause
- Grouping statements inside braces {…} makes them into a single compound statement
- A compound statement (a group of statements in { }) can appear wherever a single statement can appear

examples

```
if (a > 0)
  b++;
```

=

```
if (a > 0)
{
  b++;
}
```

recommended!

```
if (a > 0)
  b++;
  c++;
```

=

```
if (a > 0)
{
  b++;
}
c++;
```

```
if (a > 0)
{
  b++;
  c++;
}
```

Nested `if` Statements

- The statement executed as a result of an `if` statement or `else` clause could be another `if` statement

- These are called *nested if statements*

Min of three

```
int num1, num2;
int num3, min = 0;
       ...

if (num1 < num2)
   if (num1 < num3)
      min = num1;
   else
      min = num3;
else
   if (num2 < num3)
      min = num2;
   else
      min = num3;
```

```
if (num1 < num2)
{
   if (num1 < num3)
      min = num1;
   else
      min = num3;
}
else
{
   if (num2 < num3)
      min = num2;
   else
      min = num3;
}
```

It is recommended to use { } to make the structure clearer.

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

1. Evaluate the expression
   a) If it evaluates to any non-zero value, execute the first statement
   b) If it evaluates to 0, execute the `else` clause, if any.
2. Go on to the statement following the if

```
if (3)
  printf("true");
else
  printf("false");
```

```
if (3.5)
  printf("true");
else
  printf("false");
```

McGill

COMP 208 – Computers in Engineering

From Fortran to C – Part 2

Nathan Friedman

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);
          }
    }
}
```

Quadratic Roots Revisited

- Let's make one slight change to the program:

```
    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);
          }
    }
}
```

Quadratic Roots Revisited

- Can you spot the change?
- What do you think the effect is?

What Happens Here?

The equivalent statement in Fortran would cause a syntax error

The expression `(a = 0.0)` is not of type logical

But C does not have a type "logical"

What happens in C?

The C Assignment Operator

In Fortran, assignment is a statement.
In C, assignment is an operator

The C Assignment Operator

In Fortran, assignment is a statement.
In C, assignment is an operator

- It can appear as part of an expression

```
if ((a = b+c) > d)
  f++;
```

=

```
a = b+c;
if (a > d)
  f++;
```

- When evaluated it causes a value to be assigned (as in Fortran)
- It also returns a value that can be used in evaluating the expression
- If it appears independently, with a ; after it this value is discarded

The C Assignment Operator

Syntax:

variable = expression

Semantics

1. Evaluate the expression
2. Store the value in the expression in the variable
3. Return the value to the expression

Assignment statement in C has a value (the value of the right-hand side expression).
In Fortran, assignment statements have no value.

Back to our example

- In C, `(a=0.0)` would assign 0 to a and then return 0 as the value of the expression
- The if condition, with value 0, would be taken as equivalent to "false"
- The else clause would be evaluated and when the root was calculated, there would be an attempt to divide by 0

? : – Conditional operator

- C has many operators and functions that are not part of Fortran
- They come in useful in many applications
- One of these is the "?:" operator

The ? Operator

- The ? (*ternary condition*) operator is a more efficient form for expressing simple if selection

- It has the following syntax:

```
e1 ?  e2 :  e3
```

- Semantics: `e1` is evaluated, if it is true (non-0), the value of the whole expression is `e2`; otherwise it is `e3`.

The ? Operator

To assign the maximum of a and b to z:

```
z = (a>b) ? a : b;
```

which is equivalent to:

```
if (a>b)
  z = a;
else
  z = b;
```

```
printf("Time = %d %s\n", n, (n==1) ? "hour" : "hours");
```

If n is 1, it prints
`Time = 1 hour`

If n is 3, it prints
`Time = 3 hours`

Loops in C

- The basic looping construct in Fortran is the DO loop
- In C, there is a more complex looping command called the **`for`** loop

The C `for` Loop

The C for statement has the following (deceptively simple) form:

```
for (e1; e2; e3)
   statement
```

The statement can be a block of statements inside braces { … }

The Fortran equivalent is:

```
e1
DO WHILE(e2)
   statement
   e3
END DO
```

or

```
e1
DO
   IF (.NOT. e2) EXIT
   statement
   e3
END DO
```

Semantics of the **for** loop

```
for  (e1; e2; e3) statement
```

1. *e1* is evaluated just once before the loop begins. It initializes the loop.

2. *e2* is tested at the **beginning** of each iteration. It is the termination condition. If it evaluates to 0 (false) the loop terminates. Otherwise the loop continues.

3. *e3* is evaluated at the end of each iteration. It is used to modify the loop control. It is often a simple increment, but can be more complex.

A C **for** Loop Example

```
main(){
   int x;
   for (x=3; x>0; x--)
       printf("x=%d\n",x);
}
```

...outputs:

x=3

x=2

x=1

What do these do?

