CHAPTER 26

How Assembly Language Works

ASSEMBLY LANGUAGE is the lowest level of programming that you are likely to encounter. Each instruction in assembly language corresponds to a single instruction for a *central processing unit* (CPU). A program called an *assembler* translates your assembly language commands into the numbers that your computer hardware can execute. Since assembly language is closely involved with the CPU hardware, every type of CPU has its own unique assembly language, though some CPUs are intentionally designed to execute all or part of the instruction set of another CPU.

To use assembly language, it is necessary that you understand a little bit about how computers work internally. A high-level language allows you to declare and work with variables, without knowing where or how these variables are stored, and will typically handle all aspects of allocating space for those variables. It can also support user-defined variables and types of variables the hardware does not support directly. In most cases, assembly language programmers specify the location and order of variables in memory, and are limited to types of variables that the CPU itself can manipulate.

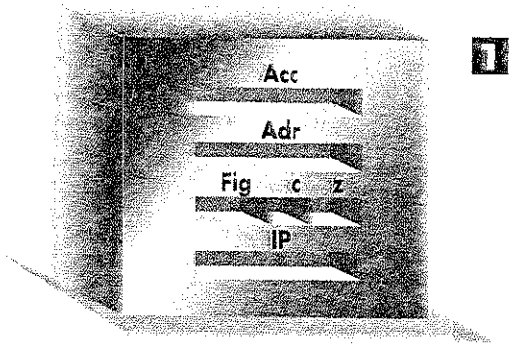
Assembly language programmers must also work with *registers*, a special type of memory that is built into the CPU itself. Registers come in many types and sizes, depending on the hardware. The CPU can access registers much more quickly than it can access the computer's main memory. This means a complex operation will usually involve loading data from memory into registers, performing operations on the registers, and then writing the results back into memory.

There are several types of registers that are common to most CPUs. Most CPUs have at least one *accumulator*, which is also sometimes referred to as a *data register* or a *general purpose register*. Accumulators support all of the fundamental arithmetic and Boolean operations and are used for calculations and for temporary storage of information. The number of bits in these registers determines the width of the processor; thus a CPU with 32-bit-wide data registers is called a 32-bit processor.

There are three types of registers that are also pointers. An *address register* is a pointer that contains memory locations. It may support address calculations as well. An *instruction pointer* (IP) is a special address register that tells the CPU the location in memory of the next instruction to execute. A *stack pointer* is a special address register that tells the CPU the location of the top of the CPU stack. Most CPUs provide instructions for implementing a stack in memory that is used for parameter passing and for storing the return address on subroutine calls. Stacks were discussed earlier in Chapter 14.

Few applications today are written entirely in assembly language. In most cases, it is used for only parts of a program where performance is absolutely critical. Unlike most other languages, assembly allows you to optimize your code to obtain the best possible speed by deciding what data to keep in registers at a given time. Although higher-level languages have special built-in optimizing algorithms that use CPU registers efficiently, they can't achieve the same performance as assembly language in the hands of a good programmer. Of course, the price for such performance is hard work, since it is more difficult and more time consuming to program in assembly language than in high-level languages.

Assembly Language



1 Using a language such as BASIC to add two variables is this easy: Result = VarA + VarB. Let's take a look at the assembly language version of this operation on an imaginary 8-bit CPU that can access only 256 bytes of memory and has only the four registers shown.

2 Imagine a CPU that has a few machine language instructions. Each instruction consists of a number that represents a command that the CPU understands how to execute, and an assembly language name.

A program called an assembler converts the assembly language into machine code. Your assembly language program consists of a text file that contains assembly language names, and special commands such as the DB command that are instructions to the assembler. The assembler reads the text file and translates it into machine code, which it loads into consecutive locations in memory.

It is important to understand the difference between a command that produces machine code and a command to the assembler. Commands to the assembler tell the assembly program to perform an operation during the assembly process—for example, to allocate space in memory or to output a program listing. These commands do not appear in the final executable file. Modern macro assemblers can have many powerful assembler commands that make assembly language programming easier.

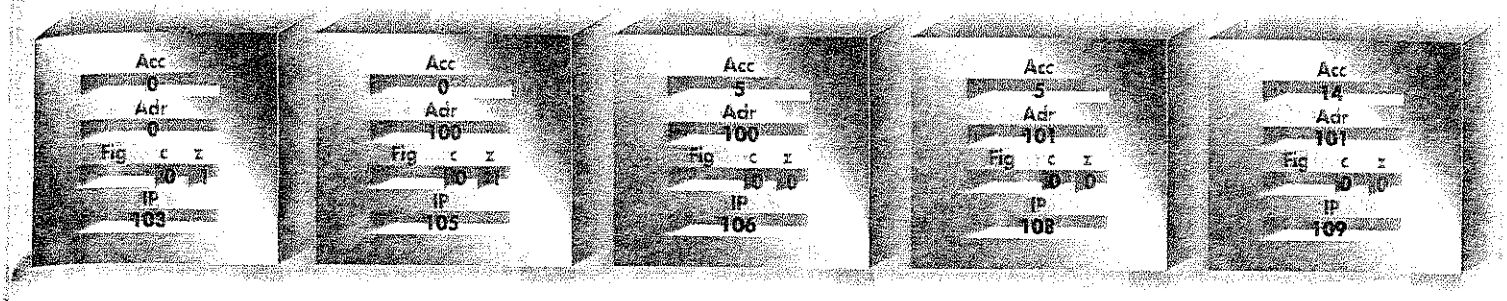
Number	Name	Description
1	Add	The contents of the memory location that Adr points to is added to Acc.
2	Load	Loads Acc with the contents of the memory location pointed to by Adr.
3	Store	Stores the contents of Acc in the memory location pointed to by Adr.
4,N	LoadA, N	Loads Adr with the value N. The memory location after the command (4) contains the value N.
	DB N = X	Tells assembler to allocate memory for a variable. Does not produce machine code, but simply leaves an empty space in memory. N is the number of bytes to allocate. X is the initial value to load into the allocated space.

