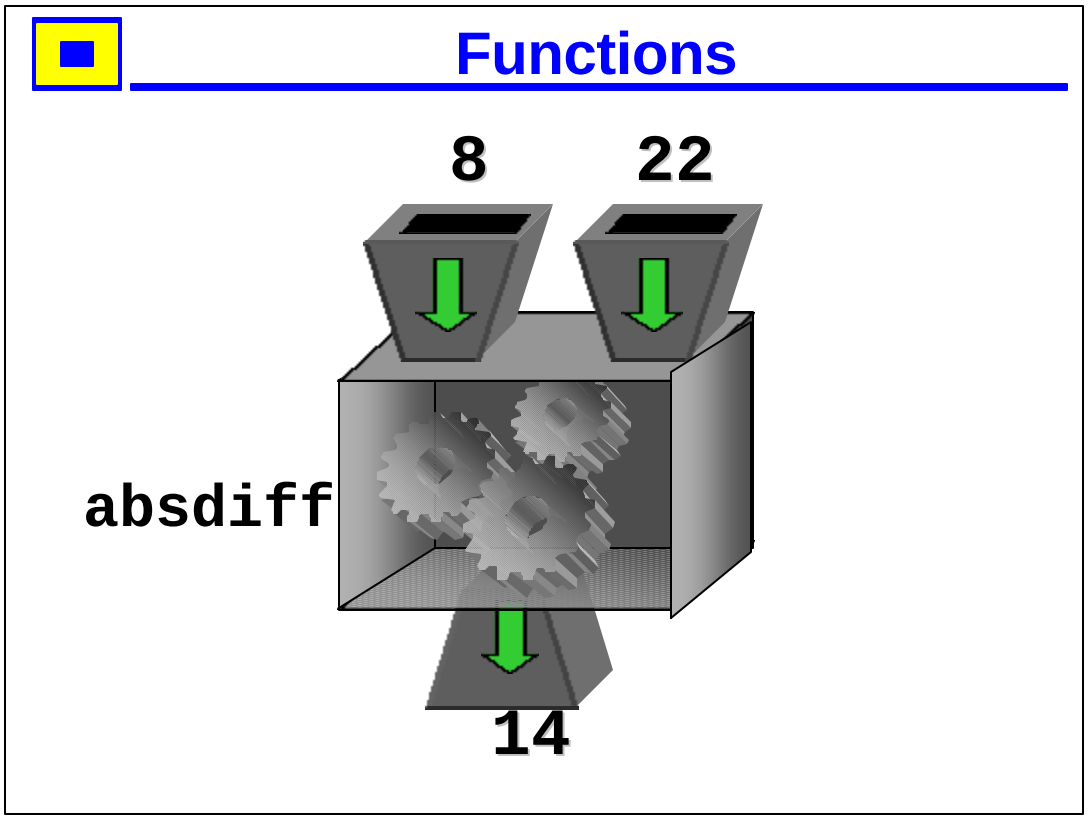


1. Perspective is messed up here.
2. Picture needs to be redone in truespace.
3. Think of a function as a black box (meaning you shouldn't worry about what is happening inside) that has some number of inputs, and possibly an output.
4. The inputs and outputs here are integers, but they could be of any type.
5. The picture corresponds to the function

```
•int absdiff(int a, int b)
•{
•  return (a < b) ? b-a : a-b;
•}
```

