Chapter One: Introduction II

Slides by Evan Gallagher & Nikolay Kirov

Analyzing Your First Program

Analyzing Your First Program

- At this point we will analyze the classic first program that everyone writes: Hello World!
  - (yes, everyone who is anyone started with this one)
- Its job is to write the words Hello World!
  on the screen.

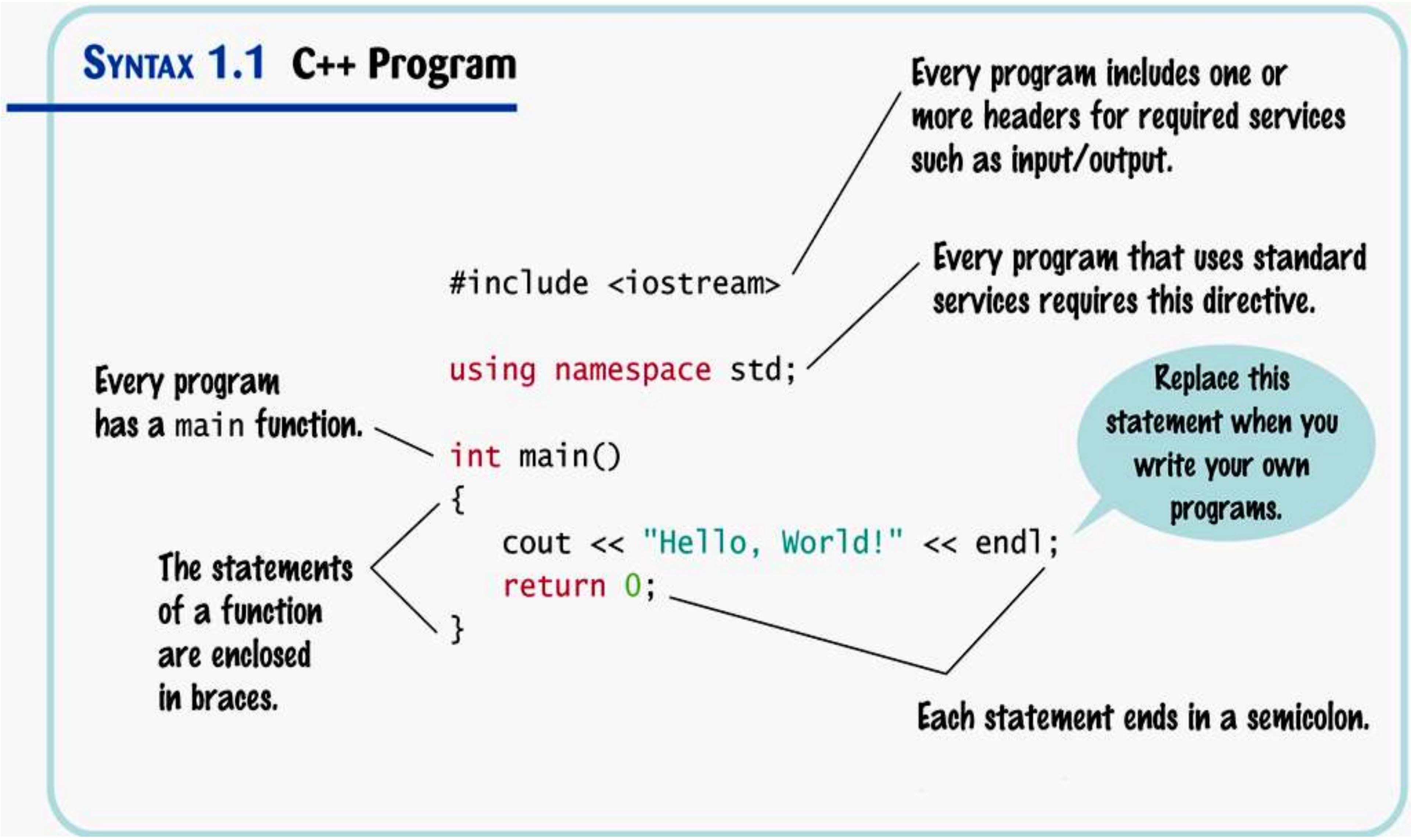
ch01/hello.cpp

```
#include <iostream>

using namespace std;

int main()
{
    cout << "Hello, World!" << endl;
    return 0;
}
```

Analyzing Your First Program



SYNTAX 1.1 C++ Program

```
#include <iostream>

using namespace std;

int main()
{
   cout << "Hello, World!" << endl;
   return 0;
}
```

Every program includes one or more headers for required services such as input/output.
Every program that uses standard services requires this directive.
Every program has a main function.
Replace this statement when you write your own programs.
The statements of a function are enclosed in braces.
Each statement ends in a semicolon.

