So you see, control is no tyrant at all. It only does what it's told—completely automatically!

If you really want to imagine the control section's personality, think of a perfectly efficient bureaucrat, acting in strict obedience to the computer's real boss: the program!

"Go to... go to..."
—Shakespeare
If programs really rule the computer, they deserve a proper scientific name... something in Greek or Latin, preferably...

Technicalulus? Reulla Rationocerulus? Cerebalonuralgia?

* * * * * * *

But that's not how it is in computer science... Instead, programs in general are called software, to distinguish them from the circuit boards, cathode ray monitors, disk drives, keyboards, and other items of computer hardware.
WHAT'S REALLY FUNNY ABOUT THE NAME IS THAT SOFTWARE IS ONE OF THE HARDEST THINGS ABOUT COMPUTING!

WHILE HARDWARE HAS BEEN DROPPING IN PRICE AND GROWING IN POWER, SOFTWARE ONLY GETS MORE HORRENDOUSLY COMPLEX!

IT'S OFTEN IMPOSSIBLE TO ESTIMATE HOW MUCH TIME, MONEY, AND AGONY A GIVEN SOFTWARE PROBLEM WILL COST TO SOLVE... WHAT A WAY TO RUN A BUSINESS!

THE THRILL OF COMPUTING!

LIKEWISE THERE'S A DIFFERENCE BETWEEN THE IMAGE OF HARDWARE AND SOFTWARE WORKERS...

HARDWARE TYPES ARE ENGINEERS... INTO GADGETS... MOSTLY MEN... BOUND BY THE LAWS OF PHYSICS...

PROGRAMMERS HAVE NO TOOL BUT THEIR BRAINS... THEY'RE MORE OFTEN WOMEN... SUPPOSED TO BE SOLITARY DREAMERS WHOSE IDEAS HAVE NOTHING TO DO WITH THE LAWS OF PHYSICS!!

PROGRAMS THESE DAYS ARE SO COMPLEX THAT NO ONE PERSON CAN UNDERSTAND THEM — SO THESE LOVERS HAVE TO WORK IN TEAMS — A SPECTACLE I LEAVE TO THE READER'S IMAGINATION...
While Ada Lovelace was the original programmer, the first person to prove the full power of software was Alan Turing (1912-1954).

Turing, who enjoyed long-distance running back when that was considered weird, probably went into computers to shrink the size of his jogging clock.

It's making my stride loloped!

Roughly speaking, a Turing machine is an input-output device: a black box that reads a sequence of 0s and 1s.

The output depends only on the present input (0 or 1) and the previous output.

The nature of the output is unimportant.

The main thing is that the changes from one output state to the next are given by definite rules, called the transition rules.

The reason Turing machines are important is that they are a way of thinking physically about logic. Any well-defined, step-by-step logical procedure can be embodied in some Turing machine.

* For details, see J. Weizenbaum's *Computer Intel and Human Reason*, Chapter 2.
WHAT TURING PROVED: IT'S THEORETICALLY POSSIBLE TO CONSTRUCT A SINGLE TURING MACHINE, THE UNIVERSAL TURING MACHINE, WHICH CAN IMITATE ALL OTHER TURING MACHINES!!!

THE TRICK IS THAT THE UNIVERSAL TURING MACHINE CAN...

"read instructions!"

THAT IS TO MAKE THE UNIVERSAL TURING MACHINE (U) ACT LIKE MACHINE T, YOU ENCODE T'S TRANSITION RULES ONTO U'S TAPE. AT EACH STEP, U OBSERVES ITS OWN INPUT, THEN REFERS TO T'S TRANSITION RULES TO SEE WHAT TO DO.

\[ \Rightarrow \text{ IN OTHER WORDS, U IS PROGRAMMABLE!} \]

THE IMPLICATIONS ARE STAGGERING: A SINGLE PROGRAMMABLE MACHINE CAN PERFORM ANY WELL-DEFINED, STEP-BY-STEP LOGICAL PROCEDURE. (REMEMBER, TURING SAW THIS TEN YEARS BEFORE A REAL COMPUTER WAS BUILT.)

JOHN VON NEUMANN CARRIED TURING'S IDEAS A STEP FURTHER. VON NEUMANN REALIZED THAT ONE COULD:

BUILD A MACHINE X WHICH BUildS OTHER MACHINES FROM PLANS ENCODED ON TAPE...

FEED X THE PLANS TO ITSELF!

\[ \Rightarrow \text{ SELF-REPRODUCING MACHINES ARE POSSIBLE!} \]
The digital computer is a fancy universal Turing machine come to life.

If you call that "life..."

Therefore, as Turing proved, it can do anything (or, more accurately, simulate anything). The only limit is the amount of time at the user's disposal... say, from now until the death of the solar system...

Is that program finished running yet?

To be perfectly honest, there are a couple of other qualifications on that "anything." What kind of "anything" can a computer do?

Can it think?

Can it whir?

In a word, computers do

Algorithms

From Al-Khwarizmi, remember?

An algorithm is simply any well-defined, step-by-step procedure: a recipe, if you will!

Step-by-step, meaning each step is completed before the next is begun.

Well defined, meaning each step is completely determined by current input and the results of previous steps. No ambiguity allowed!
EXAMPLES OF ALGORITHMS:

"If nuclear warheads are falling like hailstones, I will lie down and try to enjoy it. Otherwise, I will go to work as usual."

It's an algorithm because I always know what to do:
1. Check to see if warheads are falling.
2. If yes, lie down and enjoy!
3. If no, go to work.

