COMMAND LINE ARGUMENTS
Today

• We will recap some C++ basics
  – Type casting
  – Enumeration types
  – typedef
  – Precedence and associativity
  – Control flow

• We’ll also introduce what is probably a new topic for most of you:
  – Command line arguments
**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):
  ```c
  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)
• Syntax:
  \[ \text{static\_cast}<\text{type}>(\text{variable}) \]

• In practice this looks something like:

  ```cpp
  double d = 65.5;
  int i;
  i = static\_cast<int>(d);
  ```

  converts a double to an integer.

• Const casting:

  ```cpp
  const int c = 5;
  my\_func(const\_cast<int&>(c));
  ```

  passes a modifiable copy of \( c \) to the function.

• See \texttt{cast.cpp} on the web-page for Unit I.
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.
• 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:
  ```cpp
  enum answer { yes, no, maybe = -1 };
  ```

• If you do this you have to be careful about duplicated values (see `enum.cpp`).

• Syntax:
  ```cpp
  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`
void showSuit( int card ) {

    enum suits { diamonds, clubs, hearts, spades } suit;

    suit = static_cast<suits>( card / 13 );

    switch( suit ) {
        case diamonds: cout << "diamonds"; break;
        case clubs: cout << "clubs"; break;
        case hearts: cout << "hearts"; break;
        case spades: cout << "spades"; break;
    }

    cout << endl;

}
The `typedef` keyword can be used to create names for data types.

A `typedef` name is just a synonym.

For example:

```c
typedef int numbers; // "numbers" is my name
typedef char letters; // "letters" is my name
typedef enum suits { diamonds, clubs, hearts, spades };
```

Then you use the name you’ve created (`numbers`, `letters` or `suits` from the example above).

See `typedef.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)
# Precedence and associativity table

(listed in order of precedence)

<table>
<thead>
<tr>
<th>operator</th>
<th>associativity</th>
</tr>
</thead>
<tbody>
<tr>
<td>:: (global scope), :: (class scope)</td>
<td>left to right</td>
</tr>
<tr>
<td>[], − &gt;, ++ (postfix), − − (postfix), dynamic_cast&lt;type&gt; (etc)</td>
<td>left to right</td>
</tr>
<tr>
<td>++ (prefix); − − (postfix), !, sizeof(), + (unary), − (unary), * (indirection)</td>
<td>right to left</td>
</tr>
<tr>
<td>*, /, %</td>
<td>left to right</td>
</tr>
<tr>
<td>+, −</td>
<td>left to right</td>
</tr>
<tr>
<td>&lt;&lt;, &gt;&gt;</td>
<td>left to right</td>
</tr>
<tr>
<td>&lt;, &lt;=, &gt;, &gt;=</td>
<td>left to right</td>
</tr>
<tr>
<td>==, !=</td>
<td>left to right</td>
</tr>
<tr>
<td>&amp;</td>
<td>left to right</td>
</tr>
<tr>
<td>^</td>
<td>left to right</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>&amp;&amp;</td>
<td>left to right</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>? :</td>
<td>left to right</td>
</tr>
<tr>
<td>=, +=, − =, *=, /=, %=, &gt;&gt;=, &lt;&lt;=, &amp;=, &amp;, ^,</td>
<td>=</td>
</tr>
</tbody>
</table>

See prec.cpp
Control flow

• Branching:
  – if,
  – if-else,
  – switch

• Looping:
  – for,
  – while,
  – do...while

• See control.cpp
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;
  ```
• 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.
• The usual way to do that is to add:
  ```c
  srand(time(NULL));
  ```
• The `time(NULL)` uses the clock to generate a seed.
• We have to add `#include<ctime>` to do this.
**Command-line arguments**

- The UNIX commands we looked at last time are just C/C++ programs.
- They have a different form of interaction from the programs you wrote for CIS 1.5.
- Command line arguments.
  
  \[ g++ \ myprog\_cpp\ -o\ myprog\_o \]
- Turns out that C/C++ makes it easy to write programs like this.
Command-line arguments

• Example:

```cpp
#include <iostream>
using namespace std;
int main( int argc, char **argv ) {
    cout << "argc = " << argc << endl;
    for ( int i=0; i<argc; i++ ) {
        cout << "[" << i << "]=" << argv[i] << endl;
    }
} // end of main()
```

• cmdline.cpp
• Executed from the unix command-line like this:

  unix> ./a.out asdf 45
  argc = 3
  [0]=./a.out
  [1]=asdf
  [2]=45

• So we have a way of passing an arbitrary number of arguments to a program.
• `argc` tells us how many arguments there are.
• (Well, it actually says how many things are typed into the shell program).
• `argv` gives us the arguments.
• `argv` is (roughly speaking) an array of strings
  – Each thing typed into the shell is stored as a string.
• To use the arguments, we have to do some manipulation.
• As we saw above, accessing arguments can be done using `argv[i]`.

• The members of `argv` are strings.

• Well, actually they are not exactly strings, but if we want to use them we can easily convert them into strings:

```cpp
string s1;

s1 = argv[1];

if(s1 == "asdf"){
    cout << "Correct!";
}
```
• To convert these strings into actual numbers we need some extra help.
• Functions `atoi` and `atof` can help us here.
• These are part of the `cstdlib`.
• We use `atoi` to retrieve integers.
• `atoi(argv[2])` will convert the third element of `argv` into an integer.
• `atof(argv[3])` will convert the fourth element of `argv` into a double.
• How would we write a simple calculator?

    unix> calc + 2 3
    unix> 5
    unix> calc * 2 4
    unix> 8
    unix>

• It should be able to add, subtract, multiply and divide two integers
Summary

• This lecture finished up our quick revision of the material from CIS 1.5

• We looked at:
  – Type casting
  – Enumeration types
  – typedef
  – Precedence and associativity
  – Control flow
  – Command line arguments

• The new thing we covered was the Unix/C++ mechanism for handling command line arguments.