Белорусский государственный технологический университет

Факультет информационных технологий

Кафедра программной инженерии

 Реферат

По дисциплине «Основы алгоритмизации и программирования»

На тему «Раннее и позднее связывание. Создание и подключение собственных статических библиотек»

Выполнила:

Студент(ка) 1 курса 7 группы

Подшиваленко Диана Игоревна

Проверил:

Белодед Николай Иванович

2023, Минск

**Содержание**

1. [Предисловие . . . 3](#Предисловие)
2. [Связывание . . . 4](#Связывание)
   1. [Раннее связывание . . . 5](#Раннее_связывание)
   2. [Позднее связывание . . . 6](#Позднее_связывание)
3. [Создание собственных библиотек. Пример создания и подключения статической библиотеки . . . 7](#Создание_собственных_библиотек)
4. [Выводы . . . 11](#Выводы)
5. **Предисловие**

В данном реферате представлен обзор раннего и позднего связывания с примерами, различия между ними, плюсы и минусы каждого. А также он содержит информацию о том, как с помощью интегрированной среды разработки Visual Studio можно создать собственную библиотеку C++

1. **Связывание**

Выполнение программы в языке C++ происходит последовательно, строка за строкой, начиная с функции main(). Когда компилятор встречает вызов функции, то точка выполнения переходит к началу кода вызываемой функции. Откуда компилятор знает, что это нужно сделать?

Когда программа компилируется, компилятор преобразует каждую инструкцию в нашей программе на C++ в одну или несколько строк машинного кода. Каждой такой строке дается собственный уникальный адрес в последовательности адресов. То же самое и с функциями – когда встречается функция, она преобразуется в машинный код и получает следующий доступный адрес. Таким образом, каждая функция получает уникальный адрес.

***Связывание*** — это процесс, который используется для конвертации идентификаторов (таких как имена переменных или функций) в адреса.

Имеются два термина, часто используемых, когда речь заходит об объектно-ориентированных языках программирования: раннее(статическое) и позднее(динамическое) связывание. По отношению к С++ эти термины соответствуют событиям, которые возникают на этапе компиляции и на этапе исполнения про­граммы соответственно.

**2.1 Раннее связывание**

В терминах объектно-ориентированного программирования ***раннее связывание*** (также называемое **статическим связыванием**) означает, что компилятор (или компоновщик) может напрямую связать имя идентификатора (например, имя функции или переменной) с машинным адресом. Каждая функция имеет уникальный адрес, поэтому, когда компилятор (или компоновщик) сталкивается с вызовом функции, он заменяет вызов функции инструкцией машинного кода, которая сообщает процессору перейти к адресу функции.В качестве примеров раннего связывания можно ука­зать стандартные вызовы функций, вызовы перегруженных функций и перегруженных операто­ров.

Рассмотри простой пример программы нахождения суммы, в которой используется раннее связывание:

#include <iostream>

#include <iomanip>

#include <Windows.h>

#include <stdio.h>

#include <conio.h>

using namespace std;

int summa(int a, int b) {

return a + b;

}

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

int x, y;

cout << "Введите число ";

cin >> x;

cout << "Введите другое число ";

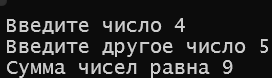
cin >> y;

cout << "Сумма чисел равна " << summa(x, y); // прямой вызов функции. Используем раннее связывание

return 0;

}

Результат выполнения программы:



Поскольку summa(x, y) является прямым вызовом функции, то компилятор будет использовать раннее связывание. Он заменит вызов summa(x, y) инструкцией, которая сообщит процессору перейти к адресу summa() .

**2.2 Позднее связывание**

В некоторых программах невозможно знать наперёд, какая функция будет вызываться первой. В таком случае используется ***позднее связывание*** (или **динамическая привязка**). Позднее связывание означает, что объект связывается с вызовом функции только во время ис­полнения программы, а не раньше. В языке C++ для выполнения позднего связывания используются [**указатели на функции**](https://ravesli.com/urok-104-ukazateli-na-funktsii/). Вызов функции через указатель на функцию также известен как косвенный вызов функции.