Analyzing Your First Program

- The first line tells the compiler to include a service for “stream input/output”. Later you will learn more about this but, for now, just know it is needed to write on the screen.

ch01/hello.cpp

```
#include <iostream>

using namespace std;

int main()
{
   cout << "Hello, World!" << endl;
   return 0;
}
```

Analyzing Your First Program

- The second line tells the compiler to use the “standard namespace”. Again more later but for now, this is used in conjunction with the first line to output – and we certainly want “standard” output – nothing fancy yet.

ch01/hello.cpp

```
#include <iostream>

using namespace std;

int main()
{
   cout << "Hello, World!" << endl;
   return 0;
}
```

Analyzing Your First Program

- The next set of code *defines* a ***function***.
  The name of this function is **`main`**.

ch01/hello.cpp

```
#include <iostream>

using namespace std;

int main()
{
   cout << "Hello, World!" << endl;
   return 0;
}
```

Analyzing Your First Program

- The **`main`** function “returns” an “integer” (that is, a whole number without a fractional part, called **int** in C++) with value 0. This value indicates that the program finished successfully.

ch01/hello.cpp

```
#include <iostream>

using namespace std;

int main()
{
   cout << "Hello, World!" << endl;
   return 0;
}
```

Analyzing Your First Program

- To show output on the screen, we use an entity called **`cout`**.
- What you want seen on the screen is “sent” to the **`cout`** entity using the **<<** operator (sometimes called the insertion operator): **<< "Hello, World!"**

ch01/hello.cpp

```
#include <iostream>

using namespace std;

int main()
{
   cout << "Hello, World!" << endl;
   return 0;
}
```

Analyzing Your First Program

- You can display more than one thing by re-using the **<<** operator: **<< "Hello, World!" << endl;**

ch01/hello.cpp

```
#include <iostream>

using namespace std;

int main()
{
   cout << "Hello, World!" << endl;
   return 0;
}
```

Analyzing Your First Program

The output statement

```
cout << "Hello World!" << endl;
```

is an *output statement*.

Analyzing Your First Program

The output statement

```
cout << "Hello World!" << endl;
```

- To display values on the screen, you send them to an entity called **`cout`**.

Analyzing Your First Program

The output statement

```
cout << "Hello World!" << endl;
```

- To display values on the screen, you send them to an entity called **`cout`**.
- The **<<** operator denotes the “send to” command or “output stream” operator.

Analyzing Your First Program

```
cout << "Hello World!" << endl;
```

- **`"Hello World!"`** is called a *string*.
- You must put those double-quotes around strings.

- The **`endl`** symbol denotes an *end of line* marker which causes the cursor to move to the next screen row.

Analyzing Your First Program

- Each statement in C++ ends in a semicolon.

ch01/hello.cpp

```
#include <iostream>

using namespace std;

int main()
{
   cout << "Hello, World!" << endl;
   return 0;
}
```

Errors

ARGH!!!!

Common Error – Omitting Semicolons

Common error

Omitting a semicolon (or two)

Oh No!

```
#include <iostream>

using namespace std;

int main()
{
    cout << "Hello, World!" << endl
    return 0;
}
```

Errors

Without that semicolon you actually wrote:

```
cout << "Hello, World!" << endl return 0;
}
```

which thoroughly confuses the compiler!

This is a *compile-time error* or *syntax error*.

A syntax error is a part of a program that does not conform to the rules of the programming language.

