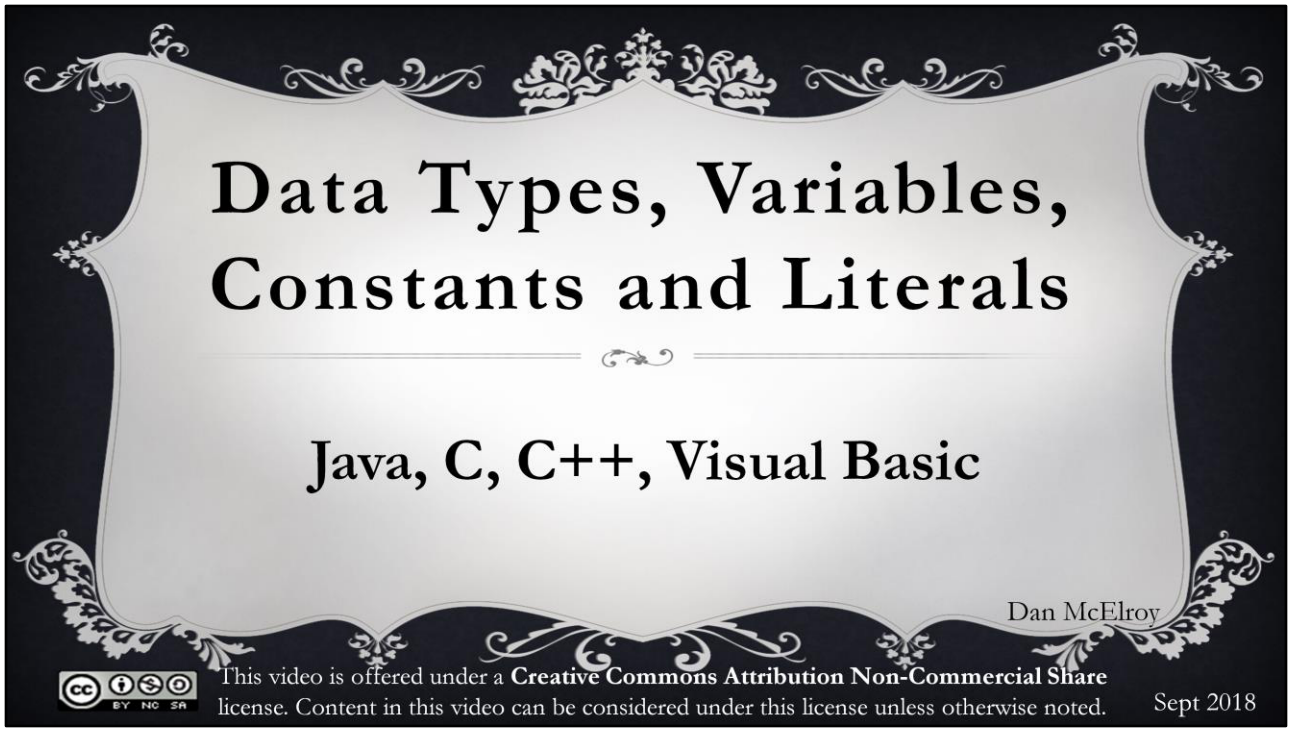


Hello programmers.
Welcome to the discussion on Data Types, Variables, Constants and Literals.

List of Topics

- Primitive data types
- Literals
- Variables
- Constants
- Magic Numbers

In addition to these topics, we will also be discussing "Magic Numbers"

Primitive Data Types

- Numeric - integer and real numbers
- Text (String is not a primitive data type, but it is covered in this discussion.
- Boolean and null

The following data types are referred to as primitive because they are of the most basic and lowest level of data types. The can be grouped into three basic categories:
Numeric data types - integer and real numbers
Text data types (In C++, Java and Visual Basic, string is actually an object instead of a primitive data type, but it is covered in this discussion anyway.
The boolean data type can only hold a true or false. The null data type is used to identify when the data type is not known