Address	Value	Assembly	Description
100	5	VarA: DB 1=5	VarA is at this address
101	9	VarB: DB 1=9	VarB is at this address
102	0	Result: DB 1=0	Result will be at this address
103	4	LoadA, VarA	First of two bytes of LoadA command
104	100		Adr now points to VarA
105	2	Load	Load Acc with VarA; Acc is now 5
106	4	LoadA, VarB	First of two bytes of LoadA command
107	101		Adr now points to VarB
108	1	Add	Adds Acc with VarB; Acc is now 14
109	4	LoadA, Result	First of two bytes of LoadA command
110	102		Adr now points to Result
111	3	Store	Stores Acc to Result; Result is now 14

3 Let's assume that VarA contains the value 5 and is found at address 100 in memory, and VarB contains the value 9 and is found at address 101. Variable Result, at address 102, can be loaded with VarA plus VarB using the assembly language program (shown here) that starts at location 103. This program resembles a typical listing produced after the assembler has converted your program to machine language. The Assembly column contains the assembly language program itself.

NOTE Assembly language is not always the lowest-level language; rather, it is only the lowest level available to programmers using a particular CPU. The CPU itself often contains an even lower level of code called *microcode*. This is the language that controls the internal operation of the CPU itself, and is typically used solely by the manufacturer.

4 The actual executable file for this function would contain a start address that is set by the programmer. The operating system would start by loading the IP register with the starting address. Here you can see the contents of the CPU registers as we trace through the first few statements in this program.



LoadA, VarA Load LoadA, VarB Add

5 As you can see, it took six assembly language statements to perform this simple addition, versus a single line in BASIC. But each one takes very little space in memory and performs far more quickly than the equivalent BASIC statement. A BASIC interpreter might actually perform thousands of machine instructions to perform the same operation. This is because it must analyze the code, find the variables in memory, and finally perform the operation.

6 What if you wanted to add two 16-bit numbers? The assembly language code to add two 16-bit variables might look as shown here as we add the hex values H685 and H182. (Data values are shown in hexadecimal to show the division into two 8-bit bytes. They use the carry flag to indicate that the result of an addition does not fit in the available space, just as you "carry" numbers when doing long addition, and the result for a column is greater than 9. See Chapter 6 for a review of hexadecimal operations.)

As you can see, the need to break the addition into two 8-bit bytes adds many instructions to the program. The extra instructions would not be necessary if you were using a 16-bit processor, one where the internal registers are 16 bits wide. This is one of the reasons performance improves with bus width. By the way, there is a bug in this listing: If the carry flag were set to 1 at the beginning of the routine, the final result would be incorrect. A correct routine would force the carry flag to 0 before the addition operation began.

Address	Value	Assembly	Description
100	H685	VarA: DB 2 = H685	VarA is at this address; 2 bytes now
102	H182	VarB: DB 2 = H182	VarB is at this address
104	0	Result: DB 2 = 0	Result will be at this address
106	4, 101	LoadA, VarA+1	Adr now points to low byte of VarA
108	2	Load	Load Acc, which is now H85
109	4, 103	LoadA, VarB+1	Adr now points to low byte of VarB
111	1	Add	Adds Acc with low byte of VarB; Acc is now 7, and the C (carry) flag is set to one
112	4, 105	LoadA, Result+1	Adr now points to low byte of Result

The EVOLUTION of the CAMERA

1500
CAMERA OBSCURA

Used a pinhole to project an inverse image when light passed through the aperture

1839
DAGUERREOTYPE CAMERA

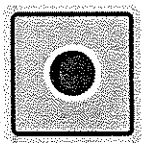
First publicly announced photographic process and the first to come into widespread use

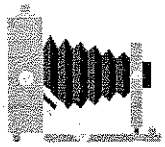
1844
THE MEGASKOP

First camera to take a panoramic image

1900
THE KODAK BROWNIE

Popularized low-cost photography and introduced the concept of the snapshot



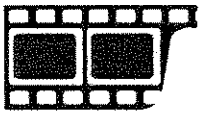


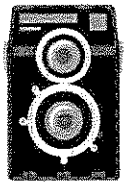
1900
THE RAISECAMERA

The first light-weight portable camera

1925
THE LEICA I

First compact camera to use 35mm film



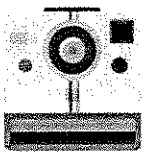


1947
THE DUAFLEX

First compact SLR with adapted eye level viewfinder

1947
POLAROID MODEL 95

Polaroid photography invented

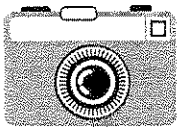


1975
THE KODAK DIGITAL CAMERA

Recorded black and white images on a cassette tape

1986
THE QUICKSNAP

The disposable camera invented by Fuji



1991
KODAK DCS SLR

First professional digital camera for sale in the US for $13,000

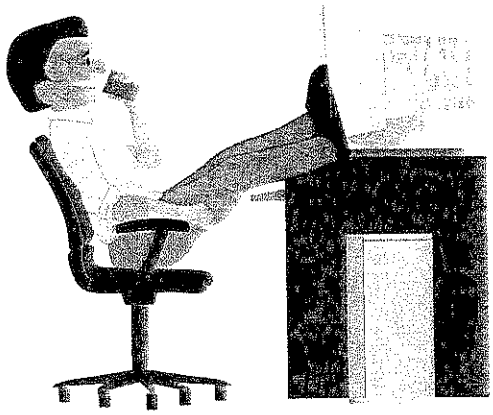
2000
SHARP J-SH04

The first camera phone

Content Source:
http://en.wikipedia.org/wiki/History_of_the_camera
http://inventors.about.com/od/pstartinventions/a/Photography.htm
http://photodoto.com/camera-history-timeline/
http://www.ehow.com/about_5283347_panoramic-camera-introduced.html