Рассмотри простой пример программы, в которой используется указатель на функцию для её вызова:

#include <iostream>

#include <iomanip>

#include <Windows.h>

#include <stdio.h>

#include <conio.h>

using namespace std;

int summa(int a, int b) {

return a + b;

}

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

int x, y;

cout << "Введите число ";

cin >> x;

cout << "Введите другое число ";

cin >> y;

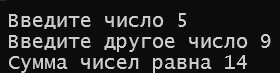
int(\*pFunc)(int, int) = summa; // мы создаём указатель на нашу фунцию для вычисления суммы

cout << "Сумма чисел равна " << pFunc(x, y); // вызываем функцию через указатель на функцию

return 0;

}

Результат выполнения программы:



В этом примере вместо прямого вызова функции summa() мы установили указатель **pFunc** так, чтобы он указывал на эту функцию. Затем мы вызываем функцию через этот указатель.

1. **Создание собственных библиотек С++**

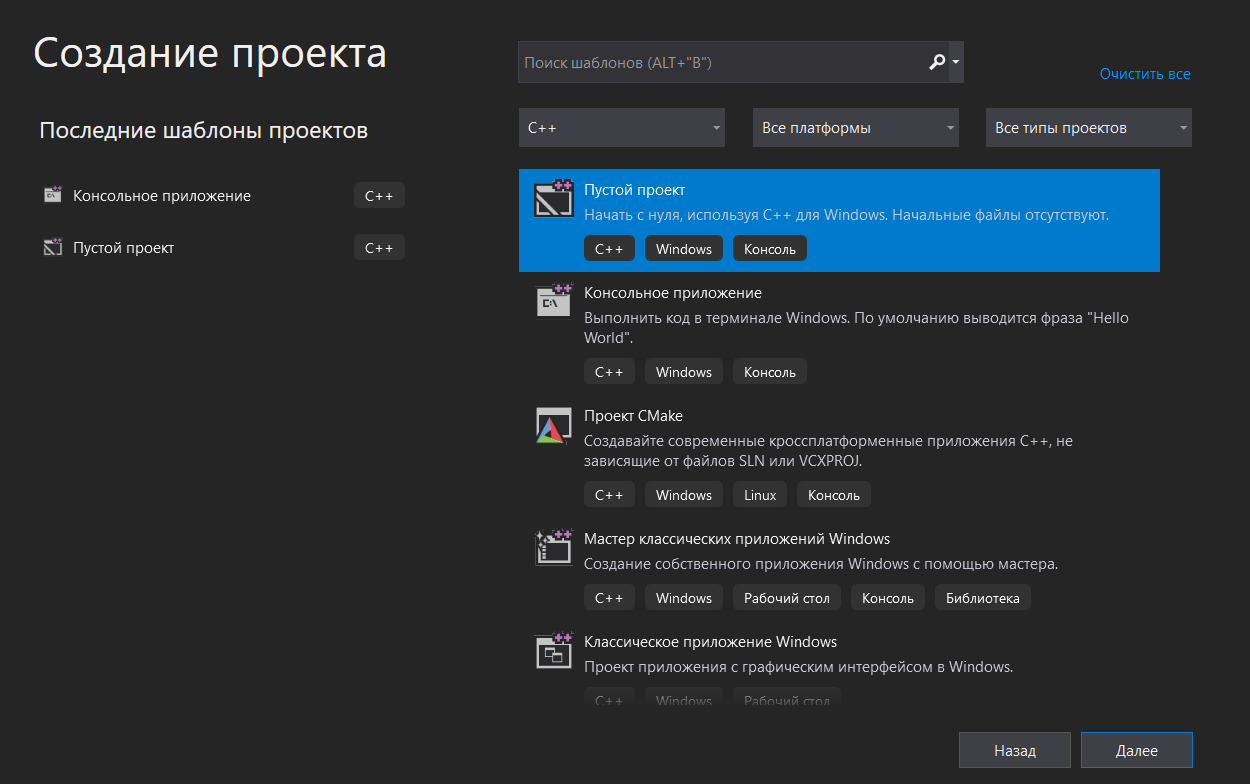
В языке программирования C++ код библиотек представляет собой функции, размещенные в файлах, которые скомпилированы в объектные файлы, а те, в свою очередь, объединены в библиотеки. В одной библиотеке объединяются функции, решающие определенный тип задач.