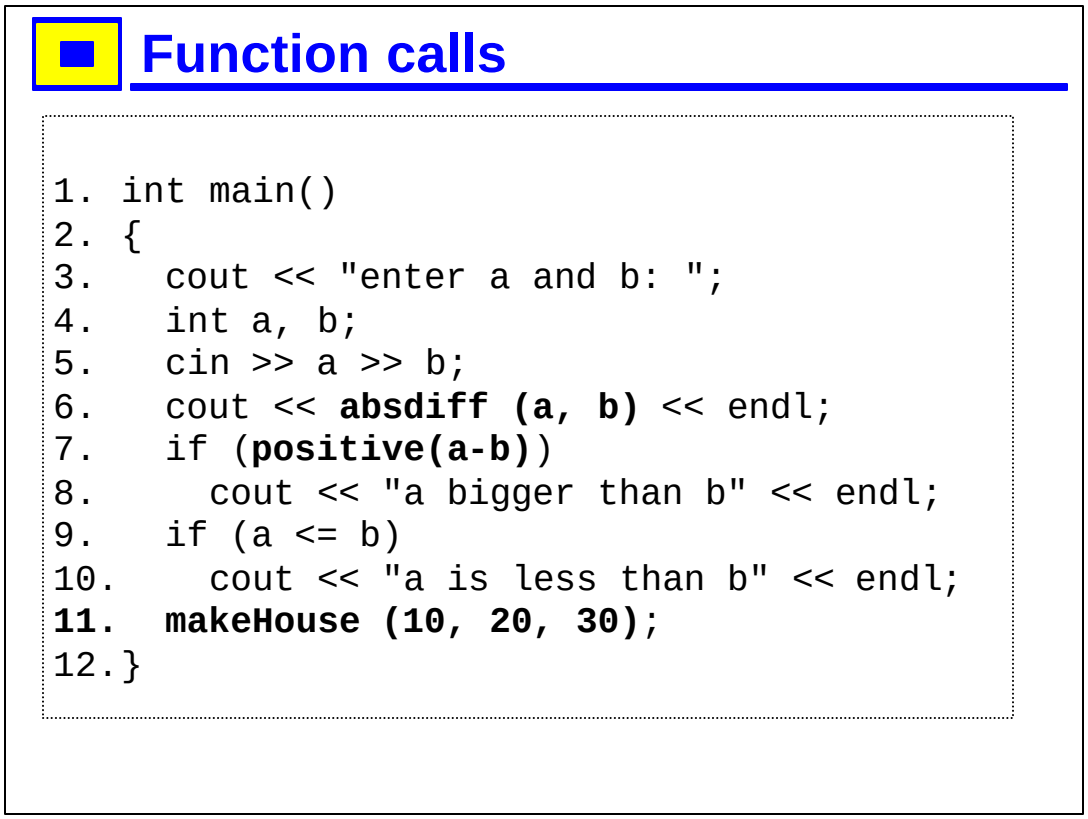
6. ?: is the C ternary operator – it is unusual in that it has three operands, instead of the usual one or two. The first operand is evaluated as a boolean. If it is true, the result of the operation is the second operand; otherwise the result is the third operand. It is sometimes called the if-then operator.