Errors

Suppose you (accidentally of course) wrote:

```
count << "Hello World!" << endl;
```

- This will cause a compile-time error and the compiler will complain that it has no clue what you mean by “count”. The exact wording of the error message is dependent on the compiler, but it might be something like
  “Undefined symbol count”.

Errors – How Many Errors?

- The compiler will not stop compiling, and will most likely list lots and lots of errors that are caused by the first one it encountered.
- You should fix only those error messages that make sense to you, starting with the first one, and then recompile (after SAVING, of course!).

Errors

C++

- has *free-form layout*
-
  ```
  int main(){cout<<"Hello, World!"<<endl;return 0;}
  ```
-
- *will* work (but is practically impossible to read)

  A good program is readable.

Errors

Consider this:

```
cout << "Hollo, World!" << endl;
```

- *Logic errors* or *run-time errors* are errors in a program that compiles (the syntax is correct), but executes without performing the intended action.

not really an error?

Errors

```
cout << "Hollo, World!" << endl;
```

- No, the programmer is responsible for inspecting and testing the program to guard against logic errors.

really an error!

Errors

Some kinds of run-time errors are so severe that they generate an *exception*: a signal from the processor that aborts the program with an error message.

For example, if your program includes the statement

```
cout << 1 / 0;
```

your program may terminate with a “divide by zero” exception.

Errors

- Every C++ program must have one and only one **`main`** function.

- Most C++ programs contain other functions besides **`main`** (more about functions later).

Errors

C++

– is *case sensitive.* Typing:

```
int Main()
```

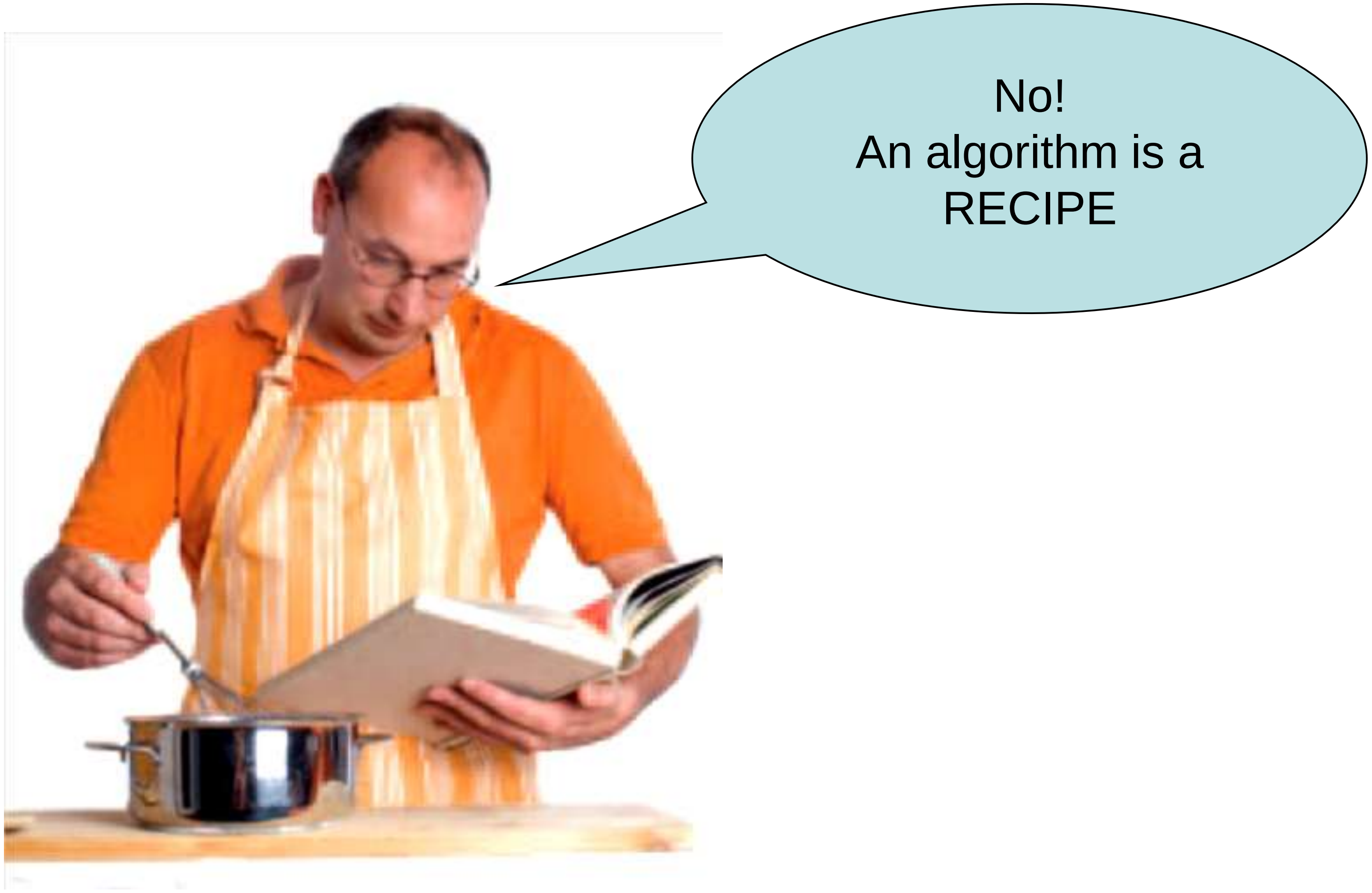
will compile but will not link.