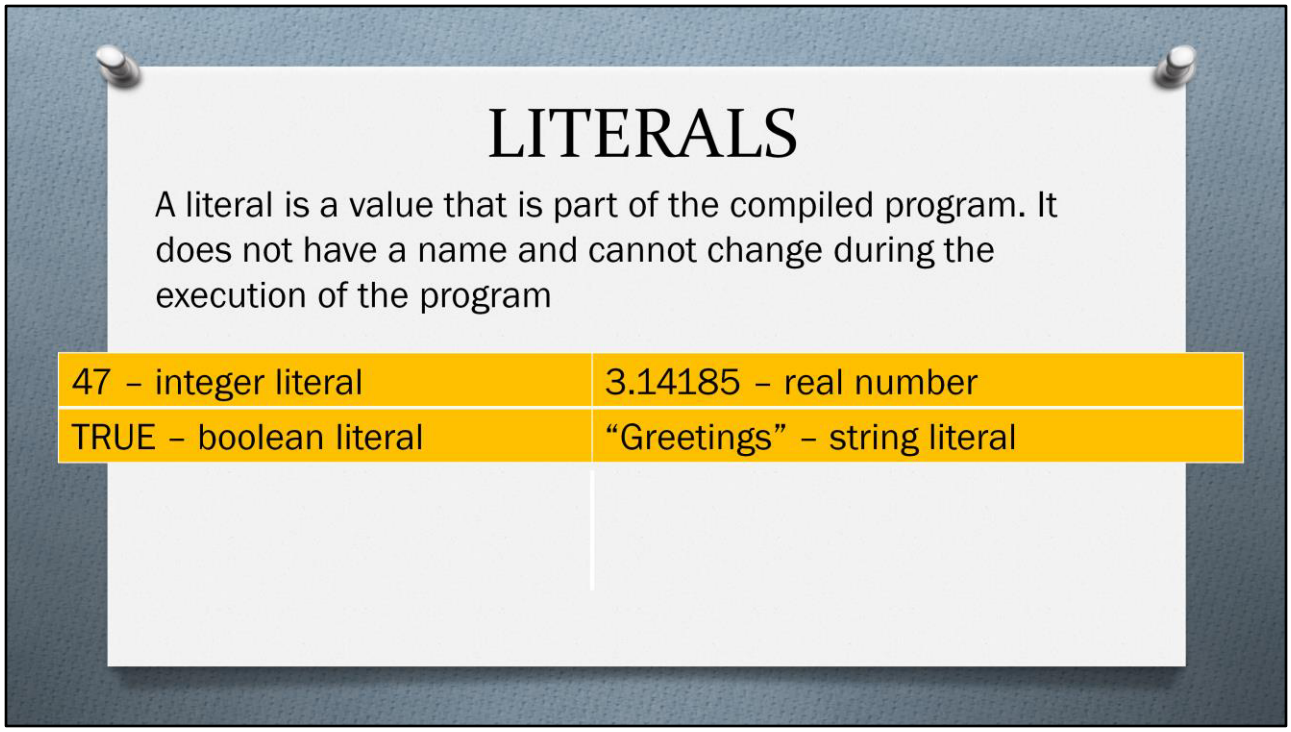
Literals
Variables
Constants

Data types can also be part of the definition for literals, variables and constants.

A literal is a value that is part of the compiled program. It does not have a name and cannot change during the execution of the program.

A variable represents a storage location in RAM that has a data type and a name and can have a value placed in it during the execution of the program.

Named Constants are similar to both literals and variables in that they cannot be changed, but constants can have a data type.

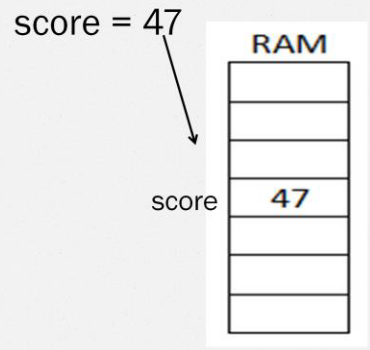


A literal is a value that is part of the compiled program. It does not have a name like variables or constants and literals cannot change during the execution of the program.

Here are some examples of literals:
47 – is an integer literal with no decimal point or digits past the decimal
3.14185 – real number because it has digits past the decimal
TRUE – boolean literals can either be TRUE or FALSE
“Greetings” – string literal. It is enclosed entirely within double-quotes

VARIABLES

- Declared with a data type and a name
- Memory is allocated to hold the data
- Variables can hold data



A variable represents a storage location in RAM that has a data type and can have a value placed in it during the execution of the program.
Variables have:
Declared with a data type and a name
Memory is allocated to hold the data
Variables can hold data, and the data within a variable can be changed by the program.
Here I see a variable that resides in memory. Its **name** is score and the **contents** of the variable are set to 47

Initializing a Variable

Java C C++	VisualBasic

int score = 0;
-or-
int score;
score = 0;
	Dim score as Integer = 0
	-or-
	Dim score as Integer
	score = 0

A value **MUST** be placed in the variable before it can be used as part of a computation.
A value can be placed into a variable when it is first declared (initialized), or the value can be placed in the variable later in the program.

A variable is declared the same way in Java, C or C++. It starts with a data type followed by the name the programmer wants to give the variable. A variable can be initialized to a value at the same time the variable is declared, or the variable can just be declared and initialized later in the program.

DATA TYPES

- Numeric (integer and floating point)
- Character and Strings (strings are treated as a data type here)
- Boolean (True or False)

- Structured Records – discussed much later
- User Defined Data Types – discussed much later

It is also possible for the programmer to define new data types, which are usually made from a collection of existing data types.

Enumerated Data Type

An enumeration has a name and a set of members. Each member represents a constant. Enum is short for enumerated data type. Enums are an easy way to define a list of constants. For example, an enum could be used to define a numeric value for each month instead of defining them individually.

Jan = 1
Feb = 2
Mar = 3
etc.

The enumerated data type was not listed yet. An enumeration has a name and a set of members. Each member represents a constant. Enum is short for enumerated data type. Enums are an easy way to define a list of constants. For example, an enum could be used to define a numeric value for each month instead of defining them individually.

Jan = 1
Feb = 2
Mar = 3
etc.

Sample Java Program

```
package javaenum;

public class JavaEnum {
   // enum defintitions go outside of main when using Java
   enum TrafficEnum { RED, YELLOW, GREEN }

   public static void main(String[] args) {
       TrafficEnum SouthSt;    // declare a variable of type TrafficEnum
       SouthSt = TrafficEnum.GREEN;  // Assign a value to SouthSt

       System.out.println ("The light is " + SouthSt);
       System.out.println ("The numeric value is " + SouthSt.ordinal());
   } // end of main
} // end of the class definition
```

Here is a sample program in Java that uses the enumerated data type. I am creating a new enumerated data type and calling it TrafficEnum. It has only three possible values, RED, YELLOW and GREEN. I can use these values later in my program. In Java, the enum declaration must either be in its own file or be placed outside of the definition for main. I see here that a variable named SouthSt is being declared using the data type TrafficEnum. It can be initialized to any of the three possible values, RED, YELLOW or GREEN. In this program it is being set to GREEN. The first println statement will output, "The light is GREEN". The second println statement will output, "The numeric value is 2" because in Java the enum definitions start at 0. RED would be a 0, YELLOW would be 1 and GREEN becomes a 2.

Sample C++ Program

```
#include <iostream>
using namespace std;

int main()
{
   enum TrafficEnum { RED, YELLOW, GREEN };
   typedef TrafficEnum trafficLight;

   trafficLight SouthSt;       // create a new data type
   SouthSt = GREEN;            // Assign a value to SouthSt

   if (SouthSt == GREEN)
      cout << "The light is GREEN" << endl;
   cout << "The numeric value is " << SouthSt << endl;
   return 0;
} // end of main
```

Using the enum is fairly similar when using C or C++. The enum statement defines the three possible conditions, but the enum by itself does not create the new data type. The **typedef** statement is used along with the enum definition to actually create the data type. Although not shown on this slide, it is possible to create the data type in C or C++ at the same time as defining the enum.

typedef enum TrafficEnum {RED, YELLOW, GREEN} traffic_light;

Since SouthSt is defined by the enum, the if statement can compare SouthSt to one of its predefined values. We can also find the numeric value of the individual definitions within the enum.

