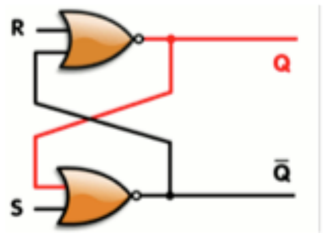
Module 04: Data types and Variables

Intro to Computer Science 1 - C++
Professor Scott Frees

What does a computer do?

1. Computers do arithmetic (we just saw this)
2. Computers remember numbers… how?

- Main Memory (RAM) consists of a vast array of devices similar to flip-flops - they hold a 1 or 0.



- Memory is arranged in *bytes* - 8 bits. ex. 10010101

- Each byte is readable and writable, and is **addressable** - via a numeric address (byte # 42391)

- A typical PC has about 4-8 **billion** bytes of memory.

Data types

- A byte can hold 8 binary digits
- But… a byte can “store” integers, booleans, characters… even colors! How?

We must tell the computer how to interpret the binary numbers we store - we do this by defining the “data type” the bytes will store

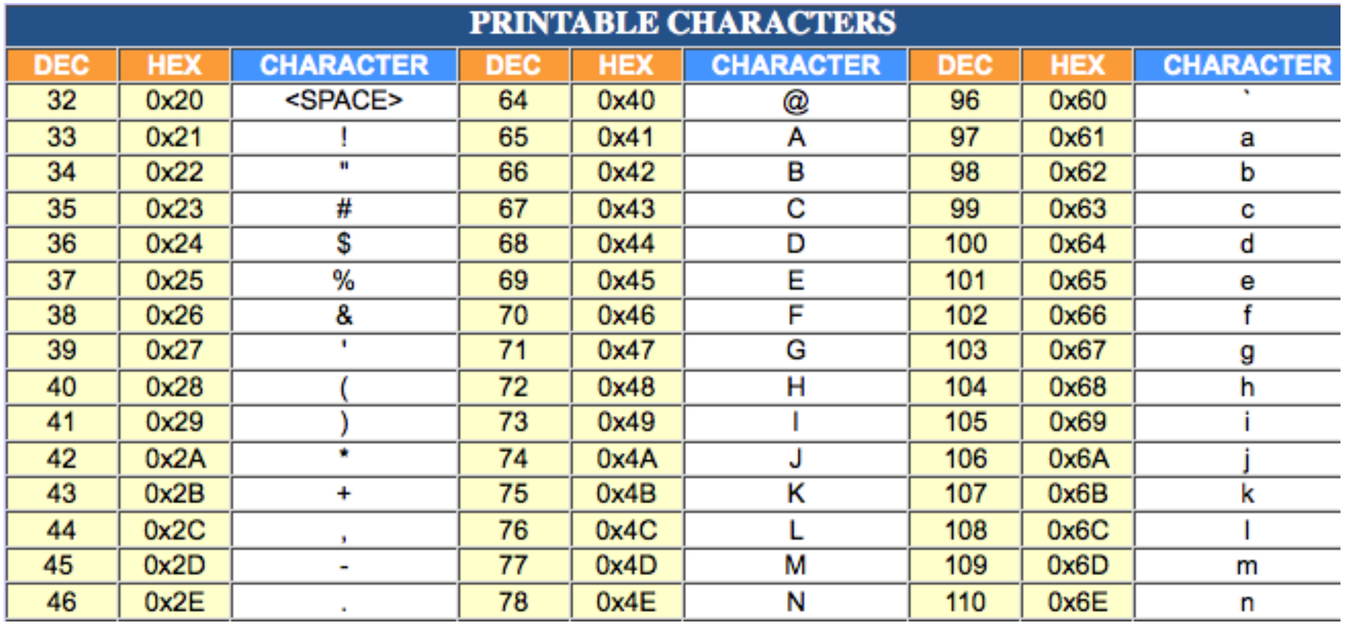
In addition, for some datatypes, we’ll need to *group* bytes together so we can use more bits

Types of data

Characters: `char`

All 128 ASCII characters

2^8 > 128, meaning we only need 1 byte!



PRINTABLE CHARACTERS

DEC	HEX	CHARACTER	DEC	HEX	CHARACTER	DEC	HEX	CHARACTER
32	0x20	<SPACE>	64	0x40	@	96	0x60	`
33	0x21	!	65	0x41	A	97	0x61	a
34	0x22	"	66	0x42	B	98	0x62	b
35	0x23	#	67	0x43	C	99	0x63	c
36	0x24	$	68	0x44	D	100	0x64	d
37	0x25	%	69	0x45	E	101	0x65	e
38	0x26	&	70	0x46	F	102	0x66	f
39	0x27	'	71	0x47	G	103	0x67	g
40	0x28	(	72	0x48	H	104	0x68	h
41	0x29	)	73	0x49	I	105	0x69	i
42	0x2A	*	74	0x4A	J	106	0x6A	j
43	0x2B	+	75	0x4B	K	107	0x6B	k
44	0x2C	,	76	0x4C	L	108	0x6C	l
45	0x2D	-	77	0x4D	M	109	0x6D	m
46	0x2E	.	78	0x4E	N	110	0x6E	n

Types of data

Integers

`short`	Holds values from -2^{15} to 2^{15} Requires 2 bytes (16 bits) 1 bit reserved for +/- Numeric range +/- 32,768
`unsigned short`	Holds values from 0 - 65,535

Types of data

Integers

`int`	Holds values from -2^{31} to 2^{31}
	Requires 4 bytes (32 bits)
	1 bit reserved for +/-
	Numeric range +/- 2,147,483,648
`unsigned int`	Holds values from 0-4,294,967,295

Data types

Are integers always 32 bits? 4 bytes?

