ционные технологии : материалы 84-й науч.-техн. конференции профессорско-преподавательского состава, научных сотрудников и аспирантов (с международным участием), Минск, 4-14 февраля 2020 года [Электронный ресурс] / отв. за издание И.В. Войтов; УО БГТУ. – Минск : БГТУ, 2020. – с. 79-80.

2. Jaber G, Patsei N., Rahal F., Abboud A. Naming and Routing Scheme for Data Content Objects in Information-Centric Network // 2020 Open Conference of Electrical, Electronic and Information Sciences (eStream): Proceedings of the Conference : April 30, 2020, Vilnius, Lithuania, IEEE -2020. P.93-97.

УДК 004.432.2

Ст. преп. А.С. Наркевич
(БГТУ, г. Минск)

C++20: МОДУЛИ

В 2022 году языком программирования года по версии TIOBE стал C++. На данный момент действует стандарт языка C++ 20. Стандарт C++23 официально еще не утвержден, но его уже можно опробовать на компиляторах GCC, Clang, MSVC [1].

Одним из главных нововведений стандарта C++20 являются модули, которые реализованы начиная с Visual Studio 2022 версии 17.1.

До модулей в C++ использовались заголовки – отдельные текстовые файлы с расширением .h или .hpp, при подключении которых содержимое файла копировалось в место их включения в программу. Возможные проблемы:

– неочевидный побочный эффект включения заголовочных файлов (в зависимости от порядка расположения два включаемых фрагмента могут влиять друг на друга);

– нарушение правила одного определения (one definition rule):

– функция или класс могут включаться в разные файлы .cpp, разные единицы трансляции, что может привести к нарушению правила одного определения;

– включения из заголовочного файла зависят от макросов, которые могут быть переопределены в момент включения;

– медленная компиляция: возникает, когда один и тот же заголовок целиком включается в разные единицы трансляции, компилятор компилирует его каждый раз;

– сложности в контроле того, что действительно нужно экспортировать, а что – нет (в единицу трансляции включается всё, что в указано в заголовке).

Модуль – это набор файлов исходного кода, которые компилируются независимо от единиц трансляции, которые их импортируют.

Модуль состоит из одного или нескольких файлов исходного кода, скомпилированных в двоичный файл. Двоичный файл описывает все экспортированные типы, функции и шаблоны в модуле. При импорте модуля, компилятор считывает его двоичный файл. Чтение двоичного файла выполняется быстрее, чем обработка файла заголовка. Кроме того, двоичный файл повторно используется компилятором при каждом импорте этого модуля. Модуль создается один раз, а потом используется повторно, время сборки сокращается.

В C++20 библиотеки и программы становятся компонентами [2].

Модули призваны улучшить:

– время сборки;

– изоляцию имен;

– обеспечение правила одного определения;

– отслеживание зависимостей (в одном файле заголовков может быть элемент из другого, занесенный случайным образом, из-за чего будут возникать зависимости);

– после компиляции модуля результаты сохраняются и могут использоваться компилятором при повторном импорте.

Рекомендуется для новых проектов использование модулей, а не файлов заголовков.

Начиная с Visual Studio 2022 версии 17.5, импорт стандартной библиотеки в качестве модуля является стандартизованным и полностью реализован в компиляторе MSVC.

Импорт стандартной библиотеки C++ в качестве модулей позволяет сократить время компиляции. Библиотека разделена на следующие именованные модули [3]:

– std.regex предоставляет содержимое заголовка <regex>;

– std.filesystem предоставляет содержимое заголовка <filesystem>;

– std.memory предоставляет содержимое заголовка <memory>;

– std.threading предоставляет содержимое заголовков <atomic>, <condition_variable>, <future>, <mutex>, <shared_mutex>и <thread>;

– std.core предоставляет все остальное в стандартной библиотеке C++.

Так как модули являются компонентом C++20, необходимо использовать параметр компилятора `/std:c++20` или `/std:c++latest`. Для использования модулей стандартной библиотеки необходимо объявить импорт в начале файла исходного кода и скомпилировать программу с параметрами `/EHsc` и `/MD`.

Пример:

Без модулей	С использованием модулей

`#include <iostream>`
`int main()`
`{`
`std::cout <<`
`"Hello World!\n";`
`}`
	`import std.io;`
	`int main()`
	`{`
	`std::cout <<`
	`"Hello World!\n";`
	`}`

С помощью ключевого слова `import` можно получать доступ к другим единицам трансляции [4]:

– модуль std.io компилируется отдельно;

– после компиляции модуля результаты сохраняются в двоичном файле, который описывает все экспортированные типы, функции, шаблоны;

– файл используется компилятором каждый раз, когда модуль импортируется в проект.

