Other C++ features
Preprocessor

• Directives:
  • `#include <filename>`
  • `#include “filename.h”`
  • `#define name value`
  • `#undef name`
  • `#if test`
  • `#ifdef name`
  • `#ifndef name`
  • `#else`
  • `#endif`
Preprocessor cont.

• The preprocessor can be used for conditional compilation:

```c
#define FEATURE
...
#ifdef FEATURE
  // code that uses FEATURE
  ...
#else
  // alternate code that doesn't use FEATURE
  ...
#endif
```
**typedef statement**

- Gives a contextually meaningful name to an existing data type
- Replacement is done early in compilation so the two names are operationally synonymous

```cpp
typedef vector<int> ivec;
typedef vector<int>::iterator ivitt;

int main() {
    ivec array;
    ...
    for (ivitt i = array.begin(); i != array.end(); ++i)
        cout << *i << endl;
    return 0;
}
```
The \texttt{const} keyword

\begin{itemize}
  \item \texttt{const} defines a quantity that does not change
  \item Can be applied to variables, arguments, and methods
\end{itemize}

\begin{verbatim}
const int maxStudents = 60;

int foo( const box & aBox ) {
  // Content of aBox cannot be changed.
}

class string {
  public:
    ...
    // Computing length does not alter the string.
    int length( ) const;
    ...
};
\end{verbatim}
The `const` keyword cont.

- `const` can be removed by casting it away:

```cpp
const char * name = "Joe";
char * p = static_cast<char *>(name);
p[0] = 'm';   // Change the name to Moe
```
Namespaces

• Namespaces help avoid naming conflicts in large programs or with third-party libraries

• Each namespace defines a separate scope

• To use a namespace member, a fully qualified name must be used

```cpp
namespace mfc {
    class string {
        ...
    };
} // no semicolon

namespace std {
    class string {
        ...
    };
}

std::string s1;
mfc::string s2;
```
Namespaces cont.

- The using declaration defines which string to use in declarations:

```cpp
using std::string;
string s1;
mfc::string s2;
```

- The using directive specifies some namespace should always be used:

```cpp
using namespace std;
string s1;
mfc::string s2;
```
Other C++ features

**static**

- Static variable at file scope
  - Allocated when program begins and deallocated when program ends
  - Only accessible to functions in the file it is declared (contrary to a regular global variable)

- Static variable in a function
  - Variable retains its value between function calls
  - Can for example be used to count number of calls to the function
**static and objects**

- **Static data members**
  - All objects of that class share one copy of the data member
  - Must be initialized at file scope
- **Static member functions**
  - Function can only access static data members