One thing that C and C++ can do when defining the enum is to initialize the values within the enum. For example:

enum MonthsEnum {JAN=1, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, DEC} month;

In this case, MAR would be a 3 but if JAN was not set to 1, then starting from 0, MAR would be a 2.

Sample C Program

```
#include <stdio.h>
int main()
{
   enum TrafficEnum { RED, YELLOW, GREEN };
   typedef TrafficEnum trafficLight;
   trafficLight SouthSt;        // create a new data type
   SouthSt = GREEN;             // Assign a value to SouthSt
   if (SouthSt == GREEN)
       printf ("The light is GREEN\n);
   printf (" "The numeric value is %d\n", SouthSt);
   return 0;
} // end of main
```

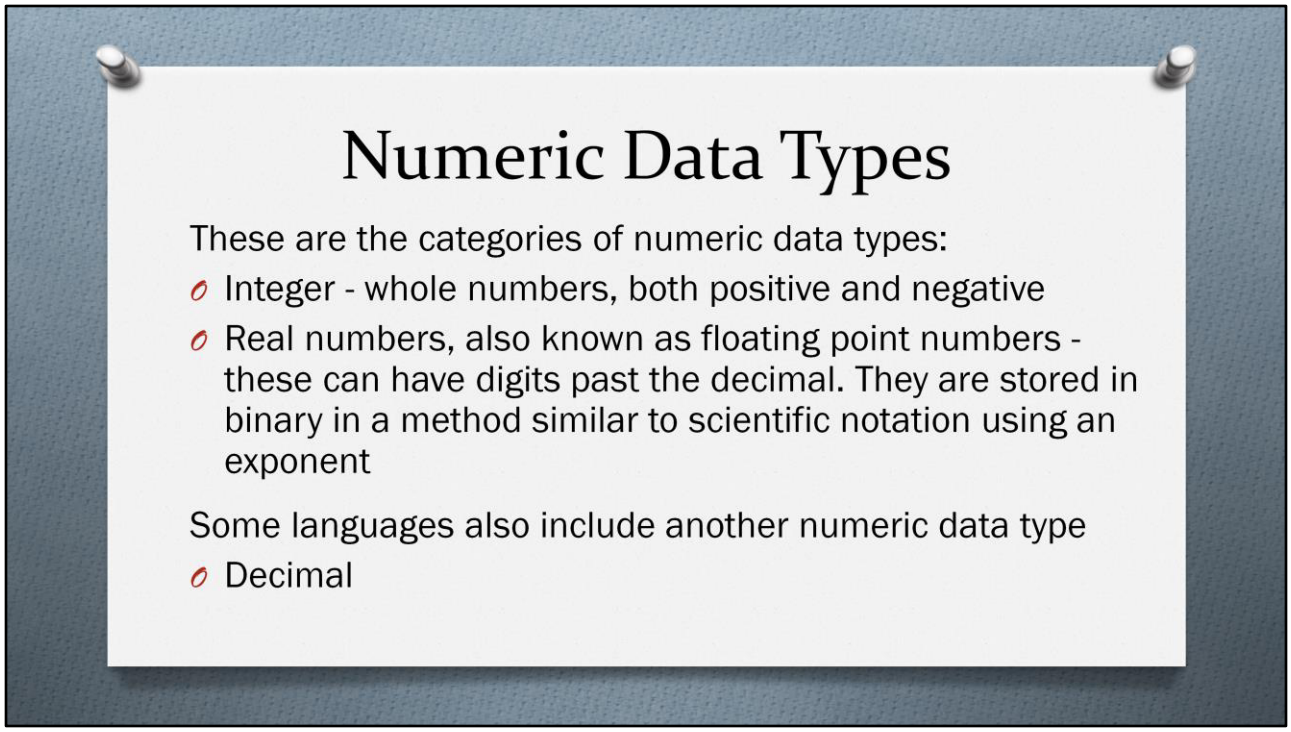
I am providing a sample C program. It is identical to the C++ program except that the printf statement is used instead of a cout.

Sample Visual Basic Program

```
Imports System
Module Program
   Enum TrafficEnum
      RED
      YELLOW
      GREEN
   End Enum
   Sub Main(args As String())
      Dim SouthSt As TrafficEnum   ' Define a variable
      SouthSt = TrafficEnum.RED    ' Assign a value to the variable
      Console.WriteLine("The light is " + SouthSt.ToString)
   End Sub
End Module
```

Even Visual Basic has the ability to define 'constants' using the enum. Since VB does not use the open and close curly-braces for BEGIN and END, we need to use the words End Enum to identify the end of the definition for the enum.

Similar to Java, when setting SouthSt to RED, we also need to identify where RED came from, such as:
SouthSt = TrafficEnum.RED



Back to the Numeric Data Types. Integers can only hold whole numbers, both positive and negative. The words **Real** and **Floating Point** can be used interchangeably. Some programming languages call floating point numbers **Real** while others use some form of **Floating Point**.

When to use an Integer?
When to use a Float?

A number like 3.14185 has digits past the decimal point and can not be stored into an integer. Only whole numbers can be stored in integers.

Integers can be converted from decimal used by humans to binary used by computers with no loss in precision. Although float is now very accurate, occasionally small errors creep in at many digits past the decimal point.

Whenever possible, use integers instead of float.

Here is a BIG question. When creating numeric variables, should an integer or a float be used?
A number like 3.14185 has digits past the decimal point and can not be store into an integer. Only whole numbers can be stored in integers.

Integers can be converted from decimal used by humans to binary used by computers with no loss in precision. Although float is now very accurate, occasionally small errors creep in at many digits past the decimal point.

Whenever possible, use integers instead of float.

INTEGERS

Integers are whole numbers. The values of integers can be positive, negative or zero.

Examples:

79 positive number
-85 negative number
0 zero

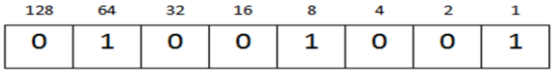
Integers are whole numbers. The values of integers can be positive, negative or zero.