Likewise, algebraic formulas represent algorithms:

\[ y = x^2 + 2x + 10 \]

If you understand, lie down and enjoy yourself!

EXAMPLES OF NON-ALGORITHMS:

"If nuclear warheads are falling like hailstones, lie down and try to enjoy it."

This fails to tell you what to do if no warheads are falling... so it's not well defined.

Another?

How about

\[ y = x^2 + 2x - 10 \]

This is no algorithm because it's not expressed in proper "algebraic grammar." We assign no meaning to the symbols "++".

If you try to make a computer do a non-algorithm, it will just sit there flashing error messages!
Some standard symbols are used to make algorithms easier to follow. Each step is represented by a specially shaped box. The shape indicates what type of step is to be executed.

There are 4 possibilities:

- Begin or end
- Perform a procedure (add, subtract, etc.)
- Conditional branch
- Input or output

The "flow" of the algorithm is represented by arrows, and when all the symbols are combined, it's a flow chart.

Here are the flow charts of the algorithms from a couple of pages back:

In both algorithms, the flow proceeds in one direction, from start to finish.

It's also possible for the flow of algorithms to jump forward or backward. For example, let's rewrite that first algorithm:

1. If bombs are falling, go to step 2. Otherwise, go to step 4.
2. Lie down and enjoy!
4. Lead a normal life for 24 hours.
5. Go to step 1
6. END

You may find the flow chart easier to grasp than the written program. Note that it may continue indefinitely!
Flow charts are useful in helping to design algorithms—simple ones, anyway—and designing algorithms is what computer programming is all about!

The first step in writing any program is to analyze the job to be done, and see how to do it algorithmically!

Failure to think algorithmically has caused many software nightmares! Most software designers have horror stories about customers who didn’t know exactly what they wanted!

Feed me algorithms!

Let's try a couple more examples... a little more like what a computer might actually be asked to do...

"Roommate Receipts" Two roommates, Lisa and Sophie, share their meals. They both shop for food and save their receipts. At the end of the month, they want to know who owes whom how much.

"Multiple Plug-ins" This one asks the computer to evaluate the expression $x + 2x + 10$ not just at one value of $x$, but for many values, namely $x = 0, 0.1, 0.2, 0.3, \ldots$ and so on... up to 2.0.

Yes... that information should go into my files... or maybe not... the vice presidents are just as bad... or maybe the treasurers...

No! No!
FOR ROOMMATE RECEITS
WE REASON LIKE SO:

LET S = SOPHIE'S EXPENSES
L = LISA'S EXPENSES

THEN THE TOTAL EXPENSE IS S + L, AND EACH ROOMMATES SHARE IS \( \frac{1}{2}(S + L) \).

IF LISA OUTSPENT SOPHIE, SO L > S*, THEN SOPHIE OWES LISA
\( \frac{1}{2}(S + L) - S \), OR
\( \frac{1}{2}(L - S) \).

OTHERWISE (WHEN S ≥ L*), LISA OWES SOPHIE
\( \frac{1}{2}(S - L) \).

THE ALGORITHM'S OUTPUT IS TO TELL US WHO OWES WHOM AND HOW MUCH.

BEGIN

INPUT \( L, S \)

\( L > S? \)

NO

YES

\( \frac{1}{2}(L - S) \)

\( \frac{1}{2}(S - L) \)

OUTPUT "SOPHIE OWES LISA" AND \( \frac{1}{2}(L - S) \)

OUTPUT "LISA OWES SOPHIE" AND \( \frac{1}{2}(S - L) \)

END

BEGIN

LET \( X = 0 \)

EVALUATE \( x^2 + 2x + 10 \)

OUTPUT \( X \)

ADD 0.1 TO \( X \)

\( X < 2.0? \)

YES

NO

END

IN "MULTIPLE PLUG-INS," WE WANT TO EVALUATE A SINGLE EXPRESSION, \( x^2 + 2x + 10 \), REPEATEDLY AT DIFFERENT VALUES OF \( x \) (NAMELY 0.0, 0.1, 0.2, ..., 1.9, 2.0)

THE CORE OF THE ALGORITHM WILL BE THIS LOOP:

1. PLUG THE CURRENT VALUE OF \( x \) INTO \( x^2 + 2x + 10 \)
2. PRINT THE RESULT
3. NEXT \( X \)
4. RETURN TO STEP 1.

WE ALSO HAVE TO SPECIFY WHAT \( X \) TO START WITH, WHEN TO STOP, AND HOW TO COMPUTE "NEXT \( X \)."

NOTE HOW THE FLOW CHART SHOWS HOW THE PROGRAM LOOPS BACK, PLUGGING IN SUCCESSIVE VALUES OF \( X \) UNTIL \( X \) EXCEEDS 2.

* > MEANS "IS GREATER THAN"; ≥ MEANS "IS GREATER THAN OR EQUAL TO";
< MEANS "IS LESS THAN"; ≤ MEANS "IS LESS THAN OR EQUAL TO".
NOW THE $758 QUESTION: ($64 AFTER INFLATION):

**How do you write an algorithm that's intelligible to a computer?**

**Huh?**

IN OTHER WORDS, HOW DO YOU PROGRAM A COMPUTER?

UNFORTUNATELY, YOU HAVE TO SPEAK THE COMPUTER'S LANGUAGE—BECAUSE THE COMPUTER IS STILL TOO STUPID TO UNDERSTAND YOURS!

WHAT LANGUAGE DOES THE COMPUTER UNDERSTAND?

IT MAY BE FAST, BUT IT'S THICK!