Интерфейс модуля экспортирует имя модуля и все пространства имен, типы, функции и т. д., составляющие открытый интерфейс модуля. Реализация модуля определяет элементы, экспортируемые модулем. В простейшей форме модуль может состоять из одного файла, объединяющего интерфейс модуля и реализацию. Реализацию также можно поместить в один или несколько отдельных файлов модуля. Для больших модулей можно разделить части модуля на подмодули, называемые секциями.

```
// Example.ixx - файл интерфейса модуля с именем Example
export module Example; // файл основного интерфейса для модуля
#define ANSWER 42
namespace Example_NS // пространство имен Example_NS
{
   int f_internal() {
        return ANSWER;
      }
   export int f() { // отображается при импорте
      return f_internal();
   }
}
```

```
// MyProgram.cpp
import Example; // размещается только в глобальной области
import std.core;
using namespace std;
int main()
{
   cout << "The result of f() is " << Example_NS::f() <<
endl; // 42
   // int i = Example_NS::f_internal(); // C2039
   // int j = ANSWER; //C2065
}
```

Модули нарушают несколько принципов C++ [5]:

1. Принцип независимости сборки. Раньше программа на C++ состояла из разных единиц трансляции – файлов .cpp. Каждый из них можно было компилировать независимо друг от друга, а потом уже компоновать все объектные модули, входящие в ее состав. При использовании модулей порядок компиляции не является произвольным. Файл нельзя собрать, пока не предкомпилированы модули, от которых он зависит. Поэтому собрать модуль не возможно, если в каком-то зависимом модуле содержится ошибка.

2. Принцип гомогенности кода. Директиву `#include` можно писать в любом месте программы. Но обычно её пишут в начале, это договорённость, а не правило. Новый стандарт вводит преамбулу. Только в ней могут располагаться импорты модулей. Как только преамбула закончилась, писать `import` уже нельзя. У файла кода появляется структура. Кроме того, перед преамбулой возможна предпреамбула – так называемый `Global module fragment`. В нём могут располагаться только директивы препроцессора. Они допускают использование `#include`, а значит, по факту – всё что угодно.

Основными достоинствами модулей являются:

– импорт только нужных имён;

– ускорение процесса сборки;

– больше не приходится писать `.cpp` и `.h` отдельно.

Модули могут зависеть друг от друга, в этом случае, независимый модуль необходимо скомпилировать первым, затем зависимый и далее файлы, которые их подключают. Комитет использовал такой подход для автоматического получения аналога заголовочного файла, который содержит все необходимое для файлов, которые их импортируют.

ЛИТЕРАТУРА

1. C++ compiler support [Электронный ресурс] – Режим доступа: https://en.cppreference.com/w/cpp/compiler_support/20 – (дата обращения: 10.01.2023г.).

2. Руководство по именованным модулям (C++) [Электронный ресурс] – Режим доступа: https://learn.microsoft.com/ru-ru/cpp/cpp/modules-cpp?view= msvc-170 (дата обращения: 10.01.2023г.).

3. Обзор модулей в C++ [Электронный ресурс] – Режим доступа: https://docs.microsoft.com/ru-ru/cpp/cpp/tutorial-named-modules-cpp?view=msvc-170 (дата обращения: 10.01.2023г.).

4. C++20 в 2020: Модули [Электронный ресурс] – Режим доступа: https://habr.com/ru/company/otus/blog/575954/ (дата обращения: 10.01.2023г.).

5. Стандарт C++20: обзор новых возможностей C++ [Электронный ресурс] – Режим доступа: https://habr.com/ru/company/yandex_praktikum/ blog/554874/ (дата обращения: 10.01.2023г.).

УДК 004.415.538

Ассист. А.Н. Мущук; ассист. А.С. Пахолко
(БГТУ, г. Минск)

МЕТОДЫ ИССЛЕДОВАТЕЛЬСКОГО ТЕСТИРОВАНИЯ

Большинство методов тестирования предполагают, что предварительное планирование и внесение изменений в следующую итерацию будут производиться отдельно от процесса тестирования, а сам процесс прописан заранее. Условно, можно назвать это сценарным подходом. Однако в 2009 году Джеймс Виттакер предложил подход [1], при котором создание тестов, а также их прохождение и корректировка происходят в одно время. Называется он исследовательское тестирование. Само собой, несознательно его использовали и до этого, однако именно Виттакер более-менее систематизировал, и дал внятное описание «турам» в исследовательском тестировании.

Концепция достаточно проста: тестировщик и приложение метафорично представляют из себя туриста и город. Так как обычно у туристов мало времени, они концентрируются на каких-то конкретных объектах или достопримечательностях. Аналогично, в исследовательском тестировании приложение разбивается на «районы», которые можно проверять различными методами. Всего выделяют шесть районов: деловой центр (Tours of the Business District), исторический (Tours Through the Historical District), район развлечений (Tours