Here are some examples of integers

79 is a positive integer
-85 is a negative integer
0 is an integer zero

Binary Representation of an Integer

73 =



64 + 8 + 1 = 73

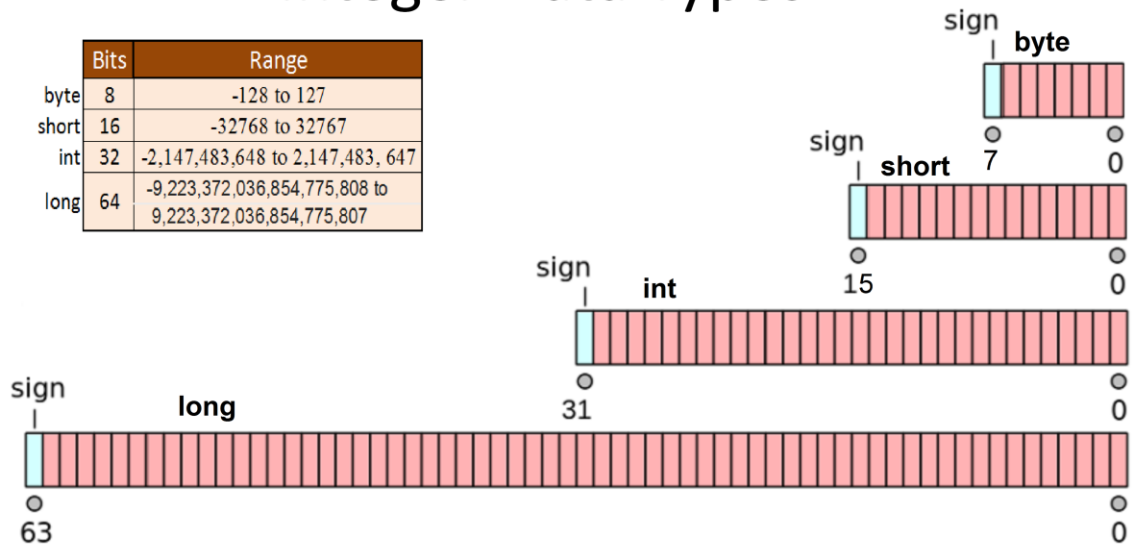
For signed numbers, the most significant bit is used as a **sign bit** to indicate a positive or negative number leaving one less bit for the value. Eight bits, including the sign bit can be used to store a value from -128 to +127.

For signed numbers, the most significant bit is used as a **sign bit** to indicate a positive or negative number leaving one less bit for the value. A zero in the sign-bit position indicates that the number is positive (or possibly a zero) and a one in the sign-bit position indicates a negative number.

In this example, the number 73 is stored in a **byte** which is only 8 bits. There is a 1 in bit values for 64, 8 and 1. Converting the binary value of 01001001 to decimal gives 73. Eight bits, including the sign bit can be used to store a value from -128 to +127. More bits are needed to store a number greater than 127.

Integer Data Types

	Bits	Range
byte	8	-128 to 127
short	16	-32768 to 32767
int	32	-2,147,483,648 to 2,147,483, 647
long	64	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807



The **short** data type uses 16 bits and can store a number from -32,768 to +32,767
The **int** data type can store a number in excess of +/- 2 billion
The **long** data type can store a humongously large number in excess of +/- 9 quintillion
A quintillion is a 1 followed by 18 zeros.

Have you heard of the counting down song, "100 bottles of beer on the wall"?

(sing) A quintillion bottles of beer on the wall, a quintillion bottles of beer
Take one down, pass it around, 999 quadrillion, 999 trillion, 999 billion, 999 million, 999 thousand, 999 bottles of beer on the wall.

999 quadrillion, 999 trillion, 999 billion, ---- Forget it. I give up !

FLOATING POINT NUMBERS

Floating point numbers can contain digits past the decimal point. Floating point numbers are called REAL NUMBERS in the math department and in some computer languages

Examples:

7.0	**0.0**
3.1418579	**0.00000000578 positive**
-85.3357	**-0.00000000578 negative**

Floating point numbers can contain digits past the decimal point. Floating point numbers are called REAL NUMBERS in the math department and in some computer languages
Examples:

7.0
3.1418579
-85.3357
0.0
0.00000000578 is a positive value less than one
-0.00000000578 is a negative value less than one

Scientific Notation on the Computer

5167.895 = 5.167895×10^3 = 5.167895e3 (e3 is 10^3)
0.000542 = $5.42e\text{-}4 \times 10^{-4}$ = 5.42e-4 (e-4 is 10^{-4})
-0.000004 = -4.0×10^{-6} = -4.0e-6 (e-6 is 10^{-6})

Here are a couple of hints:
The value of the exponent indicates the number of places to move the decimal point, either to the left or the right.

A number between one and zero has a negative exponent.
A negative number has a negative **base**.

18)

You may have been exposed to scientific notation in a science class. Scientific notation displays numbers with only one significant digit before the decimal point and all of the rest of the digits are shown past the decimal point. This is called the base. The number is then multiplied by some power of 10 which establishes the actual **size** of the number. Computer programs and **even** many calculators display the power of 10 using the letter 'e' followed by a number. For example, e3 means 10 raised to the 3rd power.

Floating point numbers use data types float or double. Floating point numbers can be displayed using the fixed point notation (which looks like a normal number) or in scientific notation using **base** and **exponent**. Here are three examples:

5167.895 is the same as 5.167895×10^3 and displayed as 5.167895e3
0.000542 is the same as $5.42e\text{-}4 \times 10^{-4}$ and displayed as 5.42e-4
-0.000004 is the same as -4.0e-6 and displayed as -4.0e-6

Here are a couple of hints:
The value of the exponent indicates the number of places to move the decimal point, either left or right.
A number between one and zero has a negative exponent.

A negative number has a negative **base**.