AT THE VERY BEGINNING, PROGRAMMERS WROTE DIRECTLY IN "MACHINE LANGUAGE"—BINARY CODE. THIS WAS OBVIOUSLY A HEADACHE!

WE'LL NEED A COMPUTER JUST TO FIGURE OUR ASPIRIN BILL!

SOON THEY SWITCHED TO ASSEMBLY LANGUAGE (SEE P. 174), ADDED BY AUTOMATIC "ASSEMBLERS," WHICH TRANSLATED ASSEMBLY LANGUAGE MACHINOMICS INTO MACHINE CODE. STILL SOMETHING MORE WAS NEEDED!

The first higher-level language was FORTRAN ("Formula Translator"), which made its debut in the early 1950s. Since then, literally hundreds of languages have been written, each with its own army of rabid devotees!

We're going to take a quick look at BASIC—Beginner's All-Purpose Symbolic Instruction Code. BASIC is easy to learn and widely used, despite criticism (especially by Pascal admirers) that it promotes "bad programming habits."

It can't handle subroutines, with local arguments, grrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrr! And "G0 TO" is an idea alien to the human psyche!

With apologies to Pascal, then, here's a little BASIC...

There are two ways to write a BASIC program: with pencil and paper, or directly at the computer.

It's good practice to plan programs on paper first, to work out the essential ideas and structure, but eventually you must sit down at that keyboard!

Please... be gentle...

Some machines are ready for BASIC as soon as you turn them on. Others only bring it up on command. If in doubt, ask!
WHEN THE COMPUTER IS READY IT GIVES YOU A "PROMPT" OF SOME KIND: THE WORD "READY" OR JUST THE SIGN ">".

LETS GET GOING!

THE COMPUTER KEYBOARD RESEMBLES A STANDARD TYPEWRITER'S "QWERTY" KEYBOARD... EXCEPT THAT AS YOU TYPE, CHARACTERS APPEAR ON THE CRT (CATHODE RAY TUBE) SCREEN, INSTEAD OF ON PAPER. TO GO TO THE NEXT LINE, HIT THE RETURN (\(\uparrow\)) KEY.

HERE'S A SIMPLE BASIC PROGRAM:

```
10 REM BASIC MULTIPLICATION
20 READ A, B
30 DATA 5.6, 1.1
40 LET C = A * B
50 PRINT "THE PRODUCT IS" ; C
60 END
```

THE PROGRAM IS NOW STORED IN MEMORY. TO RUN IT, TYPE "RUN", FOLLOWED BY THE RETURN KEY. THE SCREEN DISPLAYS:

```
RUN
THE PRODUCT IS 6.16
```

> EVERY LINE BEGINS WITH A LINE NUMBER (10, 20, ...). EVERY LINE OF A BASIC PROGRAM MUST HAVE A NUMBER! IT'S Wise TO COUNT BY TENS, SO YOU CAN INSERT LINES LATER.

> THE FIRST LINE (10) IS A REMARK. REMARKS EXPLAIN THE PROGRAM BUT AREN'T EXECUTED BY THE COMPUTER. THE PREFIX "REM" IDENTIFIES REMARKS. WE MIGHT INSERT ONE HERE:

```
20 REM THE PROGRAM'S SIMPLE:
```

> PROGRAM STATEMENTS CONSIST OF INSTRUCTIONS ("LET", "IF"), NUMBERS (5.6, 1.1), VARIABLES (A, B, C), TEXT ("THE PRODUCT IS"), AND PUNCTUATION.

```
50 PRINT "THE PRODUCT IS" ; C
```

> EACH OF THESE HAS A PRECISE MEANING!
**Numerical Variables**

Think of a variable as a labeled box in memory!

A numerical variable in BASIC is like a variable in algebra. It assumes a numerical value, which may vary (but it has only one value at a time!). Only these symbols can be used as variables:

A, B, C, D, ..., Z
A0, A1, ..., and 0, 1, 2, 3, ..., 10, 11, ..., 20
A1, B1, ..., everything between! A9, B9, ..., Z9

There are several ways to assign a value to a variable: one is the READ DATA statement.

20 READ A, B
30 DATA 5.6, 1.1

This instructs the computer to assign the numerical values in the DATA statement — in order — to the variables in the READ statement.

20 READ A, B, C
30 DATA 5.6, 1.1

Think of a bug! This is a bug!

Another way to assign values to variables is with LET.

10 LET Q = 6.5
20 LET R = 2 * Q
30 LET S = Q + 2 + R + 10

The LET statement assigns the value on the right of the equality sign "=" to the variable on the left. The right-hand side may be a number, or some mathematical expression involving other variables — as long as they already have values!!

10 LET Q = 6.5
20 LET R = 0.5 * R
30 LET S = Q + 2 + R + 10

Here statement 20 does not assign any value to R, because R is not on the left side of "=". In fact, if R has not been assigned some value earlier in the program, then statement 20 gives Q an indeterminate value but —

10 LET M = 0
20 LET M = M + 1
30 LET M = M + 1

These strange-looking statements are perfectly O.K. "LET M = M + 1" means "assign to the variable M a value equal to its current value plus 1."
PRINT

This is an output command, meaning "display on the screen," not "print on paper."

WHAT CAN BE PRINTED?

You can print any text.

