A board was cut in two.

The first part was 2/3 of the length of the original board.

The second part was four feet longer than the first.

How long was the board?

This is a math word problem. There are (at least) two answers here.

What are they?

---

**Problem**
- Definition of task to be performed (often by a computer)

**Algorithm**
- A particular sequence of steps that will solve a problem
- Steps must be precise and mechanical
- The notion of an algorithm is a (the?) fundamental intellectual concept associated with computing

**Program**
- An algorithm expressed in a specific computer programming language (C, C++, Java, Perl, ...)

Remember Turing? He helped define what an algorithm is.
Problem Solving
Clearly specify the problem
Analyze the problem
Design an algorithm to solve the problem
Implement the algorithm (write the program)
Test and verify the completed program

A Sample Problem
Is a given number even or odd?

Analysis
What numbers are allowed?
Where does the number come from?
What do “even” and “odd” mean?
How is the answer to be reported?

More Precise Problem Restatement
Given an integer number typed in from the keyboard,
If it is even, write “even” on the screen
If it is odd, write “odd” on the screen

An Algorithm
Read in the number
Divide the number by 2
If the remainder is 0, write “even”
Otherwise, write “odd”
Test: 234784632792543
An alternate algorithm:
If the rightmost digit is 0, 2, 4, 6, or 8, write “even”
Otherwise, write “odd”

Next, a C Program
Now that we have an algorithm, we would like to write a C program to carry it out.
But first, what is a program? In fact, what is a computer?
Review: What is a computer?

- CPU or processor: executes simple instructions manipulating values in memory
- Main Memory
- Monitor
- Network
- Disk (Files)
- Keyboard mouse

What is a Program?

The CPU executes instructions one after the other.

Such a sequence of instructions is called a “program” (also “software” or “code”)

Without a program, the computer is just useless hardware

Complex programs may contain millions of instructions

Lots of terminology here. Any guesses what the “wetware” might be?

Memory

Memory is a collection of locations

Within a program, the locations are called variables

Each variable has:
- A name (an identifier)
- A type (the kind of information it can contain)

Basic types include:

- int (integers – whole numbers: 17, -42)
- double (floating-point numbers with optional fraction and/or exponent: 3.14159, 6.02e23)
- char (character data: ‘a’, ‘?', ‘N’, ‘ ‘, ‘9’)

Program Sketch

Ask the user to enter a number
Read the number and call it num

Divide num by 2 and call the remainder rem

If rem is 0 write “even” otherwise write “odd”

The actual program has LOTS of details – IGNORE THEM FOR NOW
Pay attention to the main ideas

The Program in C (part I)

/* read a number and report whether it is even or odd */
#include <stdio.h>

int main (void) {
    int num;     /* input number */
    int rem;      /* remainder after division by 2 */

    /* get number from user */
    printf("Please enter a number: ");
    scanf("%d", &num);

    /* calculate remainder and report even or odd */
    rem = num % 2;
    if (rem == 0) {
        printf("even\n");
    } else {
        printf("odd\n");
    }
    /* terminate program */
    return 0;
}

Remember: Don’t sweat the details!!! (for now)
Sample Execution

```
#include <stdio.h>
int main (void) {
    int num; /* input number */
    int rem; /* remainder */
    /* get number from user */
    printf("Please enter a number: ");
    scanf("%d", &num);
    /* calculate remainder */
    rem = num % 2;
    if(rem == 0) {
        printf("even
");
    } else {
        printf("odd
");
    }
    /* terminate program */
    return 0;
}
```

A Quick Look at the Program

Text surrounded by /* and */ are comments. Used to help the reader understand the program. Ignored during program execution.

Programs change over time. It’s important that programmers be able to understand old code - good comments are essential.

Variables

Variable declarations create new variables and specify their names and types.

```
/* read a number ... */
#include <stdio.h>
int main (void) {
    int num; /* input number */
    int rem; /* remainder */
    /* get number from user */
    printf("Please enter a number: ");
    scanf("%d", &num);
    /* calculate remainder */
    rem = num % 2;
    if(rem == 0) {
        printf("even
");
    } else {
        printf("odd
");
    }
    /* terminate program */
    return 0;
}
```

Statements

Following the declarations are statements that specify the operations the program is to carry out. Lots of different kinds. Some (scanf, printf) are part of the C language proper. Others (scanf, printf) are contained in libraries of routines that are available for use in our programs. For now, don’t worry too much about the distinction.

```
/* read a number ... */
#include <stdio.h>
int main (void) {
    int num; /* input number */
    int rem; /* remainder */
    /* get number from user */
    printf("Please enter a number: ");
    scanf("%d", &num);
    /* calculate remainder */
    rem = num % 2;
    if(rem == 0) {
        printf("even
");
    } else {
        printf("odd
");
    }
    /* terminate program */
    return 0;
}
```

Functions

Functions are sequences of statements defined elsewhere. Some functions (such as printf and scanf here) are provided with the system. We will also learn how to write and use our own functions.

```
/* read a number ... */
#include <stdio.h>
int main (void) {
    int main (void) {
        int num; /* input number */
        int rem; /* remainder */
        /* get number from user */
        printf("Please enter a number: ");
        scanf("%d", &num);
        /* calculate remainder */
        rem = num % 2;
        if(rem == 0) {
            printf("even
");
        } else {
            printf("odd
");
        }
        /* terminate program */
        return 0;
    }
```
From C to Machine Language

The computer’s processor only understands "executable" programs written in its own machine language.
- Sequences of 1's and 0's
- Different for each processor family (x86, PowerPC, SPARC, ARM, ...)

How can the CPU obey instructions written in C?

Compilers and Linkers

There are two steps in creating an executable program starting from C source code:
- A program called the compiler translates the C code into an equivalent program in the processor’s machine language (1's and 0's)
- A program called the linker combines this translated program with any library files it references (printf, scanf, etc.) to produce an executable machine language program (.exe file)

Environments like Visual Studio do both steps when you “build” the program.

Compilers, Linkers, etc.

What Could Possibly Go Wrong?

Lots!
- Things are rarely perfect on the first attempt
- Both the compiler and linker could detect errors
- Even if no errors are detected, logic errors (“bugs”) could be lurking in the code
- Getting the bugs out is a challenge even for professional software developers

Terms: Syntax vs Semantics

Syntax: the required form of the program: punctuation, keywords (int, if, return, ...), word order, etc.
- The C compiler always catches these “syntax errors” or “compiler errors”

Semantics (logic): what the program means: what you want it to do
- The C compiler cannot catch these kinds of errors!
- They can be extremely difficult to find
- They may not show up right away

Try It Yourself!

Type in the even/odd program
First get it working. Then see what happens when you:
- Leave off a few semicolons or misspell something (syntax)
- In the last printf statements, change “odd” to “even”. Run the program. What happens if you enter 17? (semantics)

Experiment and see what happens
Wow!!

We’ve covered a lot of new ideas
- Algorithms and programs
- Computer organization and memory
- The basic components of C programs
- Comments, declarations, statements
- Compilers, linkers, libraries, and program execution
- Errors

Lots of terminology, too

What’s Next?

Upcoming lectures: review what we’ve seen today and fill in details

Meanwhile, get started reading and trying things on the computer!

QOTD: Playing with Syntax and Semantics

Imagine a game or a sport that you like to play. Or, use musical scores. Now, in that domain (game, sport, or music):
- Describe in words what the difference between a syntax error and a semantic error would be.
- Give an example of a semantic error.

Can you think of a really great example? Post it to the newsgroup!