Float vs. Double

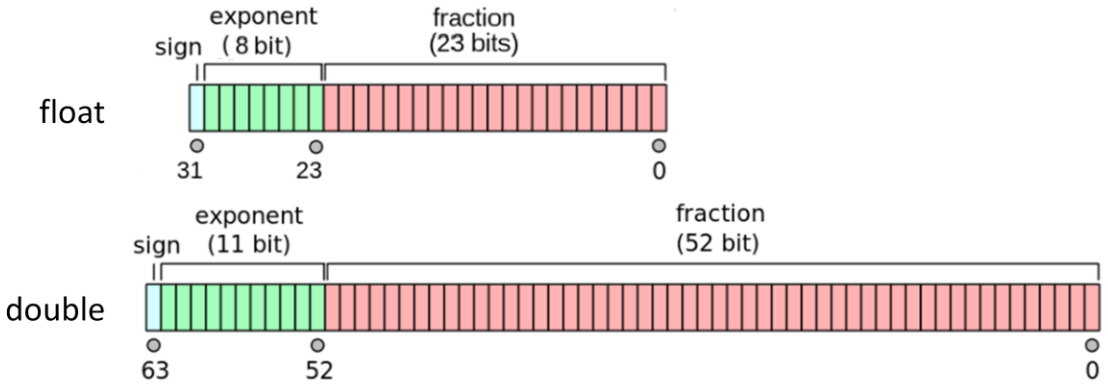
The original IBM PC used the Intel 8088, a sixteen bit processor with an 8-bit data bus. Unless you installed a separate 8087 math coprocessor, floating point calculations were done in software. In either case, the computer was very slow and not very accurate when processing floating point numbers. The data type float used 32 bits and was used to declare floating point variables.

As processors became faster and cheaper, the math co-processor was included with the main processor. We now use double precision floating point numbers with the data type double which uses 64 bits.

Double means double-precision floating point data type.

Floating Point Numbers in Binary

	Smallest positive number	Largest positive number
float	$1.40239846 \times 10^{-45}$	$3.40282347 \times 10^{38}$
double	$4.9406564584124654 \times 10^{-324}$	$1.7976931348623157 \times 10^{308}$



You will probably never need to get down to the bit level for floating point numbers, but it is good background information to know a little about how they are stored in memory. In a manner similar to how numbers are held in scientific notation, the **float** and **double** data types store numbers with their exponent and fractional parts, but in **binary** instead of decimal. The **float** data type stores a floating point number using 32 bits and the **double** data type uses 64 bits to store a floating point number. When looking at difference between the range and precision for the two different formats it is easy to see why the **double** data type is used most often with the modern computers.

What is the **Decimal** Data Type?

The **Decimal** data type is available on Microsoft VisualBasic and C# (C-sharp) but **not in C++.** **Decimal** also stores floating point numbers, but internally the number is stored as a collection of individual decimal digits and a decimal point location instead of converting the number to binary similar to the scientific notation. The **Decimal** data type is good for financial applications.

Floating Point Rounding Errors

	Internal
Interest Payment #1	3.596
+ Interest Payment #2	5.598
Total	9.194

Floating point number could have some rounding errors when being displayed. Here is an example of computing interest on two separate accounts for a customer of a bank. The interest for the two accounts might be computed at 3.596 and 5.598. When they are added together, the total is 9.194.

Floating Point Rounding Errors

	Internal	Display
Interest Payment #1	3.596	3.60
+ Interest Payment #2	5.598	5.60
Total	9.194	9.19

But when the numbers were displayed rounded to the nearest penny, the second column would be displayed. It would appear that the bank can't do simple addition and the customer is being cheated out of a penny even though the internal computations were correct. Suggestion: In Visual Basic or C# use the **Decimal** data type instead of Float or Double for currency calculations.

CONVERTING BETWEEN DATA TYPES

Since an integer is a whole number, conversion to a double is easy.

Converting from a float data type (double) to integer poses some problems. What should happen with the digits past the decimal point?

Since an integer is a whole number, conversion to a double is easy.

Converting from a float data type (double) to integer poses some problems. What should happen with the digits past the decimal point?

Converting from Integer to Double

An integer can be easily converted to a double.

Java / C / C++

```
int value1 = 75;
double value2 = value1;
```

VisualBasic

```
Dim value1 As Integer = 75
Dim value2 As Double = value1
```

The floating point result is 75.0

An integer can be easily converted to a double.

In Java / C / C++

```
int value1 = 75;
double value2 = value1;
```

In VisualBasic

```
Dim value1 As Integer = 75
Dim value2 As Double = value1
```

The floating point result is 75.0

Converting from Double to Integer

What happens to the digits past the decimal when converting to integer?

C / C++

```
double value1 = 75.8;
int value2 = value1;
```

C and C++ **round down**, also known as truncated. The result for value2 is 75. The digits past the decimal point are lost.

value2 is equal to 75 in C and C++

Many of the C and C++ compilers will report this as a warning, "Possible Loss of Precision". It is just important to look at the warnings as the errors when your program is compiled.

Converting from Double to Integer

What happens to the digits past the decimal when converting to integer?

Java

```
double value1 = 75.8;
int value2 = (int)value1;
```

Java is strongly typed and will create an error if you try to convert from a double to an integer without specifically telling Java that this is what you want to do by using the type cast (int) in front of the double value.

Converting from Double to Integer

What happens to the digits past the decimal when converting to integer?

VisualBasic

```
Dim value1 As Double = 75.8
Dim value2 As Integer = value1
```

At the top of the program you can enter the line OPTION STRICT ON
to force the compiler to generate an error instead of doing an automatic from Double to Integer.