10 PRINT "ANY NUKE'S TODAY?"
RUN
ANY NUKE'S TODAY?

Print a variable and you get its value:

10 LET X = 77001
20 PRINT X
30 PRINT A
40 LET A = 1
50 PRINT "INFINITY IS MORE THAN"
60 PRINT A
70 PRINT A
80 PRINT "INFINITY IS MORE THAN"
90 PRINT A
100 PRINT "INFINITY IS MORE THAN"
110 PRINT A

It's OK to abbreviate this:

10 LET A = 1
20 PRINT "INFINITY IS MORE THAN"
30 PRINT A
40 PRINT "INFINITY IS MORE THAN"
50 PRINT A
60 PRINT "INFINITY IS MORE THAN"
70 PRINT A
80 PRINT "INFINITY IS MORE THAN"
90 PRINT A
100 PRINT "INFINITY IS MORE THAN"
110 PRINT A

For example, we could rewrite the program on p. 208.

10 REM BASIC MULTIPLICATION
20 READ A, B
30 DATA 5.6, 1.1
40 LET C = A * B
50 PRINT "THE PRODUCT OF "; A; "AND"; B; "IS"; C; "IS"
60 END
70 PRINT THE PRODUCT OF 5.6 AND 1.1 IS 6.16.

> There are also some nifty tricks using the comma and print, but we won't get into it...
**Input**

The statement allows the user to assign values to variables while the program is running.

**The form of the statement:**

```
INPUT A
```

When the program runs and reaches an input statement, the screen displays:

```
?
```

This indicates that the program has halted, awaiting input. You type some number (followed by "RETURN" as always!).

**AND THE PROGRAM CONTINUES RUNNING.**

"INPUT" AND "PRINT" CAN BE USED IN COMBINATION TO LET YOU KNOW WHAT SORT OF INPUT IS EXPECTED:

```
10 BASIC DIVISION
20 PRINT "TYPE THE NUMERATOR."
30 INPUT N
40 PRINT "TYPE THE NON-ZERO DENOMINATOR."
50 INPUT D
60 PRINT N; "; D; " ; N/D
70 END

RUN
TYPE THE NUMERATOR.
? 5
TYPE THE NON-ZERO DENOMINATOR.
? 8
5/8 = 0.625
```

**Goto**

This is the unconditional branching instruction.

"GO TO (LINE NUMBER)" TRANSFERS CONTROL TO A LINE OTHER THAN THE NEXT. THE PROGRAM THEN CONTINUES FROM THERE, AS IN THIS ENDLESS LOOP:

```
10 LET A=0
20 PRINT A
30 LET A=A+1
40 GO TO 20
```

**IF-Then**

Is the "smart" conditional jump.

It has the general form

```
IF (Condition) THEN (Line Number).
```

The condition has the form:

```
{ < } \{ } \{ > } \{ } \{ = } \{ } \{ = } \{ } \{ * } \{ }
```

**AS IN**

```
IF A=B THEN 30
```

This always includes the unstated instruction, "Otherwise, go to the next line."

* < Less Than, <= Less Than Or Equal To, > Greater Than, >= Greater Than Or Equal To, <> Does Not Equal.

```
10 LET A=0
20 PRINT A
30 LET A=A+1
40 IF A<=2 THEN 20
50 END
```

```
"OTHERWISE, NEXT LINE!"
```

**RUN 0 1 2**
THIS IS ENOUGH TO WRITE BASIC PROGRAMS FOR THE TWO ALGORITHMS FROM P. 201:

**ROMMATE RECEIPTS**

THE FLOW CHART:

BEGIN

INPUT L, S

YES

L > 5?

NO

BEGIN

PRINT "LISA SPENT"
20 INPUT L
30 PRINT "SOPHIE SPENT"
40 INPUT S
50 IF L > 5 THEN 80
60 PRINT "LISA OWES SOPHIE"; (S - L)/2
70 DO TO 90
80 PRINT "SOPHIE OWES LISA"; (L - S)/2
90 END

SEE HOW "IF-THEN" AND "DO TO" ARE USED? IF L > 5, THEN LINES 60 AND 70 ARE NOT EXECUTED, OTHERWISE, THEY ARE EXECUTED, AND LINE 90 TO ENSURE THAT LINE 80 IS SKIPPED.

THE PROGRAM:

10 PRINT "LISA SPENT"
20 INPUT L
30 PRINT "SOPHIE SPENT"
40 INPUT S
50 IF L > 5 THEN 80
60 PRINT "LISA OWES SOPHIE"; (S - L)/2
70 DO TO 90
80 PRINT "SOPHIE OWES LISA"; (L - S)/2
90 END

NOW WE NEED A PROGRAM TO ROUND OFF THE HALF PENNY!

**MULTIPLE FLOG-INS**

THE FLOW CHART:

BEGIN

X = 0

PRINT X

X = X + 2 * X + 10

ADD 01 TO X

X = X / 2

YES

NO

END

THE PROGRAM:

10 REM LINE 20 PRINTS A HEADING
20 PRINT "X" 5 SPACES
30 LET X = 0
40 LET Y = X * 2 + X * 10
50 PRINT X; Y 8 SPACES
60 LET X = X + 0.1
70 IF X <= 2 THEN 40
80 END

RUN

X

0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0
1.1
1.2
1.3
1.4
1.5
1.6
1.7
1.8
1.9
2.0

Y

1.0
1.0
1.2
1.4
1.6
1.8
2.0
2.2
2.4
2.6
2.8
3.0
3.2
3.4
3.6
3.8
4.0

YOU COULD DRAW A GRAPH WITH THIS INFORMATION!
The "multiple plug-ins" loop is so typical that all programming languages have special commands just for such repetitions. In BASIC, it's

```
30 LET X = 0
...  
60 LET X = X + 0.1
70 IF X <= 2 THEN 30
```

This replaces these three lines:

```
30 FOR X = 0 TO 2.9 STEP 0.1
  GO TO NEXT X
```

This is replaced with these two:

```
30 FOR X = 0 TO 2.9 STEP 0.1
  60 NEXT X
```

The statement initially sets the variable equal to the lower limit, executes the lines up to "NEXT," increments the variable by the amount "STEP," and repeats the loop until the upper limit is exceeded.