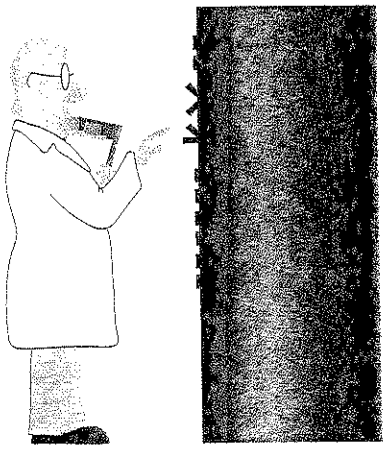
A Hierarchy of Languages

1 Most programmers use high-level languages such as C++, BASIC, and Java. These languages hide the details of the particular computer from the programmer, letting him or her work with commands and data provided by the language itself instead of worrying about a particular machine language or memory organization. This makes it possible to create programs that will run on many different computers. A program called a *compiler* or *interpreter* is used to convert a program in a high-level language into the machine language for a particular computer. One line in a high-level language might be converted into dozens of lines of machine language.



2 Before high-level languages were invented, programmers created a very simple language in which each command represents the numbers of individual machine-language commands. This is called an *assembly language*. A program called an *assembler* converts the assembly-language text into the numbers that the computer can use.

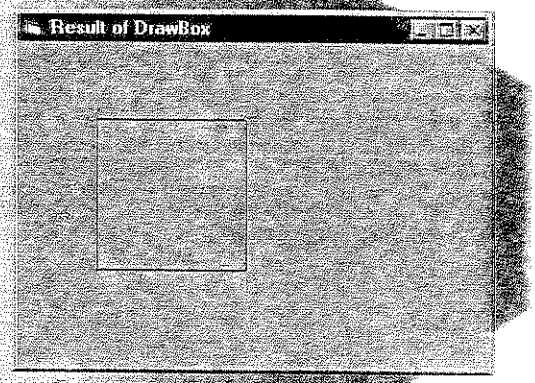
3 A computer represents all information internally in the form of numbers. This includes programs. The numbers that make up a program that the computer hardware is able to use directly are in the machine language of the computer. In the early days of computing, people would actually program in machine language, using switches or punch cards to enter the machine-language program directly into memory.



4 In this example, you have a simple program in the C language that draws a box. It is translated first into assembly language, and then into the machine language. This particular example was written for use with the Microsoft Windows operating environment. The assembly language and machine language shown is that for the Intel family of microprocessors. The numbers shown in the machine language screen are in base 16, also known as hexadecimal. You'll read more about that later. As you can see, the higher level C language is quite easy to understand, the assembly-language code is considerably more cryptic, and the machine language is meaningless to the casual reader. This demonstrates one of the major advantages of a high-level language.

```
VOID EXPORTAPI DrawBox(HDC hdc)
{
// hdc is a value used by Windows to identify a
// window to draw in.
// The numbers represent coordinates in the window.
   MoveTo(hdc, 50,50);
   LineTo(hdc, 100,50);
   LineTo(hdc, 100,100);
   LineTo(hdc, 50,100);
   LineTo(hdc, 50,50);
  }
```

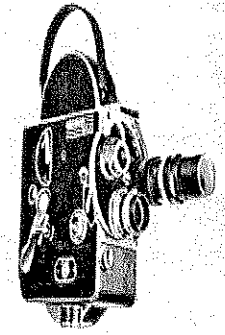
```
DrawBox:
   mov   ax,SEG con0
   enter OFFSET L08041,OFFSET 0
   push  si
   push  di
   push  ds
   mov   ds,ax
   push  WORD PTR 6[bp]
   push  OFFSET 50
   push  OFFSET 50
   call  FAR PTR MoveTo
   push  WORD PTR 6[bp]
   push  OFFSET 100
   push  OFFSET 50
   call  FAR PTR LineTo
   push  WORD PTR 6[bp]
   push  OFFSET 100
   push  OFFSET 100
   call  FAR PTR LineTo
   push  WORD PTR 6[bp]
   push  OFFSET 50
   push  OFFSET 100
   call  FAR PTR LineTo
   push  WORD PTR 6[bp]
   push  OFFSET 50
   push  OFFSET 50
   call  FAR PTR LineTo
   pop   ds
   pop   di
   pop   si
   leave
   ret   OFFSET 2
```




```
B8 B7 55
C8 02 00 00
56
57
1E
BE D8
FF 76 06
6A 32
6A 32
9A AA 0E 0F 05
FF 76 06
6A 64
6A 32
9A 96 0E 0F 05
FF 76 06
6A 64
6A 64
9A 96 0E 0F 05
FF 76 06
6A 32
6A 64
9A 96 0E 0F 05
FF 76 06
6A 32
6A 32
9A 96 0E 0F 05
E9 00 00
1F
5F
5E
C9
CA 02 00
```

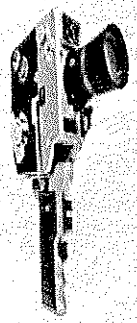
Even though they are using the same computer hardware, each of these programmers perceives the machine differently. We call the machine that they perceive a "virtual machine." For example, when you use a word processor on a computer, you don't worry about the language in which the word processing program was written—all you care about is that it handle your text correctly. Thus, the "virtual machine" that you see is a word processor.

1927



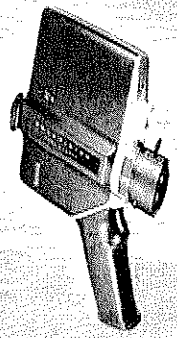
Bolex H-16
used 16mm film was popular for making home movies since 1927 to 1950s.