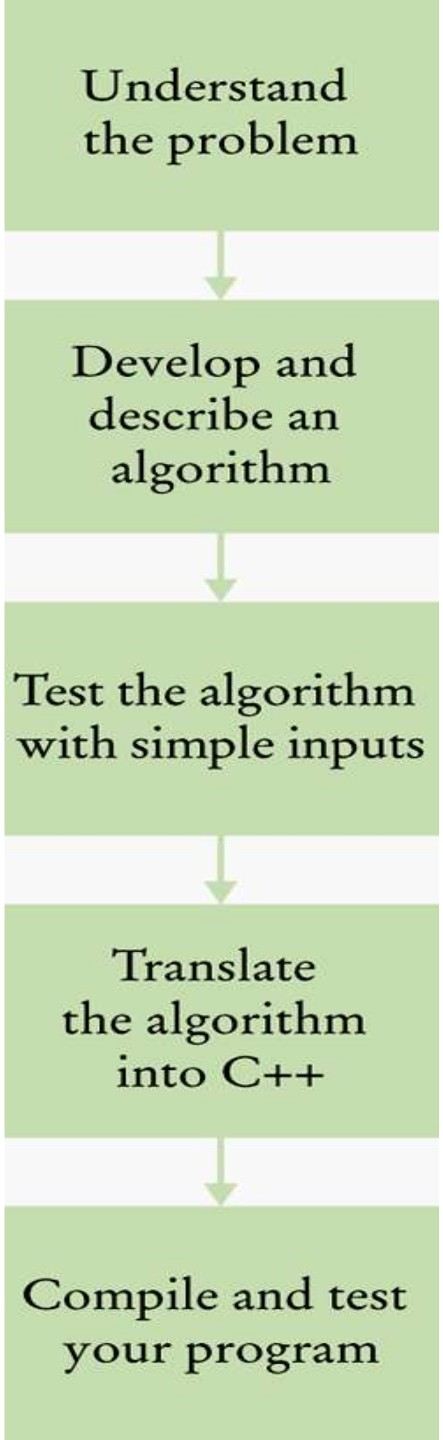
A link-time error occurs here when the linker cannot find the **`main`** function – because you did not define a function named **`main`**. (**`Main`** is fine as a name but it is not the same as **`main`** and there has to be one **`main`** somewhere.)

Algorithms

Not just any cook — an algorithmic cooker

Algorithms




Understand the problem
Develop and describe an algorithm
Test the algorithm with simple inputs
Translate the algorithm into C++
Compile and test your program

The Software Development Process

For each problem
the programmer goes
through these steps

Describing an Algorithm with Pseudocode

Pseudocode

- An informal description
- Not in a language that a computer can understand, but easily translated into a high-level language (like C++).

Describing an Algorithm with Pseudocode

The method described in pseudocode must be

- Unambiguous
  - There are precise instructions for what to do at each step
  - and where to go next.
- Executable
  - Each step can be carried out in practice.
- Terminating
  - It will eventually come to an end.