Example:

```
10 FOR I = 1 TO 4
  20 PRINT I + 1
30 NEXT I
40 END
RUN
```

Omitting "STEP" automatically makes increment = 1.

PROBLEMS?

1. What does this program do?
```
10 INPUT N
20 FOR I = 1 TO N
30 PRINT I * I
40 NEXT I
50 END
```

2. Rewrite the "multiple plug-ins" program using the "FOR NEXT" statement.

3. Write a program which adds the integers (whole numbers) from 1 to 1,000,000. Ditto from 1 to N, for any N.

4. In the Fibonacci sequence 0,1,1,2,3,5,8,13,21,... each number is the sum of the previous two numbers. Write a program which generates this sequence.

5. Read enough of a BASIC textbook to write a "Roommate Receipts" program for any number of roommates.
There are plenty of other basic features, enough to fill entire books—and in fact tons of books on basic have been published.

Software Survey

So...if you're interested in discovering string variables, subroutines, functions, arrays, nested loops, how to deal with disks and avoid bugs, etc etc, then go to your local library or bookstore and get started!

Here's a look at several important areas of software which have emerged in the years since ENIAC...
SYSTEMS SOFTWARE

Programs are commonly divided into systems software and applications software.

Applications software does "real world" jobs, while systems software exists purely to regulate the computer system itself.

A system typically consists of one or more input/output devices (terminals, printers, card readers, communications ports), processors, memory units (main and mass), and who knows what else. Someone has to coordinate it all!

THE PROGRAM THAT DOES IT IS CALLED THE OPERATING SYSTEM.

If you think of the computer's core as a giant electronic filing cabinet (with a calculator attached), then the operating system:

* creates the structure of the files
* manages memory so that different files don't bump into each other
* regulates access to the files and the movement of information to and from other parts of the system...

ETC!

Besides the operating system, system software includes other programs "in the system," such as loaders (which load programs into memory) and compilers (which translate higher-level languages into machine code).

ALL INVISIBLE TO THE USER!
**DATA BASE MANAGEMENT**

A **DATA BASE** is just a big pile of information: a library's card catalog, a bank's transaction records and account balances, an airline's flight schedules and reservations, police files, stock exchange data—all are data bases.

A **DATA BASE MANAGEMENT PROGRAM** organizes, updates, and provides access to the data base.

In the case of an airline, for example, the computer keeps track of reservations, assigns seats, erases reservations when the customer cancels, makes reassignments if a flight is canceled, prints the tickets, and provides all the flight information to travel agencies worldwide.

**WORD PROCESSING**

A "personal" use for computers...

Word processing software allows you to write, edit, and format text—all from the same keyboard. You can go from first to final draft electronically, before ever printing a word.

There are also programs to correct spelling and even to fix syntax and grammar. Soon illiterate will be creating masterpieces!

Hamlet by J. Fred Shakespeare...

A small computer with word processing can be quite inexpensive... the catch is that a "letter quality" printer can cost ten times the price of a typewriter.

An incentive to computer crime!
SCIENCE DEPENDS ON MATHEMATICS, AND COMPUTERS ARE SUPER MATH MACHINES. THE FASTEST, MOST POWERFUL COMPUTERS ARE MAINLY APPLIED TO SCIENTIFIC PROBLEMS.

CRAY-1 COMPUTER, CAPABLE OF 100 MILLION OPERATIONS PER SECOND

THESE "SUPERCOMPUTERS" EXCEL AT SIMULATION. THE IDEA BEHIND SIMULATION IS TO FEED THE COMPUTER THE EQUATIONS GOVERNING A PHYSICAL SYSTEM AND THEN MATHEMATICALLY "MOVE" THE SYSTEM ACCORDING TO THOSE LAWS.

TIME SPACE TRAVEL: A COMPUTER CAN GUIDE A CRAFT TO THE MOON, BECAUSE IT CAN INTERIALLY SIMULATE THE ENTIRE FLIGHT!

COMPUTERS CAN SIMULATE:

- THE INTERIOR OF A STAR - OR A NUCLEAR EXPLOSION...
- THE EVOLUTION OF AN ECOSYSTEM

CLIMATE (ALTHOUGH EVEN THE FASTEST COMPUTER ISN'T FAST ENOUGH TO PREDICT THE WEATHER).

I'LL TELL YOU TOMORROW'S WEATHER NEXT WEEK!
GRAPhICS

FROM THE SIMPLEST "PONG" SCREEN TO THE MOST SOPHISTICATED PILOT SIMULATOR, THE IDEA IS THE SAME:

DIVIDE THE SCREEN AREA INTO A LARGE NUMBER OF TINY RECTANGLES ("PIXELS") AND ASSIGN EACH ONE A COLOR AND BRIGHTNESS.

THAT'S WHY COMPUTER PICTURES HAVE CORNERS!

BUT THERE ARE ALSO ALGORITHMS FOR SMOOTHING CORNERS!

UNFORTUNATELY, IT TAKES A LOT OF COMPUTER POWER TO DO FANCY GRAPHICS. SMALL COMPUTERS MOSTLY DO THINGS LIKE MAKE PIE CHARTS...

