10.8 Separate files

Separating part of a program's code into a separate file can yield several benefits. One benefit is preventing a main file from becoming unmanageably large. Another benefit is that the separated part could be useful in other programs.

Suppose a program has several related functions that operate on triples of numbers, such as computing the maximum of three numbers or computing the average of three numbers. Those related functions' definitions can be placed in their own file as shown below in the file threeintsfcts.cpp.

Figure 10.8.1: Putting related functions in their own file.

|  |  |  |
| --- | --- | --- |
| main.cpp | threeintsfcts.cpp | > a.out  35  11  > |
| #include *<iostream>*  #include *"threeintsfcts.h"*  using namespace std;  *// Normally lots of other code here*  int main() {  cout << ThreeIntsSum(5, 10, 20) << endl;  cout << ThreeIntsAvg(5, 10, 20) << endl;  return 0;  }  *// Normally lots of other code here* | int ThreeIntsSum(int num1, int num2, int num3) {  return (num1 + num2 + num3);  }  int ThreeIntsAvg(int num1, int num2, int num3) {  int sum;  sum = num1 + num2 + num3;  return (sum / 3);  } |
| threeintsfcts.h |
| int ThreeIntsSum(int num1, int num2, int num3);  int ThreeIntsAvg(int num1, int num2, int num3); |

Feedback?

One could then compile the main.cpp and threeintsfcts.cpp files together as shown below.

Figure 10.8.2: Compiling multiple files together.

|  |  |
| --- | --- |
| Without #include "threeintsfcts.h" in main.cpp | With #include "threeintsfcts.h" in main.cpp |
| > g++ -Wall main.cpp threeintsfcts.cpp  main.cpp: In function int main():  main.cpp:8: error: ThreeIntsSum was not declared in this scope  main.cpp:9: error: ThreeIntsAvg was not declared in this scope | > g++ -Wall main.cpp threeintsfcts.cpp  > |

Feedback?

Just compiling those two files (without the #include "threeintsfcts.h" line in the main file) would yield an error, as shown above on the left. The problem is that the compiler does not see the function definitions while processing the main file because those definitions are in another file, which is similar to what occurs when defining functions after main(). The solution for both situations is to provide function declarations before main() so the compiler knows enough about the functions to compile calls to those functions. Instead of typing the declarations directly above main(), a programmer can provide the function declarations in a header file, such as the threeintsfcts.h file provided in the figure above. The programmer then includes the contents of that file into a source file via the line: #include "threeintsfcts.h".

The reader may note that the .h file could have contained function definitions rather than just function declarations, eliminating the need for two files (one for declarations, one for definitions). However, the two file approach has two key advantages. One advantage is that with the two file approach, the .h file serves as a brief summary of all functions available. A second advantage is that the main file's copy does not become exceedingly large during compilation, which can lead to slow compilation.

One last consideration that must be dealt with is that a header file could get included multiple times, causing the compiler to generate errors indicating an item defined in that header file is defined multiple times (the above header files only declared functions and didn't define them, but other header files may define functions, types, constants, and other items). Multiple inclusion commonly can occur when one header file includes another header file, e.g., the main file includes file1.h and file2.h, and file1.h also includes file2.h -- thus, file2.h would get included twice into the main file.

The solution is to add some additional preprocessor directives, known as header file guards, to the .h file as follows.

Construct 10.8.1: Header file guards.

#ifndef FILENAME\_H

#define FILENAME\_H

*// Header file contents*

#endif

Feedback?

***Header file guards*** are preprocessor directives, which cause the compiler to only include the contents of the header file once. #define FILENAME\_H defines the symbol FILENAME\_H to the preprocessor. The #ifndef FILENAME\_H and #endif form a pair that instructs the preprocessor to process the code between the pair only if FILENAME\_H is not defined ("ifndef" is short for "if not defined"). Thus, if the preprocessor includes encounter the header more than once, the code in the file during the second and any subsequent encounters will be skipped because FILENAME\_H was already defined.

*Good practice is to guard every header file.* The following shows the threeintsfcts.h file with the guarding code added.

Figure 10.8.3: All header files should be guarded.

#ifndef THREEINTSFCTS\_H

#define THREEINTSFCTS\_H

int ThreeIntsSum(int num1, int num2, int num3);

int ThreeIntsAvg(int num1, int num2, int num3);

#endif

10.9 Separate files for classes

**Two files per class**

Programmers typically put all code for a class into two files, separate from other code.

* ***ClassName.h*** contains the class definition, including data members and member function declarations.
* ***ClassName.cpp*** contains member function definitions.

A file that uses the class, such as a main file or ClassName.cpp, must include ClassName.h. The .h file's contents are sufficient to allow compilation, as long as the corresponding .cpp file is eventually compiled into the program too.

