

topics:

- type casting
- enumeration types
- typedef
- precedence and associativity
- control flow
- 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
- (1) the C way (technically obsolete):

```
double d = 65.0;
int i = (int)d;
char c = (char)i;
```
- (2) the C++ way:
 - `static_cast`: for conversions that are “well-defined, portable, invertable”; e.g., like the C ways, above
 - `reinterpret_cast`: for conversions that are system-dependent (not recommended)
 - `const_cast`: for conversions where the value of the variable being converted cannot be changed; data type must always be a pointer or reference
 - `dynamic_cast`: for converting between classes (to be discussed later in the term)
- syntax:

```
static_cast<type>(variable)
```

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, in order starting with 0
- so in the above example, diamonds is actually 0, clubs is 1, and so on
- but you create an enum data type if you want to use the names instead of the values, so you don't really care what the values are internally
- although there are cases when you do want to set the value explicitly, e.g.:

```
enum answer { yes, no, maybe = -1 };
```
- 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 }

- example:

```
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;
} // end of showSuit()
```

typedef

- the typedef keyword can be used to create names for data types
- for example:

```
typedef int numbers; // "numbers" is my own name
typedef char letters; // "letters" is my own name
typedef suits enum { diamonds, clubs, hearts, spades };
```
- and then you use the name you've created (numbers, letters or suits from the example above)

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)

operator	associativity
:: (global scope), :: (class scope)	left to right
, ->, ++ (postfix), -- (postfix), dynamic_cast<type> (etc)	left to right
++ (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
^	left to right
	left to right
&&&	left to right
	left to right
?:	left to right
., +, -, =, +=, -=, *=, /=, %=, >>=, <<=, &=, ^=, =	left to right

control flow

- branching: if, if-else, switch
- looping: for, while, do...while
- interruption: break, continue

command-line arguments

- example:

```
#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()
```

- executed from the unix command-line like this:

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