Function Definitions

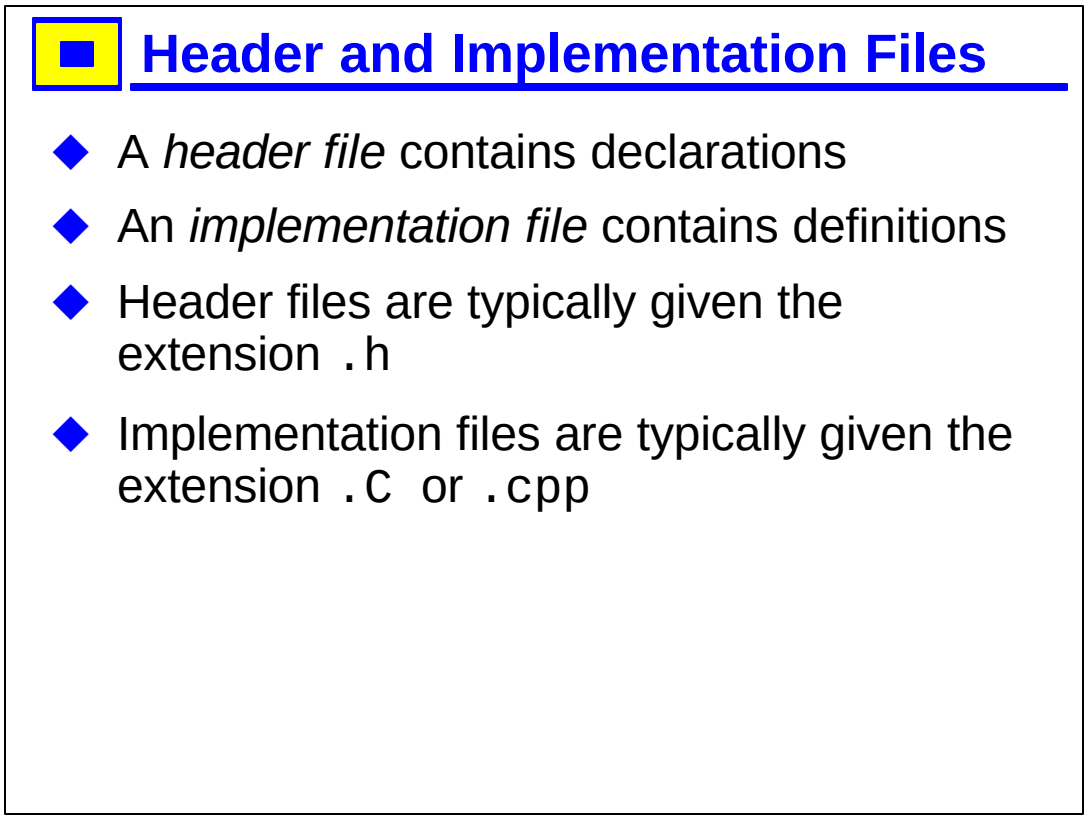
- A function has a *name*, a *signature*, and a *return type*; together these are the function's prototype:

```
1. int absdiff (int a, int b);

2. bool positive (double x);

3. void makeHouse (int width, // in pixels
4.                 int depth,
5.                 int height);
```

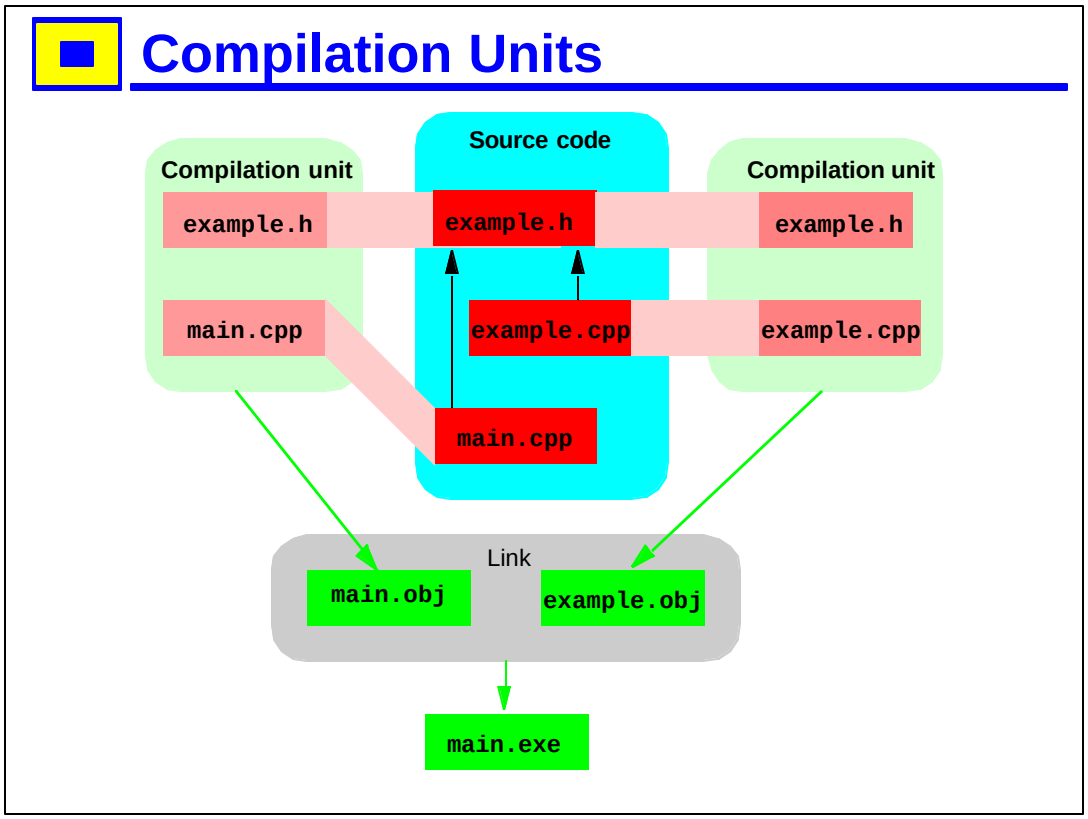
1. For anyone to use the function, they must declare it. This is usually done by having what is called a prototype of the function in a header file that the users are told to include.
2. The first function is called absdiff. It is a function taking two integers and returning an integer. You might infer from the name that it will return the absolute value of the difference of the two arguments; of course, the compiler doesn't know that.
3. You should get used to putting yourself in the compiler's shoes, so to speak. That is, try to keep in mind what it is that the compiler has seen so far when it gets to this point in your code. Typically, it will have included some number of header files.
4. Anything after a // to the end of the line is a comment.
5. Note that the prototype does not have to be on one line. The C++ compiler doesn't care about whitespace.
6. People reading the code **do** care about whitespace --- indent your code to make it easy to read.