```
for (i=1; i<10; i++)
  printf("%d %d %d\n", i, i*i, i*i*i);
```

```
1 1 1
2 4 8
3 9 27
...
9 81 729
```

```
for (x=3; ((x>3) && (x<9)); x++)

for (x=3,y=4; ((x>=3) && (y<9)); x++,y+=2)
   printf("yes");

for (x=0,y=4,z=4000; z; z/=10)
```

Comma operator

The While Statement

The while has the syntax:

```
while (expression)
   statement
```

Semantics:

- The expression is evaluated.
- If it evaluates to 0 (false) the loop terminates, otherwise the statement is executed and we repeat the evaluation.

Example

```
main()
{
    int x=3;
    while (x>0) {
        printf("x=%d\n",x);
        x--;
    }
}
```

...outputs:

```
x=3

x=2

x=1
```

While Statements

- The equivalence of a for loop and a while loop

```
for(e1; e2; e3)
   statement
```

```
e1;
while(e2)
{
   statement
   e3
}
```

- The while statement

```
while (e) s
```

is semantically equivalent to

```
for (;e;) s
```

`for` vs. `while`

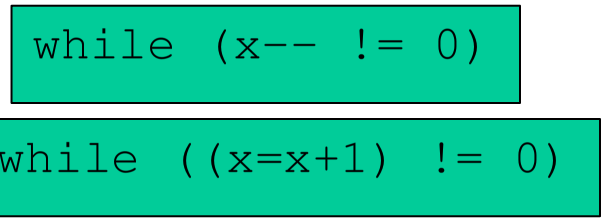
- Using Fortran parlance, for a definite iterator, we normally use the `for` loop
- For an indefinite iterator, we normally use the `while` loop.

What do these do?

- As with many constructs in C, the while statement can interact with other features to become much less clear:

```
while (x--) …;
while (x=x+1)…;
while (x+=5)…;
while (i++ < 10)…;

while ( (ch = getchar()) != 'q')
    putchar(ch);
```



Do-while

Both `while` and `do-while` execute a statement repeatedly and terminate when a condition becomes false

In the do-while, the testing is done at the end of the loop

```
do
   statement
while (expression)
```

Therefore, the loop body is executed at least once.

Break and Continue

The break statement is similar to the Fortran EXIT and allows us to exit from a loop

`break;` --exit from loop

The continue statement forces control back to the top of the loop

`continue;` --skip 1 iteration of loop

Using break and continue

```
while (scanf( "%d", &value ) == 1 && value != 0) {
  if (value < 0) {
       printf("Illegal value\n");
       break;     /* Abandon the loop */
  }
  if (value > 100) {
       printf("Invalid value\n");
    continue; /* Skip to start loop again */
   }
   /* Process the value (guaranteed between 1 and 100) */
   ....;
} /* end while when value read is 0 */
```

Inputting Values in C

In the program for computing the roots of a quadratic, we began by reading the values for the coefficients a, b and c

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 */
```

The Input Statement

Syntax:

```
scanf("format string",
      list of variable references) ;
```

Semantics:

- Each variable reference is has the form
  `&name`
- For now, ignore the & that appears in front of variable names
- It has to be there but we'll explain why later

The `&` is the "address of" operator.
So, `&a` is the address of the variable `a`.

parameter passing in C

- In C, function parameters are passed in a way known as "pass by value".
- The default way in Fortran is "pass by reference".

```
void fun(int x)
{
  x = x+1;
}
int main()
{
    int i = 1;
    fun(i);
    printf("%d\n", i);
}
```

If this were Fortran, i would be changed to 2. NOT in C though.

```
void fun(int *x)
{
  *x = *x+1;
}
int main()
{
    int i = 1;
    fun(&i);
    printf("%d\n", i);
}
```

In C, we have to pass the address of the variable