IF ONLY THEY COULD MAKE REAL PIES...

THEN I'D BE IMPRESSED!

COMMUNICATION

THE BIGGEST COMPUTER SYSTEM OUTSIDE GOVERNMENT BELONGS TO THE TELEPHONE COMPANY.
A VOICE (OR ANY OTHER SIGNAL) CAN BE DIGITALLY ENCODED, TRANSMITTED, AND DECODED.

COMPUTERS ALSO CONTROL THE ROUTING AND SWITCHING OF CALLS THROUGH THE NETWORK AND KEEP TRACK OF EVERYONE'S BILL!

COMPUTERS CAN BE PROGRAMMED TO RECOGNIZE PARTICULAR WORDS OR GRIPS OF WORDS — A CAPABILITY NOT LOST ON THE INTELLIGENCE COMMUNITY.

WE CAN AUTOMATICALLY RECORD ANY CONVERSATION CONTAINING WORDS I CAN'T SAY BECAUSE I DON'T WANT TO BE RECORDED...
ARTIFICIAL INTELLIGENCE

Despite their incredible speed and accuracy, computers are lousy at pattern recognition, analysis, music-playing, and understanding human language!

Can a machine be programmed to think?

Er...well...um...ah...let me see...

Actually, we know very little about how thinking works...

So a better question is: How can you tell if a machine is thinking?

Alan Turing suggested this test: Suppose you could communicate with something, or someone, concealed from view. If, on the basis of the conversation, you couldn't say whether it was machine or human, you would have to say it was thinking.

It's a machine!

Yeah, well...I have my doubts about you, Jack!

I personally dislike this criterion, on the grounds that a simulation isn't the real thing...

How do you create an expert system?

First, interview a bunch of experts - geologists, for example - and force them to spell out the algorithms behind their skills, hunches and brainstorming.

Then load the computer's memory with the humans' knowledge base...and the result is (sometimes) a program which can outperform any human!

This philosophical muddle hasn't stopped people from trying to make machines think. They've had some success with so-called expert systems, which mimic human experts in various fields.
CRYPTOGRAPHY

Ordinarily, information is stored as a binary string any computer can read: the plaintext. In cryptographic jargon, to encrypt it you apply some algorithm $S$, which converts it to a scrambled message called the ciphertext.

\[
\text{plaintext} \rightarrow S \rightarrow \text{ciphertext}
\]

Theoretically, it's impossible to reconstruct the plaintext from the ciphertext without knowing something about $S$. However, a potential code-breaker could put a computer to work searching for $S$.

To be secure, $S$ has to be so complicated that even the fastest computer would take, say, a few million years to figure it out!

Recently, the National Bureau of Standards approved a family of algorithms as a data encryption standard for the nation. Several scientists suspect that this standard is just complex enough to stymie ordinary computers, but not too tough for the nine acres of computers of the National Security Agency.

Secret codes used to be strictly military and spy stuff, but now more and more sensitive information is stored in computer systems:

- Medical records
- Bank records
- Census data
- Income tax records
- Grade transcripts
- Corporate memos, etc., etc.

Scrambling data has become an important way of protecting privacy!!

There are standard codes like ASCII (p. 128) for converting written text into binary... but what about using computers for secret codes??

Secret codes used to be strictly military and spy stuff, but now more and more sensitive information is stored in computer systems:
CADICAM: COMPUTER-POWERED DESIGN / COMPUTER-POWERED MANUFACTURE

Using a combination of speedy calculation and high-resolution graphics, computers can help design nearly anything—from jets to lenses to type styles to other computers.

Then they can go on to control automatic manufacturing processes as well. Yes, robots are already here!

WAR

The military can use just about every type of software we've mentioned—and then some!

ENIAC was built for calculating ballistics... now we have ballistic missiles!

Flight simulators can train pilots right on the ground.

Great graphics on these!

Then there are the famous 'smart' missiles, which can follow a moving target...

Supercomputers help design nukes...

...not to mention data processing and cryptography. So great is the Defense Department's software need that they have their own programming language: ADA, named after the unfortunate Lady Lovelace.
This little survey only begins to suggest the range of software currently available. Every day there's more... some programs move into new areas, while others integrate existing routines into new, more powerful packages.

**In Conclusion,**

A few words about this familiar sentence:

**Computers only do what people tell them to do!**

If you're looking for opportunity in the computer business, consider this: the total consumption of software, which began as a small fraction of computing costs, is expected to reach many times the amount spent on hardware over the next few decades!

(Which is what computer scientists say when they want to be reassuring...)

This baby does word processing, manages a gerbil ranch, and designs H-bombs! Every gerbil gets its own deterrent!

Write software!
TECHNICALLY, IT'S TRUE, IN THE SENSE THAT SOFTWARE CONTROLS COMPUTERS, AND PEOPLE WRITE SOFTWARE...

BUT WHO CONTROLS PEOPLE?!

FOR EXAMPLE, SUPPOSE A NATION'S STRATEGIC PLANNERS DECIDED TO PROGRAM THEIR COMPUTERS TO ORDER A MISSILE ATTACK AUTOMATICALLY "ON WARNING." CONSIDERING THAT U.S. DEFENSE COMPUTERS SOUNDED SEVERAL FALSE ALARMS A YEAR, IS THIS REASSURING?!!

I WAS ONLY FOLLOWING INSTRUCTIONS

MY CALCULATOR SAYS $2^{16} \approx 65,536.001$ (REALLY?!