1. This program prints out a prompt: enter a and b:, then waits for the user to type two integers separated by whitespace.
2. It puts the first into a, the second into b.
3. It calls the function absdiff, passing it the two arguments a and b, and prints the result to stdout (the screen) followed by a newline.
4. It then calls positive with the result of a-b, and if the returned boolean is true, it prints out a message.
5. Of course, you don't have to use a function for this; you could use a comparison operator, as shown; this is just for illustration.
6. Finally, the program calls the function makeHouse with three arguments, 10, 20 and 30.
7. To compile this code, you would need to include <iostream> for cout, cin and endl, and you would need to include a header file that contained the function prototypes for the three functions absdiff, positive and makeHouse.



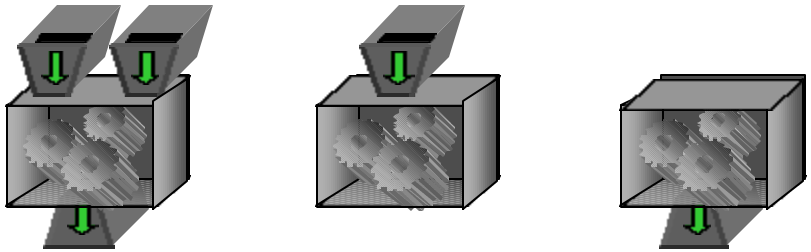
1. Declarations tell the compiler something, but don't generate any resulting code. Definitions correspond to actual code generated, or space allocated by the compiler.
2. .h and .C are just conventions; the language says nothing about the filenames of the source code.
3. On operating systems that have case-sensitive filenames, .C (dot capital C) is the most common extension; operating systems where filenames are case-insensitive (e.g., DOS), and operating systems that evolved from them (e.g., all the flavors of Windows) usually use .cpp. Some compilers preferred .cxx (the x looking vaguely like a plus sign if you tilt your head ☺)
4.



1. The source files are on the left. The project contains two implementation files, example.cpp and main.cpp. Each of these includes the file example.h. When example.cpp is compiled, it produces an object file called example.obj; similarly, when main.cpp is compiled, it produces an object file called main.obj. These object files are called relocatable, meaning that they specify code and data locations in relative terms, with no absolute addresses.
2. The linker is a separate program that takes object files, combines them and changes the relative addresses to absolute addresses. It creates an executable program.
3. The diagram here is somewhat simplified --- main.cpp includes some standard library headers, for example, and the linker links together these object files and also the C++ standard library, and possibly other libraries, depending on the platform (for example, it might also link in code from MFC or VCL or Xlib).