У каждой библиотеки должен быть свой заголовочный файл, в котором должны быть описаны прототипы (объявления) всех функций, содержащихся в этой библиотеке. С помощью заголовочных файлов мы "сообщаем" нашему программному коду, какие библиотечные функции есть и как их использовать.

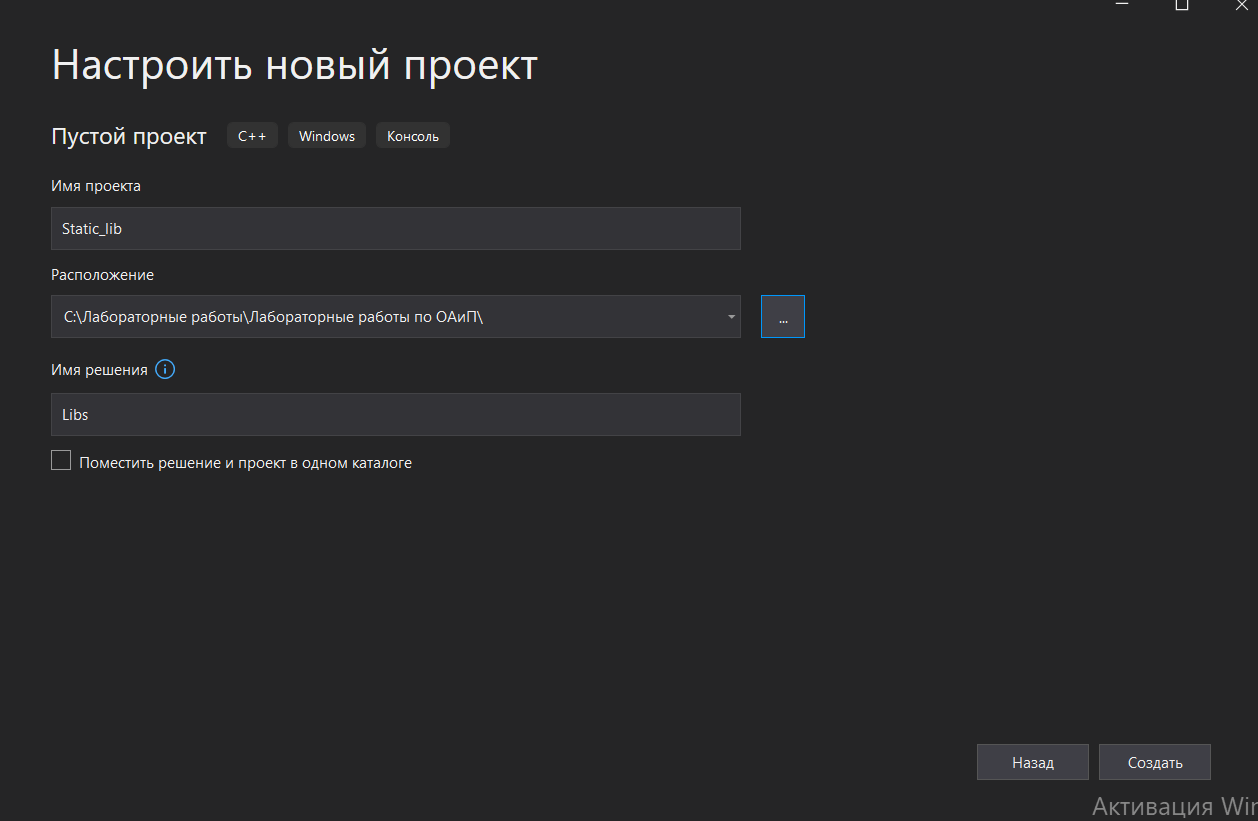
Библиотеки бывают двух видов — **статические** и **динамические**. Код **статических библиотек** при компиляции полностью входит в состав исполняемого файла, что делает программу легко переносимой. Код **динамических библиотек** не входит в исполняемый файл, последний содержит лишь ссылку на библиотеку. Если динамическая библиотека будет удалена или перемещена в другое место, то программа работать не будет. С другой стороны, использование динамических библиотек позволяет сократить размер исполняемого файла. Также если в памяти находится две программы, использующие одну и туже динамическую библиотеку, то последняя будет загружена в память лишь единожды.

