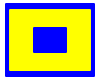


Control Flow

XXX <Flow diagrams of if, loops>

Outline

- Conditionals
  - if
  - if-else
  - switch
- Looping constructs
  - for
  - while
  - do-while
- Preprocessor conditionals
  - #ifdef, #endif
  - #if, #endif
  - #ifndef, #endif

Conditionals: `if`

- Syntax: **`if (`**<condition>**`)`** <statement>

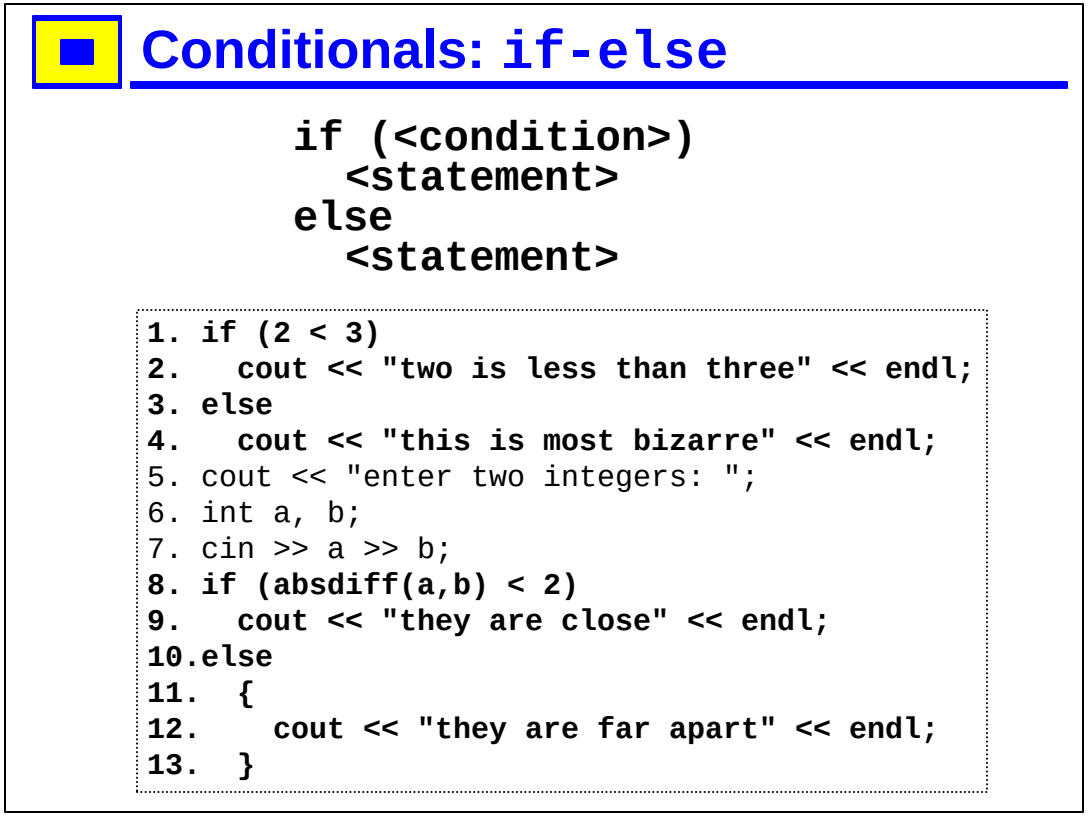
```
1. if (2 < 3)
2.   cout << "two is less than three" << endl;
3. cout << "enter two integers: ";
4. int a, b;
5. cin >> a >> b;
6. if (absdiff(a,b) < 2)
7.   cout << "they are close" << endl;
8. if (sin(a) >= 0 && cos(a) >= 0)
9. {
10.  cout << "both positive" << endl;
11.  a = -a;
12.  b = -b;
13.}
```