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

int main() {
    cout << sizeof(short) << endl;
    cout << sizeof(int) << endl;
    cout << sizeof(long) << endl;
}
```

C++ makes relative promises, but not absolute promises about data type sizes

Decimal numbers

`float`	4 bytes IEEE 754 format (scientific notation) +/- $1.79e^{+/-308}$
`double`	8 bytes +/- $1.18e^{+/-4932}$

Others

`bool`	true or false
	1 byte
	`00000001`
	`00000000`

Strings? *We will see strings later - but for now we will just think of them as a sequence of characters.*

Storing data

We must "reserve" bytes in memory before storing data there.

Instead of picking a byte # and size, we use a *higher level of abstraction* - a variable

```
int x;          // creates an integer variable (4 bytes)
double y;       // creates a double variable (8 bytes)
char c;         // creates a character variable (1 byte, ASCII)
```

Storing data

To **put things in variables**, we need to ASSIGN them.

- The = sign is the assignment operator

```
int x;
double y;
double z;
char c;
x = 5;          // stores 5 in x
y = 7.8         // stores 7.8 in y
c = 'w';            // stores 'w' in x
c = '6';            // stores the symbol '6' in c
z = y;
z = y + 1.7;
```

Program Example 01: Time

Lets make things a bit interesting…

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

int main() {
   int total = time(0);
   cout << “Seconds since January 1, 1970:  “
         << total << endl;
}
```

Lets build on this, using % and integer division to display the current time as hours, minutes, seconds GMT

= operator

= is **not** a rule.... its an **action**

Lets try this:

Left side is **always** a variable
Right side is an **expression**

Expressions are *evaluated*

```
int x = 5;
int y = x + 5;
x = 10;
cout << “x = “ << x << endl;
```

… also expressions

Variables and Input

We can store **literals** in variables

```
x = 5;
```

We can store results in variables

```
x = 5 + 9;
```

We can store result of function calls in variables

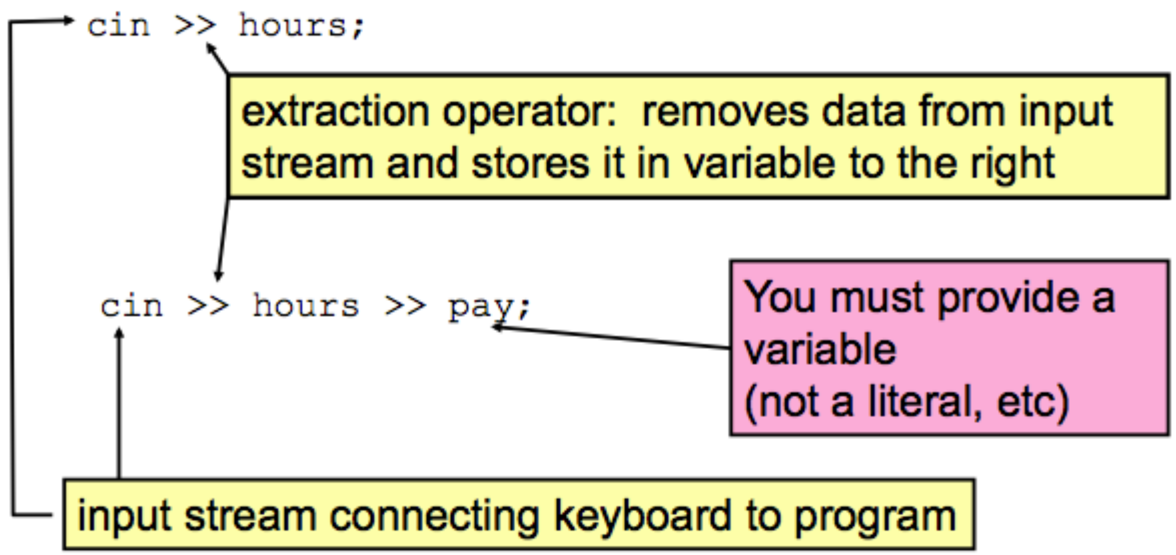
```
total = time(0)
```

We can print variables

```
cout << x << endl;
```

Ideally, we should be able to ask the user for data too...

cin - Console input


cin >> hours;
extraction operator: removes data from input stream and stores it in variable to the right
cin >> hours >> pay;
You must provide a variable (not a literal, etc)
input stream connecting keyboard to program

We can now start to build a dialog with the user - using cout to prompt, and cin to gather data

Working with user

- Use cout and cin to start a dialog with the user of your program:
    - Use cout to ask the user for some data (prompt)
    - Use cin to then read it in.
    - You cannot mix cout with cin!

- Make sure you have “read” the data before using it!

Programming Example 02: Circle

Ask the user for the radius of a circle

Calculate the area of the circle

Print out the radius and the area

const keyword

Variables can change (they vary!)

The value of pi doesn't… its always the same.

```
const double PI = 3.14159
```

- Its always a good idea to declare as const - it lets the compiler find your "logical" errors.
- This approach helps you avoid repeating special numbers - which avoids typos!

Operator Precedence

Mathematical operators follow the standard algebraic rules we all should already know…

- Operators can be strung together to form longer expressions

```
x = 5 + 6 * 18/ 9 - 2;  // sets x to 15
```

- In order to evaluate, C++ uses standard operator precedence rules (just like in algebra)
- Can be overridden using parenthesis

```
x = (5+6)*18/(9-2); // sets x to 22;
```

Lab 02

Create a program that asks the user for a temperature, in Celsius degrees

Compute the corresponding value in Fahrenheit

$F = 9/5C + 32$

Print out **both** temperature values with appropriate labels.