Далее мы рассмотрим, как можно создать и подключить статическую библиотеку с помощью интегрированной среды разработки Visual Studio.

**1)**Для начала создадим пустой проект.

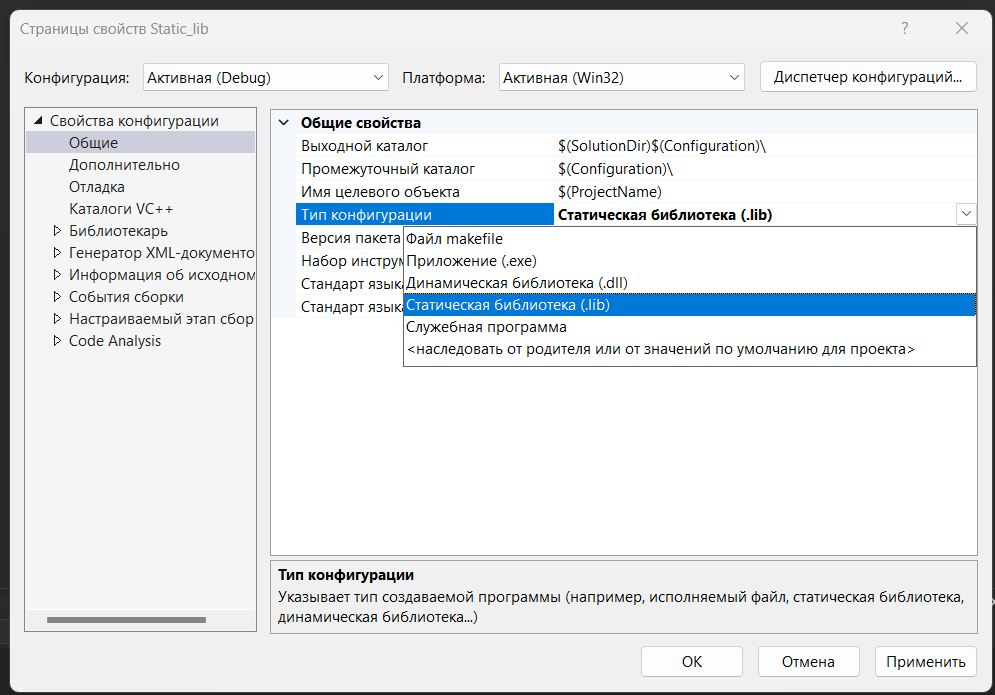


Пусть он будет называться **Static\_lib**, а имя решения будет **Libs.**

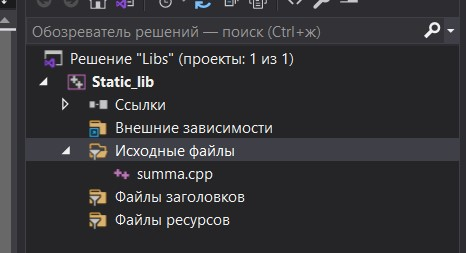
****

**2)** Далее мы должны щёлкнуть правой кнопкой мыши по нашему проекту и в выпадающем меню выбрать пункт **Свойства.**

**3)** Во вкладке **Свойства конфигурации** выбрать пункт **Общие** и в поле **Тип конфигурации** выбрать пункт **Статическая библиотека (.lib)**

****

**4)** Создаём файл с расширением **(.cpp),** где будет храниться наша функция и пишем её (в данном примере я создаю функцию с именем **summa()**, которая в качестве входных данных принимает два целых числа и возвращает результатом их сумму).



**Функция**

int summa(int a, int b) {

return a + b;

