cisc1110 fall 2010 lecture II.2

- *simple data types*
- characters
- boolean type

storing data: reprise

- last class, we talked about storing *numeric* data
- now we are going to talk about storing another type of data, specifically *character* data
- as with numeric data, you can think of the computer's memory as a bunch of boxes
- inside each box, instead of a number, there is a "character"
- a *character* can either be a letter: `a..z` or `A..Z`
  a punctuation mark:
  `~ ‘ ! @ # $ % ^ & * ( ) - _ + = { } | [ ] \ : ; " ’ , . / < > ?`
  or a digit: `0..9`
- notice that these are all things you can type on your keyboard!
- also notice that a *digit character* is NOT the same thing as single-digit number (more on that later)

- as with numeric data, when you store character data you give each box a name
  $\Rightarrow$ which, in C++ is called "declaring a *variable*"
- example:

  program code:

  ```
  char x;
  ```

  computer's memory:

  x $\rightarrow$ ☐
- in the example program code above, the name of the box is `x`
- preceding the name `x` is the word `char`, which is the variable's *data type*
- as with numeric data types, character data types come in different sizes:
  - `char` : character, for storing simple characters
  - `wchar_t` : wide character, for storing complex "wide" characters

assigning values

- we'll start by talking about `char` variables
- as with numbers, = is the assignment operator for character variables
- example:

  program code:

  ```
  char x; // declaration
  x = 'A'; // assignment
  ```

  computer's memory:

  x $\rightarrow$ 'A'

  or

  ```
  char x = 'A'; // declaration and assignment together
  ```
- note the use of the single quotes (') surrounding the character A
- next week, we'll talk about the use of double quotes (") for a different type of data (called a "string" — but let's not get ahead of ourselves today)

character digits versus single-digit numbers

- Note that:
  program code:
  ```
  char x = '2';
  ```
  computer's memory:
  x → ['2']
- is NOT the same as
  program code:
  ```
  int x = 2;
  ```
  computer's memory:
  x → [2]
- the character '2' is stored differently in the computer from how the integer 2 is stored
- the integer 2 is stored like this:
  [0 0 0 0 0 0 1 0]
- the character, or *digit*, '2' is stored like this:
  [0 0 1 1 0 0 1 0]
  it has the numeric, ASCII character code value of 50

outputting character variables

- you can output the value of a character variable using `cout`
- for example:

```
char i;
i = 'A';
cout << "the value of i is " << i << endl;
```

- or we could have written the following, which would produce the same thing as above:

```
cout << "the value of i is " << i << "\n";
```


storing characters: ASCII

- ASCII = American Standard Code for Information Interchange
- characters are stored as numbers
- standard table defines 128 characters
- for example, when you define:
  `char c = 'A';` the data is stored as a number:
  'A' = 65_{10} = 01000001_2
  like this:
  c →

7	6	5	4	3	2	1	0
0	1	0	0	0	0	0	1

- sometimes it is handy to *convert* between integers and characters explicitly
- for example:

```
char c = 'A';
int i;
i = (int)c;
```

in which case, the value of `i` will be 65.

wide characters

- wide characters take up 2 bytes in the computer's memory, instead of just 1 (like normal characters)
- the wide character data type is `wchar_t`
- it was created to support *internationalization* (as in Unicode)
- however, handling of Unicode characters is not well standardized, unfortunately
- so we'll just use regular `char` in this class for now...

boolean variables

- there is one more simple, or *primitive*, data type, and that is called `bool`
- `bool` comes from boolean with comes from George Boole, who was an English mathemetician who lived in the first part of the 1800's
- he formalized a type of logic that is now called "Boolean Logic"
- this logic formalism operates on values that are `true` or `false`
- so a `bool` variable has one of two values: `true`, which is represented by `1`, or `false`, which is represented by `0`

logical operators

- *boolean expressions* combine `bool` variables and represent things that are either `true` or `false`
- in C++, there are three *logical operators* that are used with `bool` variables and boolean expressions:

`&&`	and
`||`	or
`!`	not

- and can be used to put together multiple conditions, for example:

```
bool raining = false;
bool cloudy = true;
bool umbrella = ( raining && cloudy );
cout << "should I take an umbrella? " << umbrella << endl;
umbrella = ( raining || cloudy );
cout << "should I take an umbrella? " << umbrella << endl;
```


- the first question is answered `0` or "false"
- the second question is answered `1` or "true"
- explanation follows on the next page when we introduce *truth tables*

truth tables

- AND

a	b	a && b
true	true	true
true	false	false
false	true	false
false	false	false

- OR

a	b	a || b
true	true	true
true	false	true
false	true	true
false	false	false

- NOT

a	! a
false	true
true	false