Enumerations

User-defined types:

There are several ways of defining new type names in C++. Here are some common ones:

- struct
- class
- typedef
- enum (enumeration)

Enumerations: This page focuses on the *enumeration*. When you create an enumerated type, you are making up a new type that has a small set of specific values, which are listed in the declaration. The compiler will implement these internally as a set of integer constants. To create an enumeration, use the keyword **enum**, and the following syntax:

```
enum enumerationName { list of enumeration constants };
```

Examples:

```
enum Names {RALPH, JOE, FRED};
enum Direction {EAST, NORTH, WEST, SOUTH};
```

Now, if you declare a variable of type Names, the symbols RALPH, JOE, and FRED are the actual values that can be used with these variables. Note, these words are NOT character strings. They are stored by the computer as constant values. Enumerations are essentially used for making code easier to read, and the values of certain variables easier to remember. Examples of uses of these enum examples declared above:

```
Names who;    // who is a variable of type Names
Direction d;  // d is a variable of type Direction

who = FRED;        // assign the value FRED to the variable who
if (who == JOE)
    cout << "Halleluia!";
who = JOE;
switch(who)
{   case JOE:    cout << "Joe"; break;
    case FRED:   cout << "Fred"; break;
    case RALPH:  cout << "Ralph"; break;
}

char choice;
cout << "Type in a direction (N)orth, (S)outh, (E)ast, (W)est: ";
cin >> choice;
switch(choice)
{
  case 'N':   d = NORTH;      break;
  case 'S':   d = SOUTH;      break;
  case 'E':   d = EAST;       break;
  case 'W':   d = WEST;       break;
```

```
}

if (d == NORTH)
      cout << "Look out!  There's a polar bear!!";
```

Another enumeration example

```
enum Days {SUN, MON, TUE, WED, THUR, FRI, SAT};

Days today, tomorrow, yesterday;

today = MON;
if (today == SUN)
    yesterday = SAT;
if (tomorrow == FRI)
    cout << "Today is Thursday!");
```

Important advantages of enumerations

- Readability.
  - A statement like `{ direction = NORTH; }` is more intuitive to the reader than `{ direction = 1; }` (in which the reader must memorize what the number 1 stands for.
- Error Checking (often not needed)
  - In the **Days** enumeration above, there are only 7 possible values that a variable of type Days could take. Suppose such a variable is passed into a function. The function would not need to worry about whether this parameter had a valid day stored. There are only 7 possibilities, and all are valid.
  For contrast, think about a situation in which we pass in an integer, where 1 means Sunday, 2 means Monday, etc. What would happen if 10 were passed in? The function would have to error check to handle this.