

## Type casting

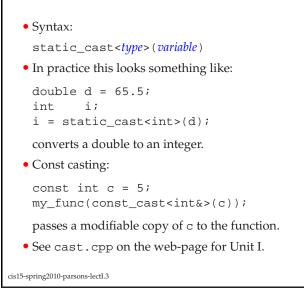
- Used to convert between fundamental (simple) data types (e.g., int, double, char)
- There are two ways to do this
- The C way (technically obsolete):

double d = 65.0; int i = (double)d; char c = (char)i;

- The C++ way:
  - static\_cast: for conversions that are "well-defined, portable, intertable"; e.g., like the C ways, above.
  - reinterpret\_cast: for conversions that are system-dependent (not recommended).
  - const\_cast: to create a modifiable copy of a const variable; data type into which the value is cast must always be a pointer or reference (see on).
  - dynamic\_cast: for converting between classes (to be discussed later in the term)

cis15-spring2010-parsons-lectI.3

3



```
You create an enum data type if you want to use the names instead of the values, so you shouldn't really care what the values are internally.
If you need to set the value explicitly, you can: enum answer { yes, no, maybe = -1 );
If you do this you have to be careful about duplicated values (see enum. cpp).
syntax: enum tag { value0, value1, ... valueN };
The tag is optional.
You can also declare variables of the enumerated type by adding the variable name after the closing }
See enum. cpp
```

# Enumeration types

- Used to declare names for a set of related items
- For example: enum suit { diamonds, clubs, hearts, spades };
- Internally, each name is assigned an int value.
- The value assigned to the first name is zero.
- The value of each member of the list is then one more than its lefthand neighbor.
- So in the above example, diamonds is actually 0, clubs is 1, and so on.

```
cis15-spring2010-parsons-lectI.3
```

```
void showSuit( int card ) {
   enum suits { diamonds, clubs, hearts, spades } suit;
   suit = static_cast<suits>( card / 13 );
   switch( suit ) {
      case diamonds: cout << "diamonds"; break;</pre>
      case clubs:
                       cout << "clubs";</pre>
                                              break;
      case hearts:
                      cout << "hearts";</pre>
                                             break;
      case spades:
                      cout << "spades";</pre>
                                             break;
    cout << endl;
cis15-spring2010-parsons-lectI.3
```

# typedef

- The typedef keyword can be used to create names for data types
- A typedef name is just a synonym.
- For example:

- Then you use the name you've created (numbers, letters or suits from the example above)
- See typedef.cpp

cis15-spring2010-parsons-lectI.3

### Precedence and associativity table

#### (listed in order of precedence)

operator	associativity		
:: (global scope), :: (class scope)	left to right		
[], ->, ++ (postfix), (postfix), dynamic_cast <type> (etc)</type>			
++ (prefix); (postfix), !, sizeof(), + (unary), - (unary), * (indirection)	right to left		
*, /, %	left to right		
+, -	left to right		
<<,>>>	left to right		
<, <=, >>=	left to right		
==,!=	left to right		
&	left to right		
$\wedge$	left to right		
	left to right		
&&	left to right		
	left to right		
?:	left to right		
$=,+=,-=,*=,/=,\%=,>>=,<<=,\&=,\wedge=, =$	left to right		
See prec.cpp			

# Precedence and associativity

- "Precedence" means the order in which multiple operators are evaluated
- "Associativity" means which value an operator *associates* with, which is particularly good to know if you have multiple operators adjacent to a single variable
- Associativity is either:
  - left to right, e.g., 3 2 (subtract 2 from 3)
  - right to left, e.g., -3 (meaning negative 3)
- Note that ++ and -- can be either:
  - *postfix* operators are left to right (meaning that you evaluate the expression on the left first and then apply the operator)
  - *prefix* operators are right to left (meaning that you apply the operator first and then evaluate the expression on the right)

10

cis15-spring2010-parsons-lectI.3

	-			
			Combrol floor	
			Control flow	
		• Branching:		
		- if,		
		-if-else,		
		- switch		
		<ul> <li>Looping:</li> </ul>		
		-for,		
		-while,		
		-dowhile		
		• See control.cpp		
11		cis15-spring2010-parsons-lectI.3		12

### Random numbers

• To generate random numbers we use the function rand()

• For example;

```
int x;
x = rand();
```

- This assigns a random value to x. The value is somewhere between 0 and (at least) 32767.
- To generate numbers between 0 and 6 we use:

x = rand() % 7;

• To generate numbers between 2 and 8 we use:

x = 2 + rand() % 7;

cis15-spring2010-parsons-lectI.3

- To use rand(), we need to add #include<cstdlib> to our program.
- Each time we run our program rand() will produce some (apparently) random numbers.
- But it will produce the *same* numbers each time we run the program.
- To get different numbers each time we run the program, we need to seed the random number generator.

14

16

• The usual way to do that is to add:

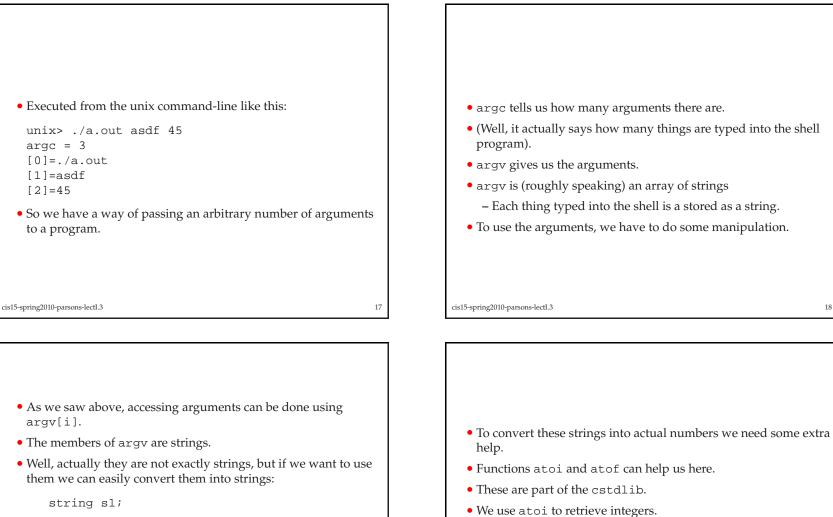
srand(time(NULL));

- The time(NULL) uses the clock to generate a seed.
- We have to add #include<ctime> to do this.

cis15-spring2010-parsons-lectI.3

### **Command-line** arguments **Command-line** arguments • Example: #include <iostream> • The UNIX commands we looked at last time are just C/C++ using namespace std; programs int main( int argc, char \*\*argv ) { • They have a different form of interaction from the programs you cout << "argc = " << argc << endl;</pre> wrote for CIS 1.5. for ( int i=0; i<argc; i++ ) {</pre> cout << "[" << i << "]=" << argv[i] << endl; • Command line arguments. g++ myprog.cpp -o myprog.o } // end of main() • Turns out that C/C++ makes it easy to write programs like this. • cmdline.cpp cis15-spring2010-parsons-lectI.3 cis15-spring2010-parsons-lectI.3 15

13



19

```
s1 = argv[1];
if(s1 == "asdf"){
   cout << "Correct!";</pre>
```

cis15-spring2010-parsons-lectI.3

```
cis15-spring2010-parsons-lectI.3
```

integer.

double.

• atoi(argv[2]) will convert the third element of argv into an

• atof(argv[3]) will convert the fourth element of argv into a

20

18