Describing an Algorithm with Pseudocode

Consider this problem:

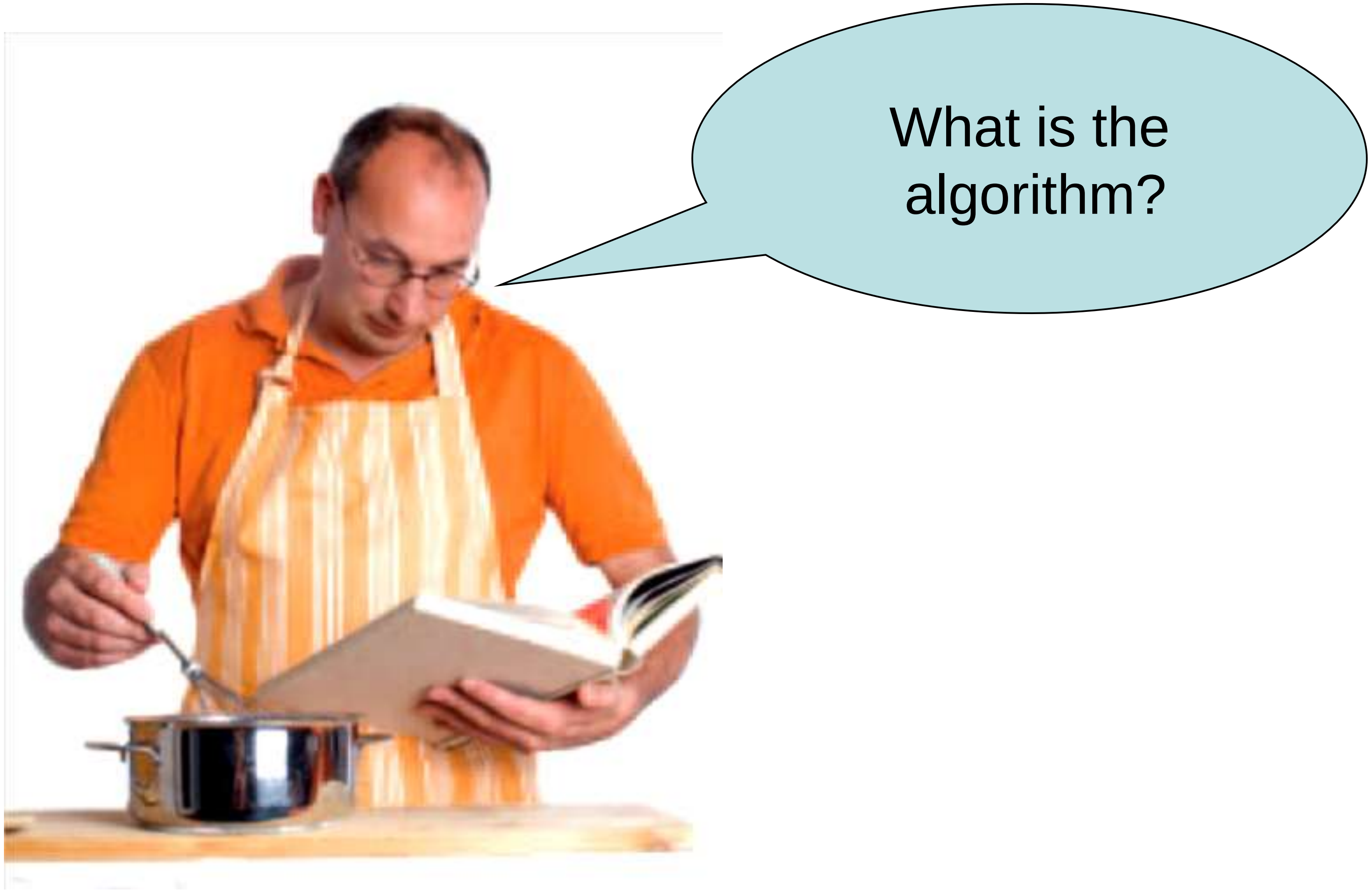
- You have the choice of buying two cars.
- One is more fuel efficient than the other, but also more expensive.
- You know the price and fuel efficiency (in miles per gallon, mpg) of both cars.
- You plan to keep the car for ten years.
- Assume a price of gas is $4 per gallon and usage of 15,000 miles per year.
- You will pay cash for the car and not worry about financing costs.