XXX flow diagram

1. The syntax for an if-statement is the keyword if, followed by a condition in parentheses, followed by a statement.
2. The condition can be any expression that evaluates to a boolean or to an integral type (int, enum, char).
3. Statement can be a simple statement, or a block (a sequence of statements enclosed in curly braces {}.
4. To try this:

```
#include <iostream>
#include <cmath>
#include "example.h"

int main()
{
  using namespace std;
  // the code on the slide
}
```



1. A block can contain a single statement; the meaning is identical to the statement alone.
2. Note that statements are terminated with a semi-colon.
3. Note also that in C and in C++, an expression followed by a semi-colon is a statement.
4. So the following is perfectly legal, though rather bizarre, C and C++:

```
1.int main()
2.{
3.  3 + 5;
4.  "huh";
5.  int eresting, dont, you, think;
6.  you;
7.}
```

Conditionals: cascading if-elses

```
if (<condition>)
  <statement>
[ else if (<condition>)
  <statement> ] *
[ else
   <statement> ] ?
```

```
1. cout << "enter 3 ints: ";
2. int a, b, c;
3. cin >> a >> b >> c;
4. if (a < b && b < c)
5.   ;
6. else if (a < b && b > c)
7.   std::swap(a,b);
8. else if (a > b)
9.   std::swap(b,c);
10.else
11.  cout << "eh?" << endl;
```

1. An if statement can have any number of else-if clauses, and may have an else at the end.
2. In grammars, a * means the preceding entity repeated 0 or more times; a ? means the preceding entry or nothing.
3. If the conditional is at all complicated, you should document what it does. For example, the conditional on lines 4-11 should definitely be commented; what does it do?
4. A statement can be empty; if a is less than b and b is less than c, nothing happens.
5. std::swap is in the standard library, in the header file <algorithms>.

Switch

- Syntax:

```
switch (<expression>)
{
[ case <constant>: <statement>* ]*
  default: <statement>*
}
```

- Often used for menus

1. The expression must be of some integral type; int or enum.
2. The constants must be compile-time constants, e.g., literals (96, '@'), or enum names, or static const integral members.
3. Examples follow.

Switch example (input o)

```
1.  cout << "o)ne a)nother t)hird q)uit";
2.  char ch;
3.  cin >> ch;
4.  switch (ch)
5.  {
6.  case 'o':
7.    cout << "one" << endl;
8.    break;
9.  case 'a':
10.   cout << "another" << endl;
11.   break;
12. case 't':
13.   cout << "third" << endl;
14.   break;
15. case 'q':
16.   break; // do nothing
17. default:
18.   cerr << "unexpected character: " << ch;
19. }
20. cout << "past the switch" << endl;
```

break

- `break` is a keyword that can be used inside a `switch` or a `for`, `while` or `do-while` loop
- In a `switch`, it means jump to the end of the switch
- In a loop it means jump to the end of the loop

```
1.  switch (ch)
2.  {
3.  case 'o':
4.    cout << "one" << endl;
5.    break;
6.  case 'a':
7.    cout << "another" << endl;
8.    break;
9.  }
10. cout << "past the switch" << endl;
```

1. We will see examples of break in loops shortly; in loops, you can also say continue, which means restart the loop.
2. If a switch is nested inside another switch, break means break out of the inner one. To jump up several control-flow levels you have to use labels and goto, which are strongly discouraged, though they remain in the language for backward-compatibility.
3. It is strongly suggested that you end each case with a break, or document carefully in comments in the source code why you did not.