Type Casting and Explicit Conversions

Java / C / C++

```
double value1 = 75.8;
// type casting
int value2 = (int)value1;
// the value is rounded down
```

VisualBasic

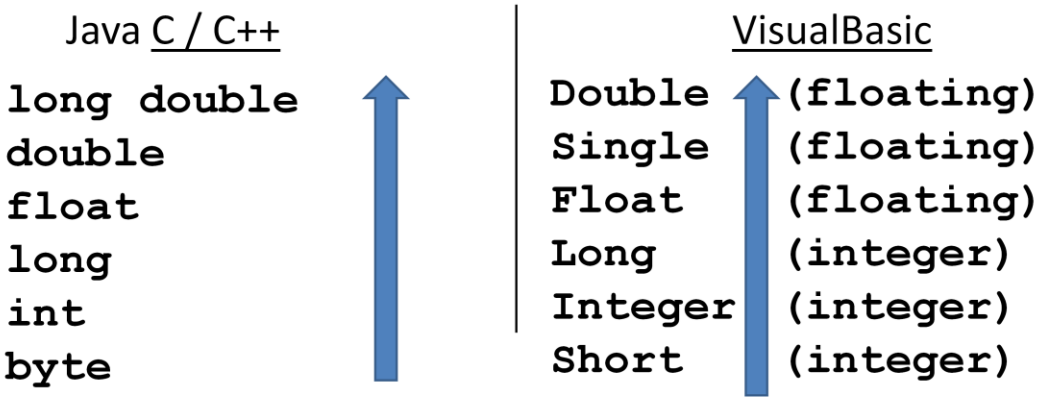
```
Dim value1 As Double = 75.8
int value2 = int(value1);
Dim value2 as Integer
value2 = Convert.toInt32(value1)

' The converted value is rounded up

   ' or
 value2 = CInt(value1)
```

The best and safest way to convert from double to integer is to call a conversion routine or in C or C++ do a type cast. Then you have defined what you want to happen instead of just letting the compiler do what it thinks you may want.

Visual Basic has two different ways to convert a floating point value into an integer. There is the **toInt32** function method and there is the **CInt** function method.

Automatic Promotions

Data types are automatically promoted up but not down



Data types can be automatically 'promoted' from a lower cast to an upper cast, but not automatically demoted to a lower cast. For example, an **int** can be promoted to a **double** but not the other way around.

CHARACTERS AND STRINGS

- A character data type can hold a single character. The character can be alpha such as (A-Z, a-z), digit (0-9), special character (!@#$%^&,. etc.), or even a non-printable control character.
- A string will hold a collection of characters.

NOTE: "123" is a character string, not a number. There are conversion routines that can convert a string of digits into a numeric value.

The character data type is typically called **char**. It can hold a single character. The character can be alpha such as (A-Z, a-z), digit (0-9), special character (!@#$%^&,. etc.), or even a non-printable control character.

In C and C++ the character data type can be treated like an integer when doing math operations on it. This is not true for Java and Visual Basic. For these languages, a character must be specifically converted to an integer, math can then be performed and then it can be transformed back into a character.

It is important to remember that a string of character that contains only digits is still a string of characters, not a number. There are conversion routines that can be used to convert a string of digits into a number.

Characters

The CHAR data type holds only a single character.

Java / C / C++

```
char grade;
grade = 'A';
grade = 'B';
```

Here is how to create a character variable in Java, C and C++. The **char** data type is used. The definition for a character literal uses single quotes. Double quotes are used for a string.

Characters

The CHAR data type holds only a single character.

VisualBasic

```
Dim grade as Char
grade = "A"c
grade = "B"c
```

VisualBasic uses a double quote followed by a small c to define a character literal. The double quote without the c is used to define a string.

Character Codes

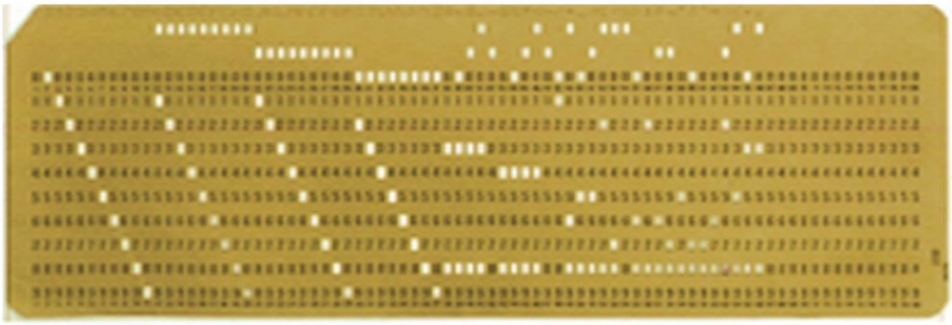
Characters are stored in memory using a binary code defined by the ASCII, Unicode or EBCDIC character sets.

ASCII stands for American Standard Code for Information Interchange. It is based on the seven-bit pattern punched tape of the old Teletype machine. The Teletype was used as the console on most computers until video display terminals were developed.

Character Codes

This brings back memories of the good old days. Of punched cards and paper tape. We had to walk to and from the computer center, in the snow, up hill both ways, feed the punched cards or paper tape in the machines, wait and wait for them to be read. You might wait a couple of hours or even a day before the data center had your printout ready for you to look for a bug in the program. Here I am reminiscing about the good old days. Maybe the good old days weren't so good after all. Actually, I would never want to go back. So back to the present :)

Character Codes



EBCDIC stands for Extended Binary Coded Decimal Interchange Code. It was based on IBM punched cards and is still used on many IBM mainframe computers. However, it is rarely used anywhere else.

ASCII Characters

The ASCII chart lists each of the codes in hexadecimal and what the code represents.

The first 32 codes are control codes. The important ones to know are BEL, BS, LF, CR and ESC.

It is important to note that the upper-case letters (A-Z) occur before the lower-case letters (a-z). Your program may need to convert the inputs from the user or disk to upper-case before comparing text when sorting data.

