Разбиение программы на модули (файлы)

Самые простые программы могут состоять из одной функции main. Чуть более сложные включают в себя другие функции. По мере возрастания сложности программы функций становится слишком много, в них становится тяжело ориентироваться. Для логического и физического разделения программы используется разбиение функций на отдельные модули по смысловому значению.

Рассмотрим программу, которая осуществляет перевод чисел из двоичной системы счисления в десятичную, из десятичной в двоичную, из десятичной в восьмеричную, из восьмеричной в десятичную, из десятичной в шестнадцатеричную, из шестнадцатеричной в десятичную.

Создадим программу, которая использует дополнительно три модуля, содержащих функции для работы с числами в разных системах счисления.

В языке C/C++ модуль логически состоит из двух файлов - файла с исходным кодом (source file) и заголовочного файла (header file):

- файл с исходным кодом (module.cpp) включает в себя определения функций, а также определения глобальных переменных и констант (если они есть); в первой строчке такого файла обычно подключается заголовочный файл того же модуля: **#include** "module.h"

- заголовочный файл (module.h) включает в себя прототипы функций, определения констант, объявления глобальных переменных - но только для тех элементов модуля, о которых должны знать другие модули.

Если какой-либо модуль использует данный, необходимо подключить его заголовочный файл:

#include "module.h"

Главный модуль программы обычно содержит только функцию main и не имеет заголовочного файла (поскольку функция main не используется в других модулях).

Некоторые модули включают только заголовочный файл – например, содержащий определения глобальных констант

Построение (сборка, build) состоит из 3 основных этапов:

1) на этапе препроцессинга (preprocessing) директивы препроцессора (например, #include) заменяются содержимым указанного в них заголовочного файла, в результате файл с исходным кодом дополняется прототипами указанных там функций и объявлениями глобальных переменных – за счет этого в файле можно вызывать указанные функции (прототип впереди) и использовать глобальные переменные (объявление впереди). Препроцессор может создавать на диске временные файлы, которые, однако, удаляются после окончания сборки

2) на этапе компиляции (compiling) для каждого исходного файла составляются таблицы определенных в нем функций и глобальных переменных, все определенные функции переводятся на машинный язык (при переводе на машинный язык могут выявляться ошибки компиляции). Результатом работы компилятора являются файлы module.obj (объектные

файлы) для каждого использованного в программе модуля

3) на этапе связывания (linking) происходит привязка всех используемых (вызванных) функций и глобальных переменных к той таблице, в которой они определены; если определения не обнаружены ни в одной таблице, происходит ошибка связывания (unresolved external symbol …). Результатом связывания является файл module.exe – исполняемый файл

Что при сборке происходит с библиотечными функциями?

Прототипы функций находятся в заголовочных файлах (math.h, stdio.h, string.h и т.п.). Объектные файлы с уже переведенными на машинный язык определениями функций заранее собраны в библиотеки (файлы с расширением lib или dll):

- определения из статических библиотек (расширение lib) на этапе связывания добавляются в исполняемый файл программы
- определения из динамических библиотек (расширение dll) в исполняемый файл не добавляются; вместо этого в ссылке на соответствующую функцию указывается, что она находится в динамической библиотеке, и при ее вызове происходит обращение к библиотеке, поэтому, статические библиотеки нужны только на этапе связывания, а динамические – и на этапе исполнения программы

Таким образом, если программа будет использовать создаваемый нами модуль для работы с двоичными числами (например, bin.h), то нужно сделать следующее.

1) Создадим пустой проект.

2) Создадим нужные файлы: главный модуль main.cpp, заголовочный файл для работы с двоичной системой счисления bin.h и файл с исходным кодом для работы с двоичной системой счисления bin.cpp. Для добавления новый файлов в проект следует выбрать «Проект»-«Добавить новый элемент».

3) Так как главный модуль будет вызывать функции, описанный в модуле bin, подключим в главном модуле модуль bin.h:

#include “bin.h”

4) Модуль с исходным кодом bin.cpp будут реализовывать функции, прототипы которых реализованных в bin.h. Поэтому подключим в файле bin.cpp модуль bin.h:

#include “bin.h”

5) Запишем следующее в модуле bin.h

```
#ifndef _BIN_H_
#define _BIN_H_

// Перевод из десятичной системы счисления в двоичную
int dectobin(int);
// Перевод из двоичной системы счисления в десятичную
int bintodec(char*);

#endif
```

Директива #define позволяет определить переменную препроцессора (например, _BIN_H_)

Директива #ifndef (if not defined) позволяет выяснить, определена ли переменная препроцессора. Если она не определена, участок кода до директивы #endif вставляется в программу, если же определена – выбрасывается из нее

Нужны эти скобки для того, чтобы текст заголовочного файла не мог вставиться в исходный файл дважды. Почему это может произойти? Потому что файл bin.h подключаются в двух файлах. Т.е. в два файла добавляется программный код из bin.h. Но, т.к. в итоге программный код из всех модулей собирается в одном модуле, повторных участков кода быть не должно.

В данном случае определяется некоторая константа _BIN_H_ (имя может быть любое, рекомендуется использовать имя файла в каком-либо виде, например, как здесь). При первом подключении модуля bin.h при помощи макроса #ifndef проверяется, не определена ли такая константа. При первом подключении она не определена, поэтому выполняется код до макроса #endif, в котором описываются прототипы функций. Когда файл bin.h подключается второй раз, константа _BIN_H_ уже объявлена. Поэтому тело внутри #ifndef не выполняется и содержимое bin.h повторно не добавляется.

Закончите программу.

Программа должна содержать:

1) главный модуль – с функцией main. Главный модель демонстрирует работу всех функций во всех модулях, которые предстоит создать.

2) модуль bin из заголовочного файла и файла с исходным кодом – в заголовочном файле описаны прототипы функций bintodec – переводит из двоичной системы счисления в десятичную (принимает строку и возвращает число) и dectobin – переводит из десятичной системы счисления в довичную (принимает число и возвращает строку из 0 и 1); в файле с исходным кодом – реализация этих функций.

3) модуль hex из заголовочного файла и файла с исходным кодом – в заголовочном файле описаны прототипы функций hextodec – переводит из шестнадцатеричной системы счисления в десятичную (принимает строку и возвращает число) и dectohex – переводит из десятичной системы счисления в шестнадцатеричную (принимает число и возвращает строку) ; в файле с исходным кодом – реализация этих функций.

4) модуль oct из заголовочного файла и файла с исходным кодом – в заголовочном файле описаны прототипы функций octtodec – переводит из восьмеричной системы счисления в десятичную (принимает строку и возвращает число) и dectooct – переводит из десятичной системы счисления в восьмеричную (принимает число и возвращает строку); в файле с исходным кодом – реализация этих функций.