1965



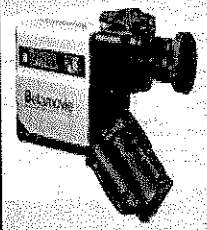
Super 8
used film in a cartridge was popular for making home movies in the 60s

1977



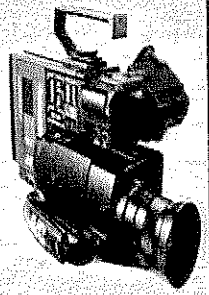
Polavision
used film in a cartridge. The catridge had to be destrod to view the film. Launched in 1977 to 1979

1983



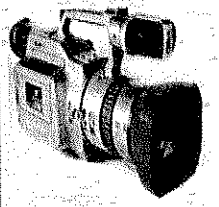
Betamovie BMC-100
the world's first camcorder with a mounted-video cassette recorder.

1984



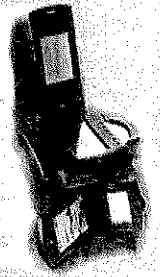
JVC GR-C1
was notable as the first all-in-one VHS camcorder by introducing VHS-C cartridge.

1995



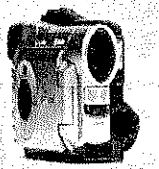
DCR-VX1000
Sony launcheched DCR-VX1000 which was the first camera to use MiniDV tape.

1995



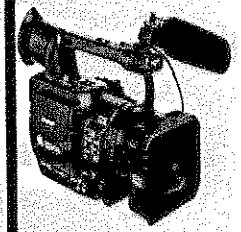
Apple video PhonePDA
Apple prototype mobile phone that is capable of recording video and taking photograph.

2000



MV100
Lanched by Hitachi was the world's first camcoder to use DVD-RAM.

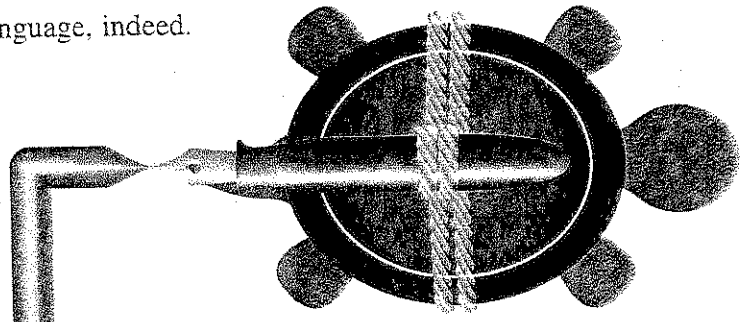
2005



Panasonic AG HVX200
Earlier High definition camera to use MiniDv tape combined with P2 digital storage..

Programming a Turtle

Imagine you could create a drawing by tying a pen to the tail of a turtle and giving the turtle directions on which path to take. The turtle would need to understand a language that contained a few simple commands, such as Forward, Left, Right, PenUp, PenDown. Now imagine a computer language with the same commands that could control a turtle on a computer screen. There is such a language: It's called LOGO, and it contains a group of graphics commands called Turtle Graphics. This language is used to teach programming to young children, but make no mistake—it is a full-featured and powerful language, indeed.

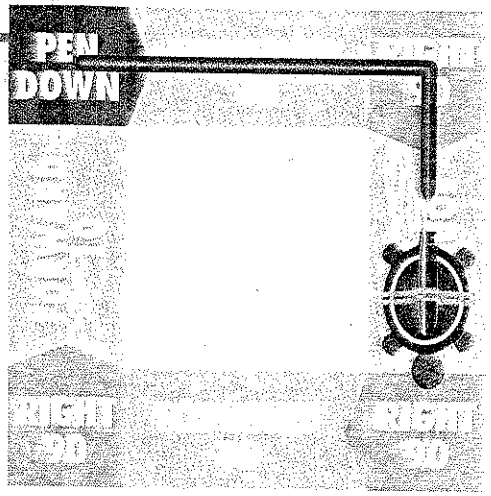


```
PenDown
Forward 10
Right 90
Forward 5
Left 90
Forward 5
```

```
Left 90
Forward 8
Right 90
Forward 5
```

```
To Box
>PenDown        >Right 90
>Forward 10     >Forward 10
>Right 90       >Right 90
>Forward 10     >PenUp
>Right 90       >End
>Forward 10     BOX
```

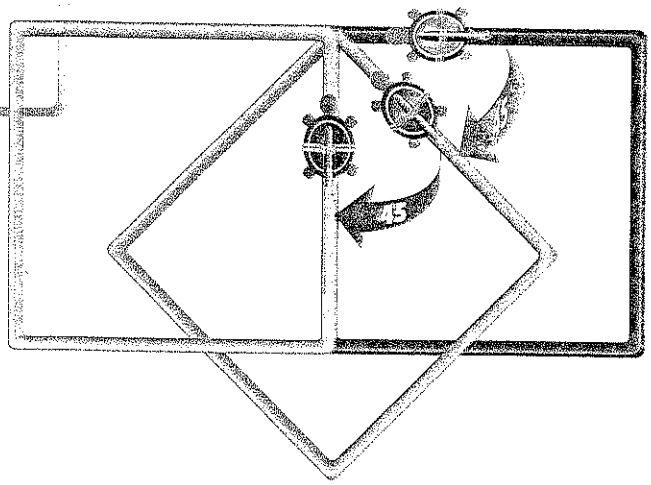
1 In LOGO, Forward moves the turtle forward, and Left and Right turn the turtle by the number of degrees specified. PenUp and PenDown lift or drop the pen. If the pen is up, you can move the turtle without drawing. You can draw a box as shown on the screen to the left.