HEX	CHAR	HEX	CHAR	HEX	CHAR	HEX	CHAR
00	NUL	20	space	40	@	60	`
01	SOH	21	!	41	A	61	a
02	STX	22	"	42	B	62	b
03	ETX	23	#	43	C	63	c
04	EOT	24	$	44	D	64	d
05	ENQ	25	%	45	E	65	e
06	ACK	26	&	46	F	66	f
07	BEL	27	'	47	G	67	g
08	BS	28	(	48	H	68	h
09	HT	29	)	49	I	69	i
0A	LF	2A	*	4A	J	6A	j
0B	VT	2B	+	4B	K	6B	k
0C	FF	2C	,	4C	L	6C	l
0D	CR	2D	-	4D	M	6D	m
0E	SO	2E	.	4E	N	6E	n
0F	SI	2F	/	4F	O	6F	o
10	DLE	30	0	50	P	70	p
11	DC1	31	1	51	Q	71	q
12	DC2	32	2	52	R	72	r
13	DC3	33	3	53	S	73	s
14	DC4	34	4	54	T	74	t
15	NAK	35	5	55	U	75	u
16	SYN	36	6	56	V	76	v
17	ETB	37	7	57	W	77	w
18	CAN	38	8	58	X	78	x
19	EM	39	9	59	Y	79	y
1A	SUB	3A	:	5A	Z	7A	z
1B	ESC	3B	;	5B	[	7B	{
1C	FS	3C	<	5C	\	7C	|
1D	GS	3D	=	5D	]	7D	}
1E	RS	3E	>	5E	^	7E	~
1F	US	3F	?	5F		7F	DEL

The ASCII chart lists each of the codes in hexadecimal and what the code represents.

The first 32 codes are control codes. The important ones to know are BEL, BS, LF, CR and ESC.

It is important to note that the upper-case letters (A-Z) occur before the lower-case letters (a-z). Your program may need to convert the inputs from the user or disk to upper-case before comparing text when sorting data.

For example, the hexadecimal value for the letter 'S' is 53, or in binary 01010011 and the lower-case 's' is 73 or in binary 01110011.

Unicode

ከእንግሊዝኛ ሌላ ቋንቋ የሚናገሩ ከሆነ ነጻ የቋንቋ ድጋፍ አገልግሎቶችን ማግኘት ይቻላል። የእኛን ድረ-ገጽ ይጎብኙ ወይም በእርስዎ የአባልነት መታወቂያ ካርድ ላይ ያለውን ስልክ ቁጥር በመጠቀም ይደውሉ። (Amharic)

Եթե խոսում եք անգլերենից բացի մեկ այլ լեզվով, ապա Ձեզ համար հասանելի են լեզվական աջակցման անվճար ծառայություններ: Այցելեք մեր վեբ կայքը կամ զանգահարեք Ձեր անդամի նույնականացման քարտի վրա նշված հեռախոսահամարով: (Armenian)

যদি আপনি ইংরেজী ব্যতীত অন্য কোনো ভাষায় কথা বলেন তাহলে বিনামূল্যের দোভাষীর পরিষেবা উপলব্ধ আছে। আমাদের ওয়েবসাইট দেখুন এবং আপনার সদস্য পরিচয়পত্রে থাকা ফোন নম্বরে ফোন করুন। (Bengali)

Yoo afaan Ingiilifa allati affan birraa dubbattan tajaajili garggarsa afaani(qooqqa) biliissan niarggama. Kannafu websitti keenya illala hookan telefoona waarraqa miseensa irra jirran bilbilla. (Cushite-Oromo)

បើអ្នកនិយាយភាសាផ្សេងក្រៅពីភាសាអង់គ្លេស សេវាកម្មជំនួយផ្នែកភាសាមានផ្តល់ជូនអ្នកដោយឥតគិតថ្លៃ។ សូមចូលមើលគេហទំព័ររបស់យើង ឬហៅទៅកាន់លេខទូរស័ព្ទដែលមាននៅលើប័ណ្ណសម្គាល់សមាជិករបស់អ្នក។ (Khmer)

Ako govorite neki jezik koji nije engleski, dostupne su besplatne jezičke usluge. Posetite našu internet stranicu ili nazovite broj telefona na vašoj članskoj identifikacijskoj kartici. (Serbo-Croatian)

Since ASCII used only 7 bits, there were only 128 combinations of characters that could be encoded. This worked fine for the early computers built and used in America but is not sufficient to cover the many different character sets used worldwide. The UTF-16 Unicode character set uses 16 bits and has the ability to have over 60,000 different characters. The first 128 characters in Unicode are taken from the ASCII character set.

Java/C/C++ Escape Sequences

The following character sequences start with the backslash character and are used to represent control codes and other printable codes.

```
\'   (single quote)
\"   (double quote)
\\   (backslash character)
```

You will need these escape sequences to insert the single quote, double quote or backslash character into a character or string literal

```
\a   (alert - beep)
\f   (form feed)
\n   (newline)
\r   (carriage return)
\t   (tab)
```

These escape sequences are used to send control codes to format the output to the display monitor, printer, or disk files.

```
\nnn (Create a character code using an octal value nnn)
\xhh (Create a character code using a hex value hh)
```

The following character sequences start with the backslash character and are used to represent control codes and other printable codes. Some of the important ones to remember are:

```
\'   (single quote)
\"   (double quote)
\\   (backslash character)

\n   (newline)
\r   (carriage return)
\t   (tab)

For example, to create a string that has a double-quote
in it, use:
"He said to me, \"What is your plan?\""
the \"" at the end uses \" to place a quote in the string
and the final " ends the qouted string.
```

C-Strings and String Objects

The C-language uses an array of characters to store a string. This is known as a C-string. A null-byte (all zeros) is placed at the end of the string to indicate the End-Of-String (EOS).

Object oriented languages such as C++, VisualBasic, C#, and Java store strings of characters in a string object. The string object is more robust and provides greater protection when processing strings.

Different subroutines or methods are used to manipulate C-strings and string objects.

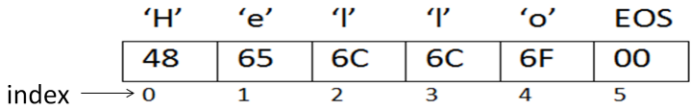
The C-language uses an array of characters to store a string. This is known as a C-string. A null-byte (all zeros) is placed at the end of the string to indicate the End-Of-String (EOS).

Object oriented languages such as C++, VisualBasic, C#, and Java store strings of characters in a string object. The string object is more robust and provides greater protection when processing strings.

Different subroutines or methods are used to manipulate C-strings and string objects.

C-Strings

```
char message[] = "Hello";
```



The array shows how the string "Hello" is stored in memory. The index shows the position in the array. The values in the boxes represent the hex representation of each character.