Switch example (input a)

```
1.  cout << "o)ne a)nother t)hird q)uit";
2.  char ch;
3.  cin >> ch;
4.  switch (ch)
5.  {
6.  case 'o':
7.    cout << "one" << endl;
8.    break;
9.  case 'a':
10.   cout << "another" << endl;
11.   break;
12. case 't':
13.   cout << "third" << endl;
14.   break;
15. case 'q':
16.   break; // do nothing
17. default:
18.   cerr << "unexpected character: " << ch;
19. }
20. cout << "past the switch" << endl;
```

Switch example (input t)

```
1.  cout << "o)ne a)nother t)hird q)uit";
2.  char ch;
3.  cin >> ch;
4.  switch (ch)
5.  {
6.  case 'o':
7.    cout << "one" << endl;
8.    break;
9.  case 'a':
10.   cout << "another" << endl;
11.   break;
12. case 't':
13.   cout << "third" << endl;
14.   break;
15. case 'q':
16.   break; // do nothing
17. default:
18.   cerr << "unexpected character: " << ch;
19. }
20. cout << "past the switch" << endl;
```

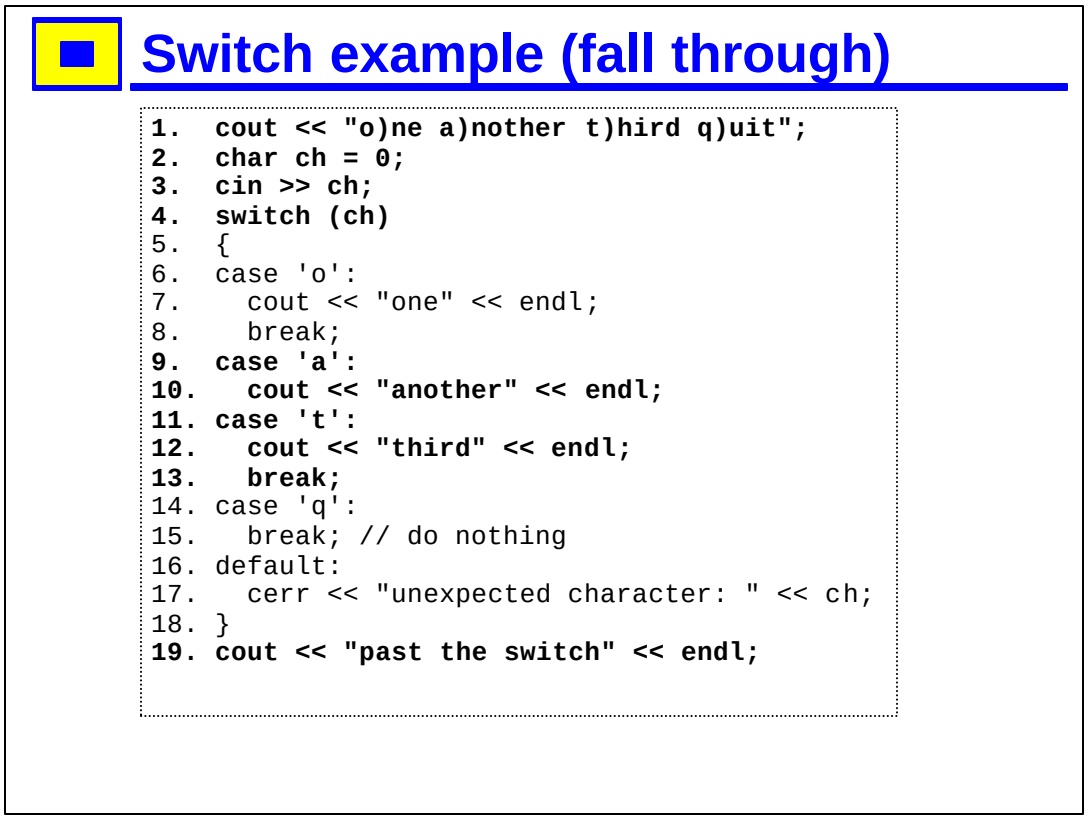
Switch example (input q)

```
1.  cout << "o)ne a)nother t)hird q)uit";
2.  char ch;
3.  cin >> ch;
4.  switch (ch)
5.  {
6.  case 'o':
7.    cout << "one" << endl;
8.    break;
9.  case 'a':
10.   cout << "another" << endl;
11.   break;
12. case 't':
13.   cout << "third" << endl;
14.   break;
15. case 'q':
16.   break; // do nothing
17. default:
18.   cerr << "unexpected character: " << ch;
19. }
20. cout << "past the switch" << endl;
```