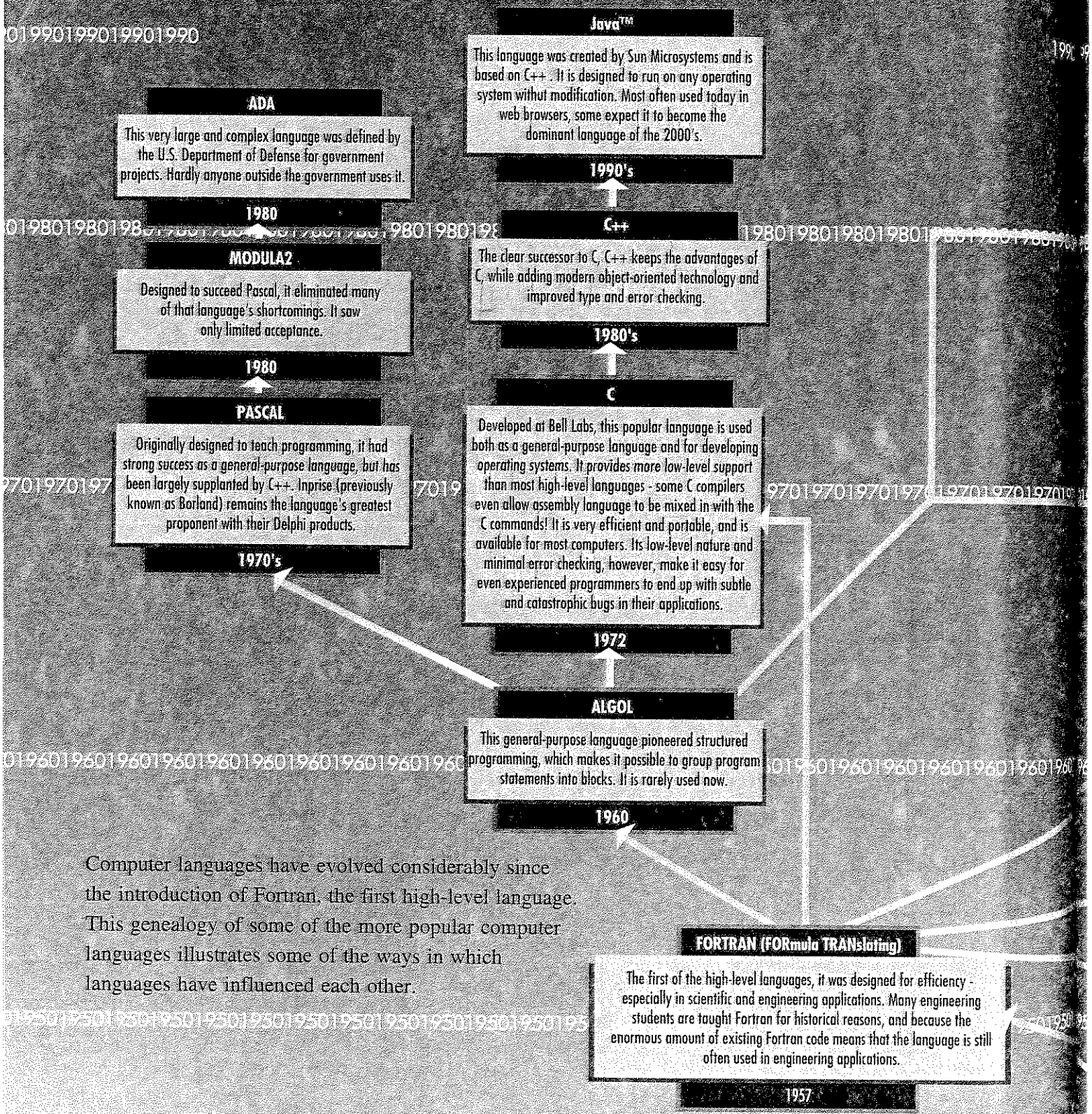
2 Why the > character? Because the first statement 'To' told LOGO to record the commands under the name *Box*. The > indicates that the command is being recorded instead of executed immediately. The End statement told LOGO to stop recording the commands. Now, any time you type *Box*, the turtle will draw a box.

```
Box
Right 45
PenColor 2
Box
Right 45
PenColor 5
Box
```