Which car is the better deal?

1 mile = 1.609344 kilometers

1 US gallon = 3.78541178 liters

Algorithms



Describing an Algorithm with Pseudocode

Step 1 Determine the inputs and outputs.

In our sample problem, we have these inputs:

- *purchase price1* and *fuel efficiency1*
  the price and fuel efficiency (in mpg) of the first car
- *purchase price2* and *fuel efficiency2*
  the price and fuel efficiency of the second car

We simply want to know which car is the better buy.
That is the desired output.

Describing an Algorithm with Pseudocode

Step 2 Break down the problem into smaller tasks.

What will we do for ***each*** car?

1. The total cost for a car is
   purchase price + operating cost
2. We assume a constant usage and gas price for ten years, so the operating cost depends on the cost of driving the car for one year.
   The operating cost is
   10 x annual fuel cost
3. The annual fuel cost is
   price per gallon x annual fuel consumed
4. The annual fuel consumed is
   annual miles driven / fuel efficiency

Describing an Algorithm with Pseudocode

Step 3 Describe each subtask in pseudocode.

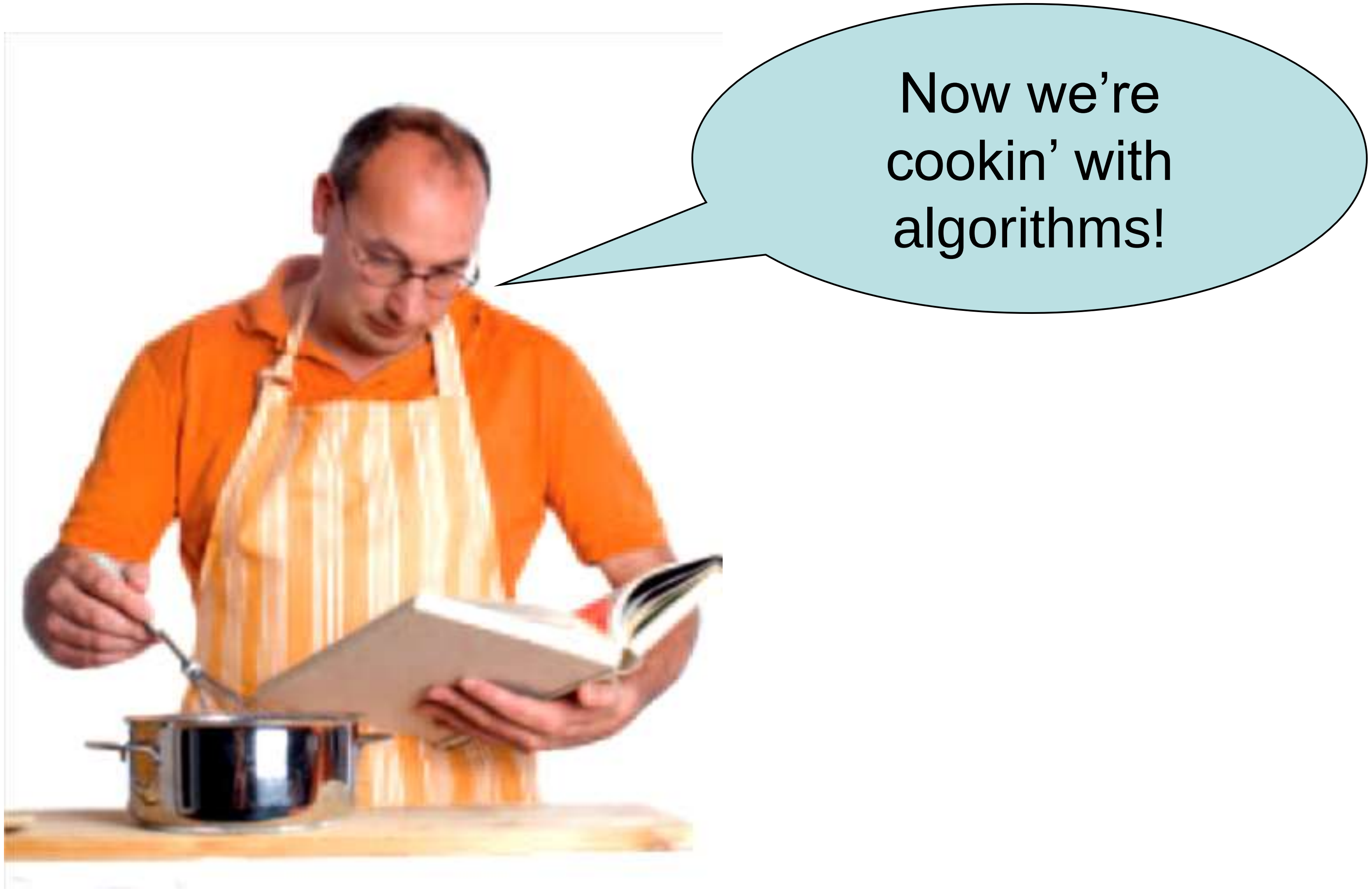
You will need to arrange the steps so that any intermediate values are computed before they are needed in other computations.