ANOTHER PROBLEM IS THAT ALGORITHMS DON'T ALWAYS DO EXACTLY WHAT THEY ARE SUPPOSED TO.

LARGE SOFTWARE SYSTEMS ARE WRITTEN BY TEAMS OF PROGRAMMERS. LIKE THE ELEPHANT, NO ONE UNDERSTANDS THE WHOLE THING!

COMPUTERS ROUTINELY DO BIZARRE AND UNEXPECTED THINGS, ESPECIALLY WHEN RUNNING NOW, UNTESTED SOFTWARE!

I'M GETTING TO BE A TIRED METAPHOR!
Finally, consider this ominous algorithm:

BEGIN

REPRODUCE YOURSELF

DOES ANYTHING STAND IN YOUR WAY?

YES

DESTROY IT!

NO

AND IF YOU THINK THAT BECAUSE "IT'S ONLY A MACHINE," YOU CAN ALWAYS TURN IT OFF, PONDER THE WORDS OF NOBERT WIEHER, A SCIENTIST WHO THOUGHT DEEPLY ABOUT THESE THINGS:

"TO TURN A MACHINE OFF EFFECTIVELY, WE MUST BE IN POSSESSION OF INFORMATION AS TO WHETHER THE DANGER POINT HAS COME. THE MORE FACT THAT WE HAVE MADE THE MACHINE DOES NOT GUARANTEE THAT WE SHALL HAVE THE PROPER INFORMATION TO DO THIS.... THE VERY SPEED OF MODERN DIGITAL MACHINES STANDS IN THE WAY OF OUR ABILITY TO PERCEIVE AND THINK THROUGH THE INDICATIONS OF DANGER."

* CYBERNETICS, SECOND EDITION, P. 179

WHILE NO COMPUTER IS INTELLIGENT, MOBILE, OR WELL EQUIPPED ENOUGH — YET — TO EXECUTE THESE INSTRUCTIONS, SUCH A MACHINE REMAINS A THEORETICAL POSSIBILITY. THIS PROGRAM WOULD MAKE IT SOMETHING VERY MUCH LIKE A COMPETING LIFE FORM!!