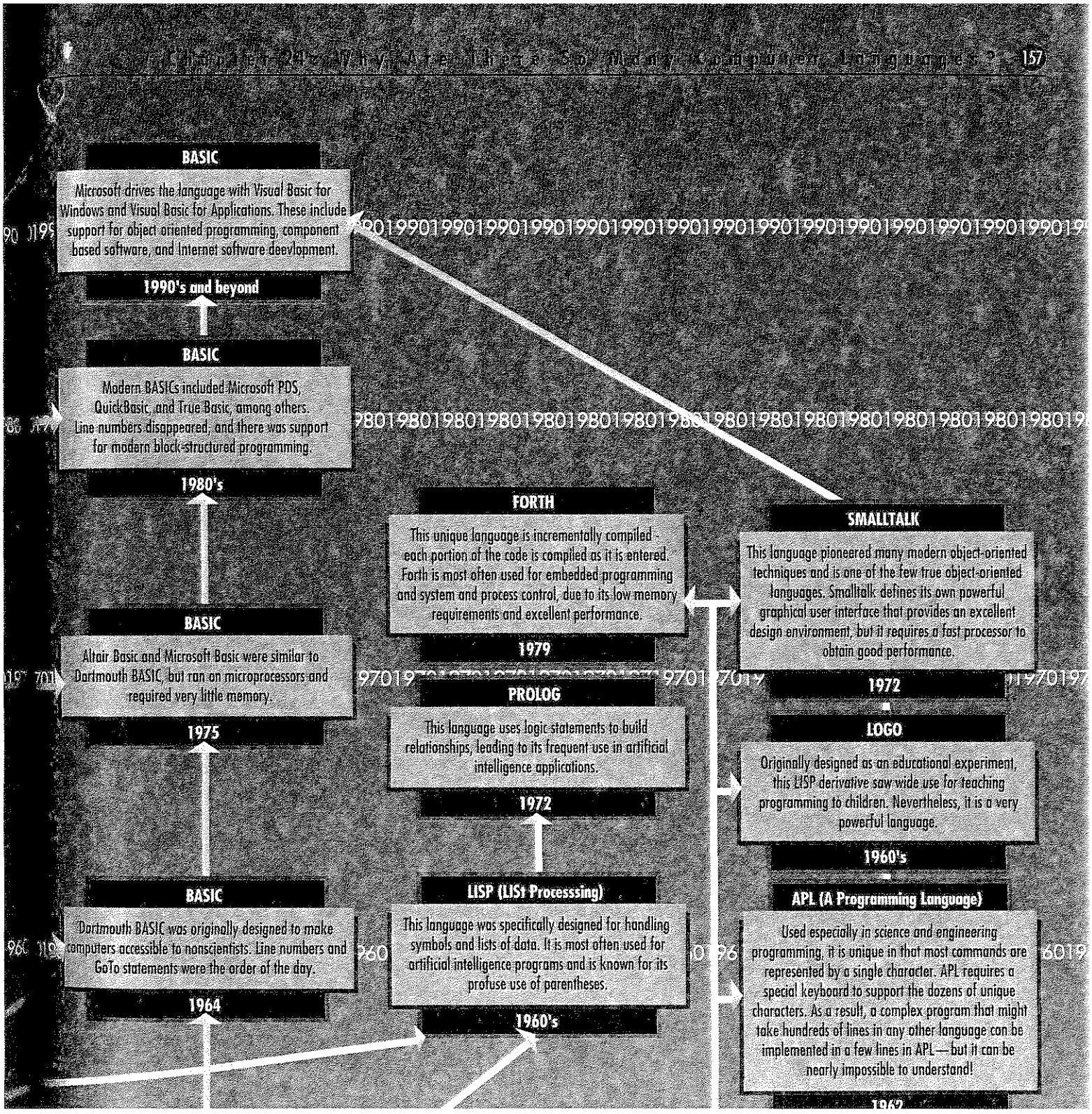
3 This is a very simple LOGO program that can be used to draw lots of boxes easily. We can use the PenColor command to change the color of the pen, as well.



The Evolution of Computer Languages



Computer languages have evolved considerably since the introduction of Fortran, the first high-level language. This genealogy of some of the more popular computer languages illustrates some of the ways in which languages have influenced each other.


BASIC
Microsoft drives the language with Visual Basic for Windows and Visual Basic for Applications. These include support for object oriented programming, component based software, and Internet software deevlopment.
1990's and beyond
BASIC
Modern BASICs included Microsoft PDS, QuickBasic, and True Basic, among others. Line numbers disappeared, and there was support for modern block-structured programming.
1980's
BASIC
Altair Basic and Microsoft Basic were similar to Dartmouth BASIC, but ran on microprocessors and required very little memory.
1975
BASIC
Dartmouth BASIC was originally designed to make computers accessible to nonscientists. Line numbers and GoTo statements were the order of the day.
1964
FORTH
This unique language is incrementally compiled - each portion of the code is compiled as it is entered. Forth is most often used for embedded programming and system and process control, due to its low memory requirements and excellent performance.
1979
PROLOG
This language uses logic statements to build relationships, leading to its frequent use in artificial intelligence applications.
1972
LISP (LISt Processsing)
This language was specifically designed for handling symbols and lists of data. It is most often used for artificial intelligence programs and is known for its profuse use of parentheses.
1960's
SMALLTALK
This language pioneered many modern object-oriented techniques and is one of the few true object-oriented languages. Smalltalk defines its own powerful graphical user interface that provides an excellent design environment, but it requires a fast processor to obtain good performance.
1972
LOGO
Originally designed as an educational experiment, this LISP derivative saw wide use for teaching programming to children. Nevertheless, it is a very powerful language.
1960's
APL (A Programming Language)
Used especially in science and engineering programming, it is unique in that most commands are represented by a single character. APL requires a special keyboard to support the dozens of unique characters. As a result, a complex program that might take hundreds of lines in any other language can be implemented in a few lines in APL—but it can be nearly impossible to understand!
1949

How the C Language Works

MANY BUSINESS PROGRAMMERS use COBOL. Engineers have traditionally programmed in Fortran. Professors teach every language under the sun (sometimes the more obscure, the better), and it seems that everyone programs in BASIC. But when it comes to serious computer scientists, odds are they are using C++. This is true whether they are writing operating systems, compilers, interpreters, games, simulations, word processors, spreadsheets, project managers or virtually anything else. However, C++ is built on the C language, and learning some C is an essential first step to learning C++ (which will be discussed further in chapter 31).

Dennis Ritchie designed C in 1972 as a tool to develop the UNIX operating system, one of the more popular workstation operating systems still in use today. The C language has two great strengths: First, it is a highly transportable language; you can write C software on one computer and quickly recompile it to run on another computer. And the C language is in a sense both a high-level and a low-level language at the same time; herein lies its other strength.

Low-level languages provide excellent performance, but require extra effort on the part of the programmer to work closely to the underlying hardware capabilities. At the low level, C lets you directly access memory locations and even influence the allocation of CPU registers for data. C supports Boolean operations such as bit shifts, which are essential to arithmetic operations and graphic operations. With most versions of C, you can call functions written in assembly language directly, enabling you to use assembly for hardware-dependent or time-critical routines.