For each car, compute the total cost as follows:

```
    annual fuel consumed = annual miles driven / fuel efficiency
    annual fuel cost = price per gallon x annual fuel consumed
    operating cost = 10 x annual fuel cost
    total cost = purchase price + operating cost
```

```
If total cost1 < total cost2
    Choose car1
Else
    Choose car2
```

Algorithms



Describing an Algorithm with Pseudocode

Step 4 Test your pseudocode by working a problem.

Use these sample values:

- Car 1: $25,000, 50 miles/gallon
- Car 2: $20,000, 30 miles/gallon

FIRST CAR:

annual fuel consumed = 1500 / 50 = 300
annual fuel cost = 4 x 300 = 1200
operating cost = 10 x 1200 = 12000
total cost = 25000 + 12000 = 37000

SECOND CAR:

(let's assume you can do the math) total cost = 40000

If total cost1 < total cost2 …

The algorithm says: choose the FIRST CAR

Chapter Summary

Define "computer program" and programming.

- Computers execute very basic instructions in rapid succession.
- A computer program is a sequence of instructions and decisions.
- Programming is the act of designing and implementing computer programs.

Describe the components of a computer.

- The central processing unit (CPU) performs program control and data processing.
- Storage devices include memory and secondary storage.

Chapter Summary

Describe the process of translating high-level languages to machine code.

- Computer programs are stored as machine instructions in a code that depends on the processor type.
- C++ is a general-purpose language that is in widespread use for systems and embedded programming.
- High-level programming languages are independent of the processor.

Chapter Summary

Become familiar with your C++ programming environment.

- Set aside some time to become familiar with the programming environment that you will use for your class work.
- An editor is a program for entering and modifying text, such as a C++ program.
- C++ is case sensitive. You must be careful about distinguishing between upper and lowercase letters.
- Develop a strategy for keeping backup copies of your work before disaster strikes.

- The compiler translates C++ programs into machine code.
- The linker combines machine code with library code into an executable program.

Chapter Summary

Describe the building blocks of a simple program.

- Every C++ program contains a function called **main**.
- Use **cout** and the **<<** operator to display values on the screen.
- Enclose text strings in quotation marks.
- Use **+** to add two numbers and ***** to multiply two numbers.
- Send **endl** to **cout** to end a line of displayed output.
- End each statement with a semicolon.

Write pseudocode for simple algorithms.

- Pseudocode is an informal description of a sequence of steps for solving a problem.
- An algorithm for solving a problem is a sequence of steps that is unambiguous, executable, and terminating.

Chapter Summary

Classify program errors as compile-time and run-time errors.

- A compile-time error is a violation of the programming language rules that is detected by the compiler.
- A run-time error causes a program to take an action that the programmer did not intend.
- The programmer is responsible for inspecting and testing the program to guard against run-time errors.

End Introduction II