BLAST

SO WELCOME TO THE INFORMATION AGE, AND HAPPY COMPUTING!!
Calculation, 34
Calculators, 36
mechanical, 59
Calculi, 34
Cards, punched, see Punched cards
Carroll, Lewis, 100
Carry bit, 125
Cash registers, 59
Census tabulators, 60
Charge-coupled devices, 156
Chinese arithmetic table, 29
Chinese number system, 27-29
Chinese writing, 22
Circuits
electrical, completing, 62
integrated, 84
Clocks, 142-143
Codes, secret, 233-233
Combination logic, 142
Communication, 235
Computers, 205, 213
Computer-aided design/computer-aided manufacture, 234
Computers, 5-6
clocks and, 142-143
control of, 217-241
Cray-1, 226
describing, 89
evolution of, 14
IBM Personal, 96
information and, 6
problems with, 237-241
research in, 153
size of, 152
Conditional branch box, 198
Conditional jumps, 57, 183
"smart," 215
Control, 170
transfer of, 183
Control bus, 170
Control flow, 93-94
Control unit, 92
Core memories, 158
Counters, 146-148
asynchronous ripple, 148
synchronous, 148
Counting
binary and decimal, 120
digital, 19
Cray-1 computer, 226
Cradles, 44
Cryptography, 233-233
Cycles, 142
Cypher text, 233
Data base management, 324
Data bays, 224
Data processing, 63
Decimal system, 11
Binary numbers translated into, 119
Decoders, 119
Deductive logic, 99
"Difference Engine, The," 51-53
Digital counting, 19
Disallowed input, 173
Disks, magnetic and floppy, 166-167
DNA, 11
DNA-protein system, 13
DNA technology, recombinant, 86
Duns Scotus, 99
EBBCDC, 128
Eckert, J. Presper, 74
Egyptian number system, 26
Egyptians, 12
Electric "mouse," 7
Electrical circuits, completing, 62
Electricity, 65
Electromechanical memories, 154
Electromechanical switches, 71
Electronic memories, 154
Electronic Numerical Integrator and Calculator (ENIAC), 73-76
Encodors, 129
Encoding instructions, 79
End box, 198
ENIAC (Electronic Numerical Integrator and Calculator), 73-76
EPROM, 164
Expert systems, 231
External storage of information, 20
Facts, 7
Fetching instructions, 178
Fibonacci sequence, 219
Fields, 176
Fiven, counting by, 23
Flight simulators, 235
Flip-flop inputs, 137
Flip-flops, 133-137
master-slave, 144
Floating point representation, 127
Floppy disks, 166-167
Flow charts, 198-200
two examples of, 199
for multiple plug-ins, 203
for roommate receipts, 203
For-next commands, 218
FORTRAN, 206
Gating network, 140
Gene, 12
Glitches, 143
Go-to statement, 215
Grammar, laws of, 18
Graphics, 218
Gravitation, theory of, 45-46
Greek mathematicians, 33
Handfuls, counting by, 24-25
Hardware, 187
Hertz (one cycle per second), 142
Hexadecimal numerals, 157
Higher-level programming languages,
205
Hindas, 37-38
Holleith, Herman, 65, 66
IBM, 64
IBM Personal Computer, 96
If-then statement, 215
Inca, 12
Increments, 146
Inductive logic, 99
Industrial Revolution, 49
Information
ages of, 1-86
alphanumeric, 130
computers and, 5
defined, 7-8
excess, 3-5, 86
external storage of, 20
forms of, 8-9
power of, 12
stored, 20
Information flow, 93-94
Information processing, 11
understanding, 90
Information theory, 7
Input, 48, 92, 95
card-reading device, 54
disallowed, 137
flip-flop, 137
Input box, 198
Input-output (I/O) tables, 110, 112-113
Input statement, 214
Input wire, 166
Instruction register, 173
Instruction set, Motorola 6800, 182
Instructions, 48
branch or jump, 183
8-bit, 156
encoding, 79
fetching, 178
machine, 176
microinstructions, 178
to mill, 53
Integers, 127
Integrated circuits, 84
Integration, large-scale and very
large-scale, 84
Intelligence, artificial, 230-231
Internal memory, 155
Interpreters, 205
Inverters, 108
I/O (input-output) tables, 110, 112-113
Jaccard, Joseph Marie, 59
Japanese calculation of pi, 29
Jump, conditional, see Conditional jumps
Jumps instructions, 183
K (kilo), 163
Lagrange assembly, see Assembly language
entries
BASIC, see BASIC language
expressive, 17
higher-level programming, 103
machine, 177
Large-scale integration (LSI), 84
Laurens, 138
good, 140
Leibniz, Gottfried Wilhelm, 47
Let statement, 211
Life form, 13
composing, 240
Line numbers, 209
<table>
<thead>
<tr>
<th>Term</th>
<th>Page(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Switches (cont.)</td>
<td></td>
</tr>
<tr>
<td>- electromechanical, 71</td>
<td></td>
</tr>
<tr>
<td>- patterns of, 70</td>
<td></td>
</tr>
<tr>
<td>Symbol-manipulation, abstract, 42</td>
<td></td>
</tr>
<tr>
<td>Symbolic logic, 100</td>
<td></td>
</tr>
<tr>
<td>Synchronous counters, 148</td>
<td></td>
</tr>
<tr>
<td>Systems, 222</td>
<td></td>
</tr>
<tr>
<td>- expert, 231</td>
<td></td>
</tr>
<tr>
<td>- operating, 223</td>
<td></td>
</tr>
<tr>
<td>Systems software, 222</td>
<td></td>
</tr>
<tr>
<td>Tabulators, census, 60</td>
<td></td>
</tr>
<tr>
<td>Tally bone, 20, 23</td>
<td></td>
</tr>
<tr>
<td>Tape, paper and magnetic, 165</td>
<td></td>
</tr>
<tr>
<td>Tartaglia, Niccolo, 45</td>
<td></td>
</tr>
<tr>
<td>Telephone company, 229</td>
<td></td>
</tr>
<tr>
<td>Telephone relay, 69</td>
<td></td>
</tr>
<tr>
<td>Telephones, 68</td>
<td></td>
</tr>
<tr>
<td>Ten, 116-117</td>
<td></td>
</tr>
<tr>
<td>Tens, counting by, 23</td>
<td></td>
</tr>
<tr>
<td>Three-body problem, 46</td>
<td></td>
</tr>
<tr>
<td>Timing, 142</td>
<td></td>
</tr>
<tr>
<td>Toggle switches, 67</td>
<td></td>
</tr>
<tr>
<td>Toggling, 147</td>
<td></td>
</tr>
<tr>
<td>Transfer of control, 183</td>
<td></td>
</tr>
<tr>
<td>Transistors, 83-84</td>
<td></td>
</tr>
<tr>
<td>Transition rules, 191</td>
<td></td>
</tr>
<tr>
<td>Truth-values, 102</td>
<td></td>
</tr>
<tr>
<td>Tube, vacuum, 69</td>
<td></td>
</tr>
<tr>
<td>Turing, Alan, 193, 230</td>
<td></td>
</tr>
<tr>
<td>Turing machines, 191-192</td>
<td></td>
</tr>
<tr>
<td>Two, powers of, 118</td>
<td></td>
</tr>
<tr>
<td>&quot;Two's complement&quot; method, 121</td>
<td></td>
</tr>
<tr>
<td>Type-setter, automated, 54</td>
<td></td>
</tr>
<tr>
<td>Unconditional branching statement, 215</td>
<td></td>
</tr>
<tr>
<td>Universal Turing machine, 192</td>
<td></td>
</tr>
<tr>
<td>Vacuum tube, 69</td>
<td></td>
</tr>
<tr>
<td>Variations, 210</td>
<td></td>
</tr>
<tr>
<td>- numerical, 210-211</td>
<td></td>
</tr>
<tr>
<td>Very large-scale integration (VLSI), 84</td>
<td></td>
</tr>
<tr>
<td>von Neumann, John, 77, 193</td>
<td></td>
</tr>
<tr>
<td>Wiener, Norbert, 241</td>
<td></td>
</tr>
<tr>
<td>Word processing, 225</td>
<td></td>
</tr>
<tr>
<td>Words, 8</td>
<td></td>
</tr>
<tr>
<td>World War II, 72</td>
<td></td>
</tr>
<tr>
<td>Writing, 21-22</td>
<td></td>
</tr>
<tr>
<td>Chinese, 22</td>
<td></td>
</tr>
<tr>
<td>Written zero, 37-38</td>
<td></td>
</tr>
<tr>
<td>Zero, 17-18</td>
<td></td>
</tr>
<tr>
<td>- written, 37-38</td>
<td></td>
</tr>
<tr>
<td>Zuse, Konrad, 71</td>
<td></td>
</tr>
</tbody>
</table>