High-level languages provide extra capability by handling details such as memory management. They also provide high-level instructions that translate into many lower-level instructions, but this takes a toll on performance. C does both. At the same time, C lets you create well-structured code, supports most numeric types, and provides a versatile set of commands. Most implementations of C also come with a large library of functions derived from the original function library created for

The C Language

A large part of the power of C is due to its support for pointers. A pointer variable in C is defined using the * character. Thus

```
char *ptr;
```

creates a variable named ptr that contains the location of a character. You can define a variable that holds a character using the declaration

```
char c;
```

You can set variable ptr to point to character c using the address operator, the & character as follows:

```
ptr = &c;
```

You can now set the value of variable c in two ways:

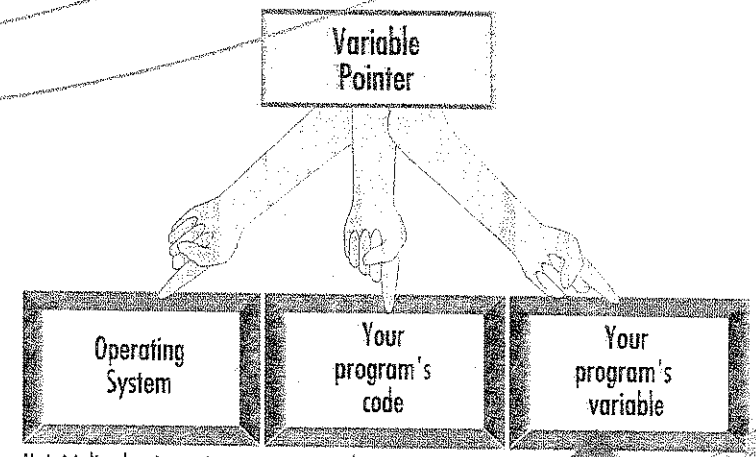
```
c = 5;
```

or

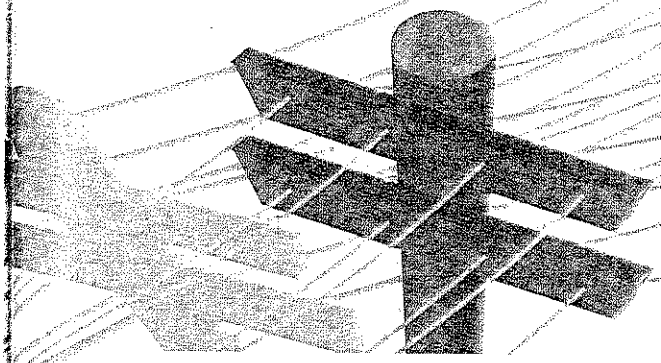
```
*ptr = 5;
```

But beware! If you forget to set variable ptr to point to a variable or another valid memory location before you try to use it, your program can crash. An uninitialized pointer can contain any value. For instance, if ptr pointed to a location in memory containing code, even operating system code, assigning data to that location could cause an incorrect or invalid instruction to be executed—a sure way to halt your system.

One of the prices programmers pay for C's high performance is in error checking. Runtime error checking by the language takes time—and C programmers would rather spend that time executing their programs. As a result, an error that would be detected and identified in another language might be ignored in C, at least until the program crashes. Unfortunately, C allows you to happily overwrite other variables, code, and on some systems, even the operating system itself. Fortunately, C compilers usually come with special programs called *debuggers* that can help track down these problems.



Uninitialized pointer, it can point anywhere



Here is a closer look at a pair of functions that can be used to convert a lowercase string to uppercase. They implement the simple algorithm of looping through each character in the string, checking if it is lowercase, and changing it to uppercase if necessary.

```
int IsCharLowerCase(char c)
{
   int result;
   if(c >= 'a' && c<='z') result = 1;
   else result = 0;
   return(result);
}

VOID MakeStringUpperCase
(char * ptrToString)
{
   char *cptr;

   cptr = ptrToString;

   while(*cptr) {
      if(IsCharLowerCase(*cptr)) {
         *cptr = *cptr - 32;
         }
      cptr = cptr+1;
      }
   return;
}
```

Functions, like variables, have types. In this case, function IsCharLowerCase is defined to return an integer. The function takes a character as a parameter, which will be named C. Notice that all C statements are in lowercase letters.

The if command in C tests a condition that you must enclose in parentheses. Because of the parentheses, you don't need a Then statement to mark the end of the condition (a requirement in BASIC). The && operator is a Boolean AND operator. In this case, the result is set to TRUE (nonzero) if c is between the letters *a* and *z* (lowercase).

Blocks in C are enclosed in brackets. A function is one way to group a block of code. Each statement within a block must end with a ; (semicolon) character.

The VOID function type defines a subroutine that has no return value. The parameter passed here is a pointer to a character. Strings in C are passed using character pointers. The pointer references the first character in the string. The string is terminated with a character with the value zero.

We create a temporary variable cptr that will be used to scan through the string one character at a time. It is initialized to point to the ptrToString parameter.

A character with the value zero marks the end of a string. Any nonzero character needs to be checked. Since the TRUE condition is defined as nonzero, we can simply place the value of the character as the conditional. This is the same as saying while(*cptr!=0). C has many such shortcuts. You'll also notice a bracket that opens a new block. Every statement within this block will be executed each time the while condition is TRUE.

We can use the * operator to refer to the character that is pointed to. The lowercase characters have an ASCII value 32 greater than their corresponding uppercase character, so the uppercase conversion is a simple subtraction.

We set the pointer to point to the next character at the end of the block. C also has a ++ operator that increments a pointer, so we could have used the command cptr++ instead of cptr= cptr+1.

This bracket marks the close of the block opened with the while statement. Notice that blocks can be placed within other blocks.