The figure below shows how all the .cpp files might be listed when compiled into one program. Note that the .h file is not listed in the compilation command, due to being included by the appropriate .cpp files.

Figure 10.9.1: Using two separate files for a class.

|  |  |
| --- | --- |
| StoreItem.h  #ifndef STOREITEM\_H  #define STOREITEM\_H  class StoreItem {  public:  void SetWeightOunces(int ounces);  void Print() const;  private:  int weightOunces;  };  #endif | StoreItem.cpp  #include *<iostream>*  using namespace std;  #include *"StoreItem.h"*  void StoreItem::SetWeightOunces(int ounces) {  weightOunces = ounces;  }  void StoreItem::Print() const {  cout << "Weight (ounces): " << weightOunces << endl;  } |
| main.cpp  #include *<iostream>*  using namespace std;  #include *"StoreItem.h"*  int main() {  StoreItem item1;  item1.SetWeightOunces(16);  item1.Print();  return 0;  } | Compilation example  % g++ -Wall StoreItem.cpp main.cpp  % a.out  Weight (ounces): 16 |

Feedback?

Good practice for .cpp and .h files

*Sometimes multiple small related classes are grouped into a single file to avoid a proliferation of files. But for typical classes, good practice is to create a unique .cpp and .h file for each class.*

**Ex: Restaurant review classes**

The restaurant review program, introduced in an earlier section, declared the Review, Reviews, and Restaurant classes in main.cpp. Each of the 3 classes should instead be implemented in .h/.cpp files, thus making for cleaner code in main.cpp.

Figure 10.9.2: .h and .cpp files for Review, Reviews, and Restaurant classes.

|  |  |
| --- | --- |
| Review.h  #ifndef REVIEW\_H  #define REVIEW\_H  #include *<string>*  class Review {  public:  void SetRatingAndComment(  int revRating,  std::string revComment);  int GetRating() const;  std::string GetComment() const;  private:  int rating = -1;  std::string comment = "NoComment";  };  #endif | Review.cpp  #include *"Review.h"*  using namespace std;  void Review::SetRatingAndComment(int revRating, string revComment) {  rating = revRating;  comment = revComment;  }  int Review::GetRating() const {  return rating;  }  string Review::GetComment() const {  return comment;  } |
| Reviews.h  #ifndef REVIEWS\_H  #define REVIEWS\_H  #include *<vector>*  #include *"Review.h"*  class Reviews {  public:  void InputReviews();  void PrintCommentsForRating(int currRating) const;  int GetAverageRating() const;  private:  std::vector<Review> reviewList;  };  #endif | Reviews.cpp  #include *<iostream>*  #include *"Reviews.h"*  using namespace std;  *// Get rating comment pairs, add each to list. -1 rating ends.*  void Reviews::InputReviews() {  Review currReview;  int currRating;  string currComment;  cin >> currRating;  while (currRating >= 0) {  getline(cin, currComment); *// Gets rest of line*  currReview.SetRatingAndComment(currRating, currComment);  reviewList.push\_back(currReview);  cin >> currRating;  }  }  *// Print all comments for reviews having the given rating*  void Reviews::PrintCommentsForRating(int currRating) const {  Review currReview;  unsigned int i;  for (i = 0; i < reviewList.size(); ++i) {  currReview = reviewList.at(i);  if (currRating == currReview.GetRating()) {  cout << currReview.GetComment() << endl;  }  }  }  int Reviews::GetAverageRating() const {  int ratingsSum;  unsigned int i;  ratingsSum = 0;  for (i = 0; i < reviewList.size(); ++i) {  ratingsSum += reviewList.at(i).GetRating();  }  return (ratingsSum / reviewList.size());  } |
| Restaurant.h  #ifndef RESTAURANT\_H  #define RESTAURANT\_H  #include *<string>*  #include *"Reviews.h"*  class Restaurant {  public:  void SetName(std::string restaurantName);  void ReadAllReviews();  void PrintCommentsByRating() const;  private:  std::string name;  Reviews reviews;  };  #endif | Restaurant.cpp  #include *<iostream>*  #include *"Restaurant.h"*  using namespace std;  void Restaurant::SetName(string restaurantName) {  name = restaurantName;  }  void Restaurant::ReadAllReviews() {  cout << "Type ratings + comments. To end: -1" << endl;  reviews.InputReviews();  }  void Restaurant::PrintCommentsByRating() const {  int i;  cout << "Comments for each rating level: " << endl;  for (i = 1; i <= 5; ++i) {  cout << i << ":" << endl;  reviews.PrintCommentsForRating(i);  }  } |