The square brackets [ ] are used to define an array. Without the square brackets, only a single character would be used and it could not hold a string. The null byte is automatically placed at the end of the string when the string is declared with quote marks.

Defining a String

Language	Type of String	String Definition
C, C++	Array	char message[ ] = "Hello";
C++	Object	string message = "Hello";
Java	Object	String message = "Hello";
Visual Basic	Object	Dim message As String = "Hello"

Here are examples of defining a string using an array in C and C++, or using an object in C++, Java or Visual Basic.

One difference to notice in defining a string object with C++ and Java is that the word String is capitalized in Java but not in C++. Java is very specific in stating that objects have their first letter capitalized. C++ is not quite so definite.

NAMED CONSTANTS

C / C++

```
#define  TAX  0.0725
```

Java

```
public static final double TAX = 0.0725;
```

C++

```
const  double  TAX = 0.0725 ;
```

VisualBasic

```
Const  TAX  As Double = 0.0725
```

Named Constants are similar to both literals and variables in that they cannot be changed, but constants can have a data type. Constants are usually declared in ALL CAPS to distinguish them from variables.

Named Constants

C / C++

```
#define  TAX  0.0725
```

C++

```
const  double  TAX = 0.0725 ;
```

Java

```
public static final double TAX = 0.0725;
```

VisualBasic

```
Const  TAX  As Double = 0.0725
```

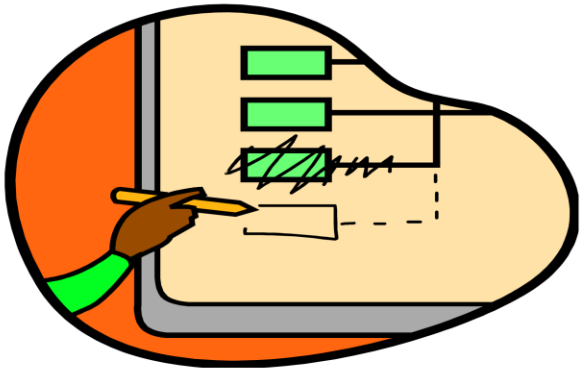
Pre-processor directives start with the hash-mark '#'. The **#define** statement causes a text substitution in the rest of the source code file For example, whenever the word TAX is found, the word TAX is replaced with the characters 0.0725 before the program is compiled. It is important that you do NOT place a semicolon at the end of the line, or the semicolon will also be substituted into your code when it is compiled. The #define directive does not have a data type.

Constants are declared in C++ and VisualBasic the same way as variables are declared, except that the keyword **const** is placed at the beginning of the declaration. Although Java does not exactly have a definition for constants, the keywords **static final** can be used to make the equivalent of a constant.

Magic Numbers

Maybe you want to compute tax on a sale at 7.5% so you multiply the total by 0.075. The answer is correct so you leave the program alone. Either you or someone else comes along later but can't figure out what the 0.0725 is used for. The number just seems to have shown up in the program magically to solve a problem but now no one can figure out why it is there. The number is known as a "**Magic Number**"

Magic Numbers



There is another problem with using Magic Numbers. Suppose you used the value of 30 several times in your program. Later you discovered that the number should be 32, so you changed all the copies of 30 to 32.

Unfortunately, you also might change 300 to 320.

Magic Numbers vs. Named Constants

Using a Magic Number

```
Dim Item1 As Double = 3.95
Dim Item2 As Double = 4.75
Dim SubTotal as Double
Dim SalesTax as Double
Dim Total As Double

SubTotal = Item1 + Item2
SalesTax = SubTotal * 0.0725
Total = SubTotal + Sales Tax
```

Using a Named Constant

```
Const TAXRATE As Double = 0.0725
Dim Item1 As Double = 3.95
Dim Item2 As Double = 4.75
Dim SubTotal as Double
Dim SalesTax as Double
Dim Total As Double

SubTotal = Item1 + Item2
SalesTax = SubTotal * TAXRATE
Total = SubTotal + Sales Tax
```

Here is some sample code in Visual Basic that shows how to use a named constant instead of just placing a numeric variable somewhere in the middle of a program. The concept is the same for Java, C or C++.

In the first example, the value 0.0725 is placed in the code. Someone later may wonder where it came from. But if a constant is defined at the top of the program, TAXRATE = 0.0725 then this named constant can be used later in the program and it becomes a little more evident why it is there. Another good reason to use the named constant is that the value 0.0725 may appear several other times in a larger program when computing tax. If the tax rate changes, all that needs to be done is change the value for the named constant and this happens in only one place. There is a much smaller change of not making all the changes if it appears multiple times in the program.

Boolean Data Type

The boolean data type can only hold a value of **true** or **false**. The boolean data type is great for holding the result of a logical expression. For example:

In C, C++ or Java

```
lightStatus = (switch1==true || switch2==true);
```

In VB

```
lightStatustatus = (switch1 = TRUE OR switch2 = TRUE)
```

The boolean data type can only hold a value of **true** or **false**. The boolean data type is great for holding the result of a logical expression. For example:
In C, C++ or Java
lightStatus = (switch1==true || switch2==true);
In VB
lightStatus = (switch1 = TRUE OR switch2 = TRUE)

Null Data Type

The **null** data type is most often used when the program needs additional memory. When additional memory is allocated from the operating system, the OS returns a reference to the block of memory using the **null** data type.

The **null** data type is most often used when the program needs additional memory. When additional memory is allocated from the operating system, the OS returns a reference to the block of memory using the **null** data type.

The OS has no idea of how the block of memory is going to be used by the program, it just provides the memory. Before the program can use the memory, it must use a **typecast** to something like **int, char** or another data type to identify how the block of memory will be used internally.

```
int buffer[1000] = (int) malloc (1000 * sizeof(int) ); // in C, allocate room for an array of 1000 integers
```

ACKNOWLEDGEMENTS

VIDEO of a Working Teletype

https://www.youtube.com/watch?v=CiUXDHtPnWY

David Comley - jpkiwigeek on YouTube

IMAGES

Wikimedia® Commons

Microsoft® ClipArt

You have reached the end of the video.