Mobile Phones = the most rapidly adopted tech. in human history

In this Text ~ Brandon's Wheel?

EVOLUTION OF THE MOBILE PHONE

1st Mobile Phone: 1973 (not commercially available)

Mobile phones originated from permanent vehicle-mounted radio systems in battery technology had not yet been refined to a portable enough size. A prototype of Motorola's DynaTAC was used to make the first phone call while walking the streets of New York in 1973. Although it took another ten years for a commercial handheld device to be released. From then on mobile phones continued to decrease in size and weight and gain in technological features. Through the end of the twentieth century, mobile phones changed from being luxury devices to a mainstream commodity - becoming the most rapidly adopted technology in human history. Now we are tethered to the rest of the world through this small device in our pockets and another person is only a few button presses away. For better or worse - the evolution of the mobile phone has changed the way we live our lives forever.

1983 MOTOROLA DYNATAC 8000X

Grey Bar = Standby Time (Time before needing direct power)

Colors: Grey, Brown
7-character LED Display

8 hours
0.5 hours

Cost: $4,000
World's 1st Commercially-Available Mobile Phone

Contains address book, currency calculator
8-character LED Display

1989 MOTOROLA MICROTAC 9800X

9 hours
0.75 hours

First flip Phone
Bottom + Top don't work - Microphone + antennae are in base.

Pop-out antennae

memory for up to 250 names + numbers
4x13 character LED Display

1993 NOKIA 2110

30 hours
2.7 hours

First use of context-sensitive buttons (changes depending on mode)
Vibration added (instead of just audio ringer)

1998 NOKIA 5510

180 hours
3.2 hours

Stores 30 ringtones
Monochrome 5-line LCD with changeable font size

Can play "Snake" on phone (1st game)
Comes with 2 other games - memory & Logic
Can change faceplate of phone.

2000 NOKIA 3310

260 hours
4.5 hours

3rd-party accessories are compatible
Uses predictive text messaging.

2004 MOTOROLA RAZR V3

mP3 + 24-channel polyphonic ringtones
256K Color Display

280 hours
7 hours

15mm thick (thinnest on market at time)

camera (640 x 480 pixel)

2007 APPLE IPHONE

Camera (2M pixel)

16M color Display
Wifi, Bluetooth, + GPS Support
Touchscreen

250 hours
6 hours

White Bar = Talk Time (Time before needing direct power)

C++

C++ is an object-oriented programming language used to develop software, video games, and more.

C++ was first developed in 1983, as an enhancement to the C programming language.

C++ files have a .c++ extension.

C++ can develop:

- Apps for Windows and Linux
- Video games
- Mobile apps

C++ Jobs

Average Salary: $90,000

Job Count: 31,893

Top Cities:
- New York
- Seattle
- San Francisco

Top Employers:
- Amazon
- CyberCoders
- Microsoft

Tidbits of Wisdom

C++ is highly portable across multiple devices.

C++ is supported by Apple, Android, Windows Phone, and Blackberry.

OBJECTIVE C

used by Apple developers.

Objective C was first developed in the 1990s.

Objective C files have an .m extension.

Objective C can:

Develop mobile apps for iOS

iOS

Develop applications for OS X

Objective C Jobs

Average Salary: $70,000

Job Count: 18,849

Top Cities:
- New York
- San Francisco
- Chicago

Top Employers:
- Regis Corporation
- SmartStyle
- SmartCuts

Tidbits of Wisdom

Though considered the correct language for iOS development, it cannot be used for other platforms such as Android.

PYTHON™

This is a server-side interpreted, open-source, non-compiled, scripting language. It can be used on its own, or as part of another framework, like Django.

Python can:

- Build websites
- Provide database access
- python™
- Build desktop graphic user interfaces (GUIs)
- Build software and games

Python Jobs

Average Salary: $83,000

Job Count: 19,627

Top Cities:
- Mountain View
- San Francisco
- New York

Top Employers:
- Amazon
- Intel®
- Dell

Tidbits of Wisdom

NASA's shuttle support contractor, United Space Alliance (USA) uses Python.

JAVASCRIPT

This is a client-side scripting language. It is embedded in most web browsers.

Developed in 1995 by Netscape

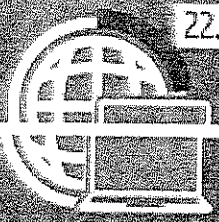
Netscape®

Used in website:


Advertising


Analytics


Widgets



22.9%

jQuery is the most detected JavaScript library in use on the web, used for:

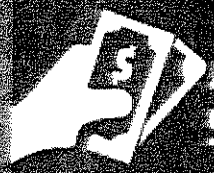
22.9%

of the top million websites on the Internet.

It provides features and functions to make:

- JavaScript browser agnostic
- An easier development process

JavaScript Jobs



Average Salary:
$88,000

Job Count:
43,189

Top Cities:
New York
San Francisco
Washington D.C.

Top Employers:
Amazon
Microsoft
IBM

Tidbits of Wisdom

You can use JavaScript to:


Check username availability as user enters it, preventing the need to reload the page


Build an autocomplete function on your website


Fix layout issues


Enhance HTML text boxes so users have a combination of presets and the ability to enter their own text

JAVA®

This is a server-side interpreted compiled language, using a virtual machine. It is not JavaScript, and is not related to it.

Java was developed in 1995, and is one of the oldest programming languages on the web.

Java lets you:


Play online games


Upload pictures


Take virtual tours


Use interactive maps

Java Jobs

Average Salary:
$95,000

Job Count:
66,485

Top Cities:
New York
Washington D.C.
San Jose

Top Employers:
Amazon
IBM
eBay

Tidbits of Wisdom

Users can disable Java on their machines.

Java is the basis of Android

Slow to change, so it's easier to keep up with