C Programming

for, do-while, switch,
break, continue

FOR STATEMENTS

Review of while Statement

```
int i = 0;
  while( i < 10 )
  {
      read_record();
      process_record();
      i++;
  }
```

Three Common Characteristics of a while Statement

- a condition
- an initial value for the variable used in the condition
- something that changes so that the condition can eventually be false

Example, Revisited

```
int i = 0;
  while( i < 10 )
  {
      read_record();
      process_record();
      i++;
  }
```

Initial value for i

Condition

Incrementing the variable so the condition can be false

The for Statement

Combines

those three elements for conciseness

Syntax:

- for(initialization; keep_going_condition; change_statement)

Example:

- for(i = 0; i < 10; i++)

Followed by the body of the for statement.

Rewriting the Example

```
int i = 0;
  for( i = 0; i < 10; i++ )
  {
      read_record();
      process_record();
  }
```

Benefits of Using for

1. All of the important stuff is in one place.

2. It’s harder to forget stuff.

Leaving Out Parts of a for

Semicolons
required

(missing condition is true by
default)

Thus, an infinite loop can be done with for(;;)

e.g.:

```
for( ; ; )
{
        print(“Infinite loop\n”);
}
```

do - while statements

A while statement's body may or may not be executed

e.g.

```
i = getNum();
// if user enters 10, body isn’t executed
while( i != 10 )
{
    // insert meaningful code here
}
```

Loop
at least
once

do-while Example

```
int i = 0;
do
{
    read_record();
    process_record();
    i++;
} while( i < 10 );
```

Note no semicolon here.

Note semicolon at end.

It’s always considered OK to have the while on the same line as the brace.

Body

executes

at least once

Remember to watch for:

the absence of semicolon at the top

and

the presence of the semicolon at the bottom

break and continue

break and continue Statements

break and *continue* affect control flow with loops

Need to get out of a loop right away?

break executes the first statement after the loop next.

Example of break

```
while( i < 10 )
{
    i = getNum();
    if( i == 0 )
    {
        break;
    }
    printf(“i is %d\n”, i);
}
printf(“Done\n”);
```

This loop will normally keep looping while i is less than 10.

It’ll print the value entered by the user before looping up to the condition.

Exception: if the user enters 0, the break statement will cause the program to go immediately to the green printf() outside of the loop’s body.

continue Statement

Just want to skip the body this time but not leave the loop?

continue will skip to the end but execute the condition to see if the loop should be terminated or not

Example of continue

```
while( i < 10 )
{
    i = getNum();
    if( i == 0 )
    {
        continue;
    }
    printf(“i is %d\n”, i);
}
printf(“Done\n”);
```

This loop will normally behave in the same way as the other example.

Exception: if the user enters 0, the continue statement will go immediately to the while loop’s condition.

break and continue statements

Example:

```
const int kTrue = 1; const int kFalse = 0; const int kSkipThisOne = 2;
int i = 0, status = kFalse;
do
{
        if( (status = read_record()) == kFalse)
        {
                break;
        }
        if( status == kSkipThisOne)
        {
                continue;
        }
        process_record();
        i++;
} while( i < 10 );
```

Important Note about break

There is no direct way

to

get out

of nested loops

switch statements

In certain situations,
you can replace
a series of
chained if-else statements
with
a switch statement involving cases.

"In

certain

situations"

Restriction #1

One side of the comparison must be the same for all if-else conditions.

- e.g.:

```
char c = 9;
if( c == 1 )
    // do something
else if( c == 10 )
    // do something else
else if( c == 15 )
    // do something else
else if( c == ‘A’ )
    // do something else
```

- In this example, the left-hand side only contains c.

Restriction #2

The other side of the comparison must contain literals or constants.

- In the example, the right-hand side contained 1, 10, 15, and ‘A’.

(but you can’t use const)

Restriction #3

The data type must be an integer-like data type, such as int, char, unsigned long, etc.

- Note: This does **not** allow for comparison using strings (later lecture).

If

your if-else chain satisfies all of these restrictions,

then

you can replace the chain with a switch statement.

The bodies of the if-else statements simply go after the appropriate case statements.

The bodies usually

do not

have braces around them

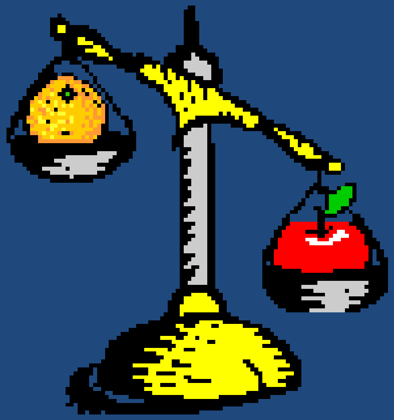
There is usually a break statement at the end of each body.

Example of a switch Statement

```
int i = 0, done = 0;
do
{
        switch( read_record() )
        {
        case  -1:
                break;
        case  0:
                done = 1;
                break;
        default:
                process_record();
                i++;
                break;
        } /* end switch */
} while( (i < 10) && (done == 0) );
```

In this example, all comparisons are against a return value from a function (read_record()).

The return value is compared against the constant values -1 and 0 to see if they are equal.



Each case is compared in turn,
until a match is found.

When a match is found,
the code following
the appropriate case statement is executed until
either
the end of the switch statement
or
a break statement.

break statements break you out of the switch statement only, not the outer loop.

default

If no match is found, the default case is used.

You can have, but do not require, a default case.

It is typically at the end of the switch statement but does not have to be.

Combining cases

You can have more than one case statement apply to code that is being executed.

For example, if you want to execute the same code whether the value you're comparing is 1, 2, 3, or 10, you simply put four consecutive case statements.

e.g.:

```
switch( i )
{
case 1: case 2: case 3: case 10:
    // code to execute
    break;
case 4:
    // more code
    break;
}
```

break Statements in switches

As stated earlier, break statements can be used to break out of switches.

If you do not have a break statement in the code that you're executing, the code corresponding with the next case will be executed!

This is typically a bad thing.

This is different from some other languages.

Intentional Fall-through

It is possible that you **do** want to leave the break statement out, so that the code that is attached to the next case statement is also executed.

You would do this if you had some code that applied to both conditions but you had more code that only applied to one.

e.g.

```
switch( severity )
{
case 10: case 9:
        emailSupervisor();
        /* intentional fall-through */
case 7: case 8:
        emailAttendant();
        break;
// more code goes here
```

In this example, you want to e-mail an attendant if severity is 7 or above but you **also** want to e-mail a supervisor if the severity is 9 or 10.

The comment in green is usually used to indicate that you didn’t just forget to put the break statement in.

Indentation of switch statements

Religious Issue!

How I indent a switch

```
switch( variable )
{
case 1:
case 2:
case 3:
        // stuff goes here
        break;
case 4:
        // stuff goes here
        break;
default:
        // stuff goes here
        break;
}
```

I use the principle that the code that is executed for each case should be one indentation level in from the switch statement

Another way to indent a switch

```
switch( variable )
       {
        case 1:
        case 2:
        case 3:
        // stuff goes here
            break;
        case 4:
        // stuff goes here
            break;
        default:
        // stuff goes here
            break;
       }
```

Others might use this indentation under the reasoning that they want to see where each case is (through indentation) and then see where the code for each case is (again, through another level of indentation).

I consider this to be double-indentation (and, thus, misleading) but I will not consider it wrong because it is generally accepted as a valid option.

That which must be never named

There is a goto statement in C

Never use it

Never

Never

Penalty for using

goto

in this course:

marks!

Summary

1. There are more control structures than just if and while.
2. Just never use goto.