Function return types

- No difference in syntax between procedures and functions
- Functions return a single object
- You can return several objects by putting them in a structure



1. A function that returns no value declares its return type as void.
2. Some languages allow you to return several objects; C++ does not.
3. We will discuss structures in chapter XXX.

Function arguments

- Each argument to a function is specified by a type and a name
- Functions can take any number of arguments
- Arguments can have default values

```
1. void makeHouse(int w = 20, int h = 14, int d = 10);
2. void makeOther(int w = 20, int h, int d); // ERROR
```

```
1.   makeHouse ();        // makeHouse(20, 14, 10);
2.   makeHouse(15);       // makeHouse(15, 14, 10);
3.   makeHouse(10, 20);   // makeHouse(10, 20, 10);
4.   makeHouse(1, 2, 3); // makeHouse(1, 2, 3);
5.   makeHouse( , 2, 3); // ERROR
```

1. In a function declaration, the names of the variables are for documentation purposes only; they are ignored by the compiler. In particular, they do not have to match the names used in the definition of the function.
2. In a definition, if an argument is not currently used, compilers will usually warn you. To avoid the warning, remove the name of the variable (or comment it out with /* */).
3. Only trailing arguments can have defaults, and it is never legal to skip an argument and then continue to list arguments.
4. C++ does not have keyword arguments, which exist in some languages like Python, Perl and CLOS.

Call By Value

- There are different mechanisms for passing parameters to functions
  - Call-by-value
  - Call-by-reference
  - ??
- In C++, functions are by default call-by-value
- Each argument is copied
- Modifications to the functions' arguments do not affect the caller

Example

```
1.  #include <iostream>
2.  using std::cout;
3.  using std::endl;

4.  void printPowers (int n)
5.  {
6.    cout << "n: " << n << endl;
7.    n = n*n;
8.    cout << "squared: " << n << endl;
9.  }

10. int main()
11. {
12.   int a = 8;
13.   cout << "a is now " << a << endl;
14.   printPowers(a);
15.   cout << "a is still " << a << endl;
16. }
```

1. Modifications made to the parameter n inside the function printPowers do not have any effect on the variable a which was passed to the function: a was copied into n.
2. <picture of stack needed>

Inline Functions

- Normally, a call to a function causes execution to jump to a new part of the code
- Registers may ned to be saved and restored
- Function calls add overhead
- An inline function call is expanded without causing a function call
- C++ functions can be declared inline

Examples

```
1. inline int square(int n) { return n * n; }

2. inline char tolower(char ch)
3. {
4.   return isupper(ch) ? (ch – 'A' + 'a') : ch;
5. }

6. inline bool isupper(char ch)
7. {
8.   return ch >= 'A' && ch <= 'Z';
9. }

10.int main()
11.{
12.  cout << "enter a char";
13.  cin >> ch;
14.  if (isupper(ch))
15.    ch = tolower (ch);
16.}
```

1. The inline keyword is only a suggestion to the compiler; compilers can ignore the suggestion.
2. Inline functions must be defined in header files --- the compiler needs to know what the body of the function is in order to replace a call with the appropriate code.
3. An inline function call looks just like a non-inline call; and the semantics (the meaning) of the call is identical; the only thing that may change is how long it takes.
4. The disadvantage of inline functions is that it exposes the implementation of the function: if this changes, every file where the function is used must be recompiled.
5. Occasionally (though more and more rarely nowadays) the actual algorithm the function uses may be proprietary. Typically, in this case, the cost of a function call is negligible, so inlining the function doesn't make sense.
6. The vast majority of functions in the standard library are inline, so don't worry about overhead from calling standard library functions. In the newer parts of the library, e.g., the STL (standard template library), every function is inline, as they are all templates.