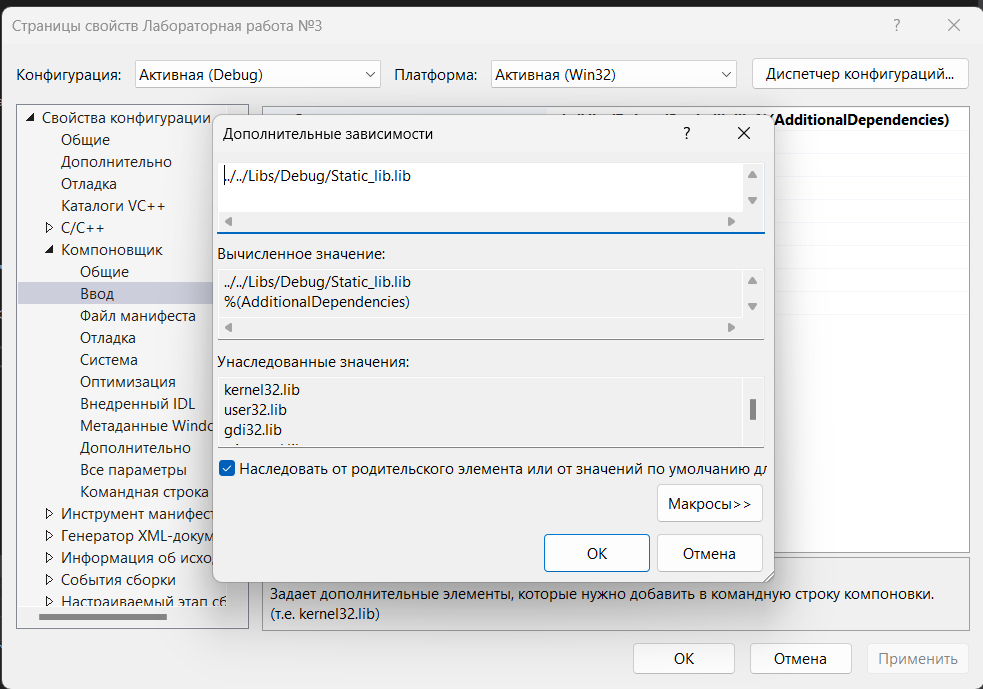
}

**5)** Далее создаём заголовочный файла с расширением **(.h)**, в который добавляем нашу функцию. После этого собираем наш проект (библиотеку).

#pragma once

int summa(int, int);

Наша библиотека создана. Теперь мы можем подключать её к нашим проектам. Но ещё мы должны снова зайти во вкладку **Свойства**, затем во вкладку **Компоновщик** пункт **Ввод** и в поле **Дополнительные зависимости** указать путь к нашей библиотеке.



#include <iostream>

#include <iomanip>

#include <Windows.h>

#include <stdio.h>

#include <conio.h>

#include "../../Libs/Static\_lib/Header.h"// указываем путь нашего заголовочного файла

using namespace std;

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

int x, y;

cout << "Введите число ";

cin >> x;

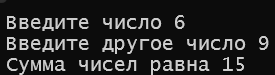
cout << "Введите другое число ";

cin >> y;

cout << "Сумма чисел равна " << summa(x, y);

return 0;

}

****Результат выполнения:

**4.Выводы**

Подводя итоги, можно выделить следующие отличия связываний:

1. Статическая привязка происходит во время компиляции, а динамическая привязка — во время выполнения. Следовательно, их также называют ранним и поздним связыванием соответственно.
2. При статической привязке определение функции и вызов функции связываются во время компиляции, тогда как при динамической привязке вызовы функций не связаны до времени выполнения.
3. Статическая привязка происходит, когда вся информация, необходимая для вызова функции, доступна во время компиляции. Динамическое связывание происходит, когда компилятор не может определить всю информацию, необходимую для вызова функции, во время компиляции.
4. Статическая привязка может быть достигнута с помощью обычных вызовов функций, перегрузки функций и перегрузки операторов, тогда как динамическая привязка может быть достигнута с использованием указателей на функцию.
5. Поскольку вся информация, необходимая для вызова функции, доступна до выполнения, статическая привязка приводит к более быстрому выполнению программы. В отличие от неё динамическая привязка выполняется медленнее.
6. Основным преимуществом динамического связывания является его гибкость, поскольку одна функция может обрабатывать различные типы объектов во время выполнения. Это значительно уменьшает размер кодовой базы, а также делает исходный код более читабельным.

Какое именно связывание должна использовать программа, зависит от её предназначения. Фактически достаточно сложные программы используют оба вида связывания.

Библиотеки позволяют использовать разработанный ранее программный код в различных программах. Таким образом, программист может не разрабатывать часть кода для своей программы, а воспользоваться тем, что входит в состав библиотек.