Switch example (input x)

```
1.  cout << "o)ne a)nother t)hird q)uit";
2.  char ch = 0;
3.  cin >> ch;
4.  switch (ch)
5.  {
6.  case 'o':
7.    cout << "one" << endl;
8.    break;
9.  case 'a':
10.   cout << "another" << endl;
11.   break;
12. case 't':
13.   cout << "third" << endl;
14.   break;
15. case 'q':
16.   break; // do nothing
17. case 0:
18.   cerr << "unexpected eof" << endl;
19.   break;
20. default:
21.   cerr << "unexpected character: " << ch;
22. }
23. cout << "past the switch" << endl;
```

1. If the user types end of file, the cin >> ch will silently fail. The eof bit in cin will be turned on, but here we are not checking for it. The value of ch remains 0, so the case 0 would get called.
2. Note that case 0 is the same as case '\0', but different from case '0'. The first two are the null character, the other is the zero character, ascii value 48.



1. Here the input is 'a'.
2. The case 'a' is executed, but as there is no break, execution continues with the printing of "third". This makes it possible to reduce duplicate code in some cases, but be very careful when doing this.
3. The general rule should be that every case ends with a break; if you do not follow this, explain why in comments. One common example is something like this:

```
• cout << "yes or no? ";
• cin >> ch;
• switch (ch)
• {
• case 'y':
• case 'Y':
•   cout << "yes" << endl;
•   break;
```

For loops

- Syntax:

```
for (<init-stmt>; <cond>; <post-stmt>)
    <body-stmt>
```

- Most convenient for iterating through a range of integers
- Much more widely applicable
- Recommended for infinite loops
- Any of the statements can be empty

1. The first statement specifies what to do before the loop starts. The second component is a condition: an expression which is evaluated as a boolean. If it is false, the for terminates immediately. Otherwise, the body statement is executed, then the post-statement. Then the condition is checked again; as long as it is true, the body and post statements are executed.
2. This is equivalent to
   - <init-stmt>
   - while (<cond>)
   - {
   - <body-stmt>
   - <post-stmt>
   - }

For example

- Program to print the first 10 squares:

```
1. #include <iostream>

2. int main()
3. {
4.   using std::cout;
5.   using std::endl;
6.   for (int i = 0; i < 10; ++i)
7.     cout << i*i << endl;
8. }
```

1. In small examples like this, the namespace code seems clutterful, but in practice this is not too much of a problem; it is fine to be more lax about namespaces in implementation files, but it is particularly important that you be careful with them in header files.
2. Read this: for int i set equal to zero, as long as i is less than ten, incrementing i each time, print to standard output i times i and a newline.
3. Purists would object to using endl every time through this loop, as it flushes the output buffer unnecessarily.
4. I could just as well have written this program:

```
1. #include <iostream>
2. int main()
3. {
4.   for (int i = 0; i < 10; ++i)
5.     std::cout << i*i << std::endl;
6. }
```

Infinite loop example

- Copy stdin to stdout, line by line:

```
1. string line;
2.
3. for (;;)
4. {
5.   getline(cin, line);
6.   if (cin.eof())
7.     break;
8.   cout << line << '\n';
9. }
```

1. Read this: forever…
2. I have omitted the includes (iostream and string), and the namespaces, for clarity.

While loops

- Syntax:

`while` (<condition>)
    <statement>

```
1. string line;
2.
3. while (true)
4. {
5.   getline(cin, line);
6.   if (cin.eof())
7.     break;
8.   cout << line << '\n';
9. }
```

1. We could rewrite the previous example with a while loop.

Do-while loops

- Syntax:

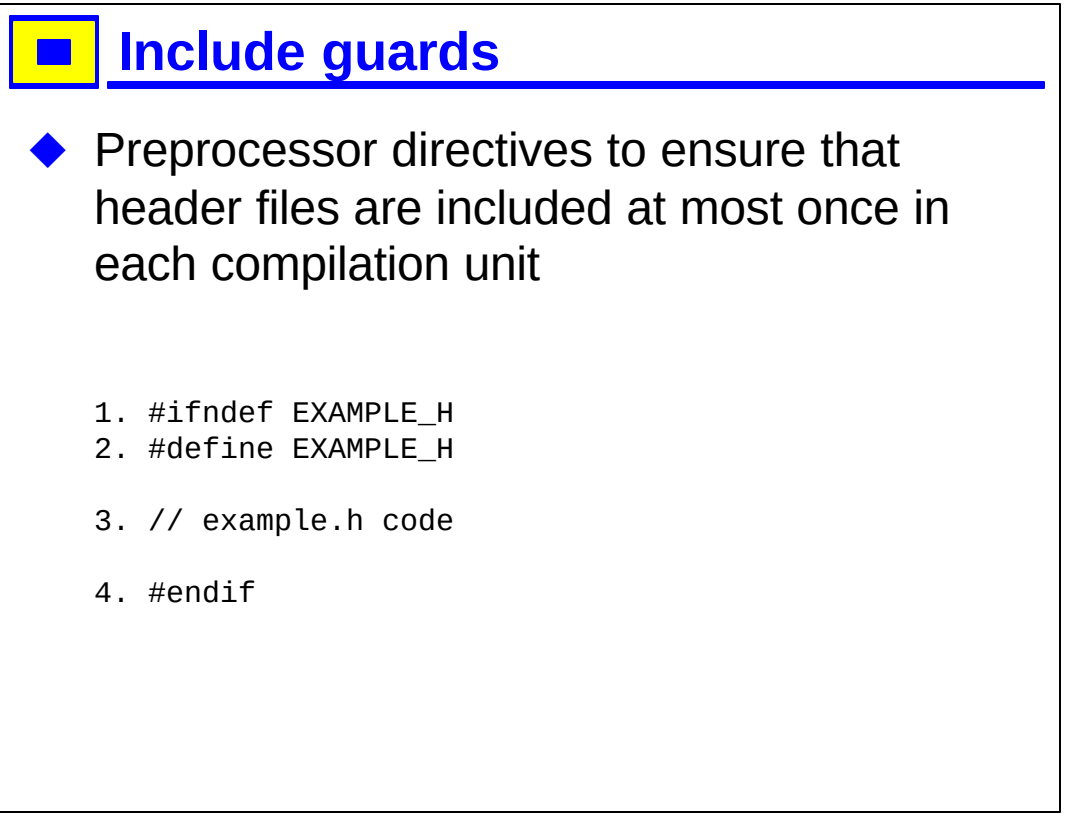
```
do
{
    <statement>*
}
while (<condition>);
```

```
1. string line;
2.
3. do
4. {
5.   getline(cin, line);
6.   if (!cin.eof())
7.     cout << line << '\n';
8. }
9. while (!cin.eof());
```

1. Here is the same example with a do-while loop.
2. This form of loop is not as common as the others, but it is sometimes the most natural way to express things.

Preprocessor flow control

- In a few cases, it is most convenient to use the preprocessor for flow control
  - Include guards
  - Platform-specific code
  - Include dependencies



1. The preprocessor keeps tracks of which symbols (names) are defined, and what their values are. The first line here says "if the symbol EXAMPLE_H is not defined", the second line says "define that symbol", and the last line terminates the condition.
2. The first time this file is included in a compilation unit, the symbol will not have been defined (if you have a reasonable scheme for choosing the symbol-name based on the file), so the entire file is read.
3. Subsequent times, if this file is included again, the symbol will be defined, so the entire file is skipped.