C/C++ PROGRAMMING

Data Types Literals, Variables & Constants

Under the Hood

As a DRIVER of an automobile, you may not need to know everything that happens under the hood, although it would be good to know how to jump start the engine without causing an explosion or destroying the battery.

Under the Hood

As a PROGRAMMER, you need to know how to create programs from scratch, and many times you need to know how to maintain and fix programs that other people have written.

You will be the inventor, the creator, the maintainer, the fixer and the documenter. And you may end up going to many meetings and writing many reports.

Literals, Variables & 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 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

LITERALS

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

Examples:

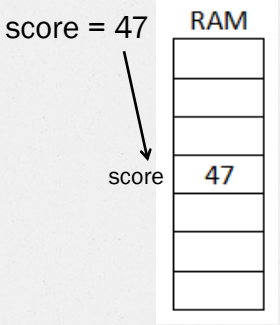
47 – integer literal	3.14185 – real number
TRUE – boolean literal	"Greetings" – string literal

VARIABLES

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



Declaring a Variable

Variables must be declared before they can be used.

Examples of declaring variables:

C C++
int score;
double length;
char selection;
bool PassFail;

Initializing a Variable

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.

C C++

int score = 0;
-or-
int score;
score = 0;

DATA TYPES

- Numeric (integer and floating point)
- Character and Strings
- Boolean (True or False)
- Structured Records
- Pointers and References – discussed much later
- User Defined Data Types – discussed much later

NUMERIC DATA TYPES

- **Integers – whole numbers**
- **Floating-point – numbers with decimal places**

- Character strings are not numbers
  Example:
  StudentID "0123456" (see the quotes)
- Character strings of only digits 0-9 can be converted into integers or floating-point numbers
- WARNING: If the leading digit is a zero, C and C++ consider this an octal number (base 8) not decimal (base 10)

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 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

Enumerated Data Type

An enumeration has a name and a set of members. Each member represents a constant.

Define

```
enum TraffcEnum {
  Red,
  Yellow,
  Green
} trafficLight;
```

Use

```
trafficLight SouthSt;
SouthSt = Red;
```

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

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 math coprocessor, all floating point calculations were done in software. In either case, the computer was very slow and not very accurate when processing floating point math. The data type float used 32 bits and was used 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.

Floating Point Rounding Errors

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.

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

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?

Converting from Integer to Double

An integer can be converted to a double.

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

The result is 75.0

Converting from Double to Integer

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

```
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++

Converting from Double to Integer

What happens to the digits past the decimal when converting from double to integer? Some compilers issues a "Possible loss of precision" warning. If you don't look for the message, you may miss it. Other compilers identify this as an error.

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

value2 is equal to 75
```

Type Casting and Explicit Conversions

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.

```
double value1 = 75.8;
// conversion routine
int value2 = int(value1);
// typecast
int value2 = (int)value1;

// The converted value is

// still rounded down, also
// called truncated
```

Automatic Promotions

Data types are automatically promoted up but not down

```
long double
double
long int
int
char
byte
```

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.

Characters

The CHAR data type holds only a single character.

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

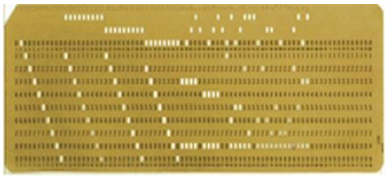
C and C++ use a single quote to define a character literal. The double quote is used to define a string.

ASCII

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

EBCDIC

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.



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

C/C++ Escape Sequences

In C and C++ 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)
```

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

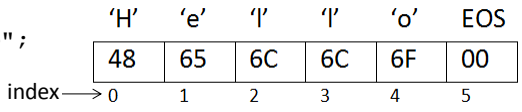
The C-language uses an array of characters to store a string. This is known as a C-string. A null-byte (all zeros) is placed at the end of the string to indicate the End-Of-String (EOS).

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.

C-Strings

The array below 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.

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



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.

NAMED CONSTANTS

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.

C and C++

```
#define  TAX  0.0725
```

C++ only

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

Named Constants

C / C++

```
#define  TAX  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. A semicolon is used in C++ the same as when declaring a variable

C++

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

These constants have a data type that is used during math computations.

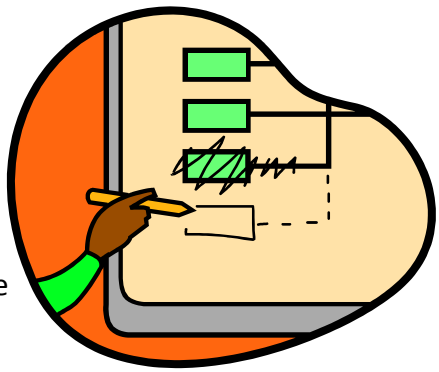
Magic Numbers

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 Numbersvs. Named Constants

Using a Magic Number

```
double item1 = 3.95;
double item2 = 4.75;
double subTotal;
double salesTax ;
double total;

subTotal = item1 + item2;
salesTax = subTotal * 0.0825;
total = subTotal + salesTax;
```

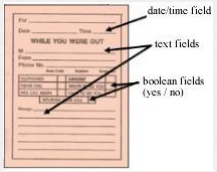
Using a Named Constant

```
const double TAXRATE = 0.0825;
double item1 = 3.95;
double item2 = 4.75;
double subTotal;
double salesTax ;
double total;

subTotal = item1 + item2;
salesTax = subTotal * TAXRATE;
total = subTotal + salesTax;
```

STRUCTURED RECORDS

A structured record can have fields of different data types. You can access an individual field by giving the name of the record, a period, and then the name of the field. More info later on records.




```
struct Employee
{
  int employeeNo;
  char name[40];
  double payRate;
}
```

ACKNOWLEDGEMENTS

IMAGES

Wikimedia® Commons

Microsoft® ClipArt

VIDEO EDITING SOFTWARE

Roxio® Creator™ NXT Pro 2