Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Системное программирование

Студент: Лопатнюк П.В.

ФИТ 3 курс 1 группа

Преподаватель: Бернацкий П.В.

Минск 2024

**Лабораторная работа №3**

**Библиотеки**

**Цель работы:** Получение практических навыков в работе со статическими и динамическими библиотеками.

**Функционал для библиотек:**

Любая библиотека из данной лабораторной работы должна содержать следующий функционал:

* Итеративную функцию бинарного поиска;
* Рекурсивную функцию бинарного поиска;
* Массив целых чисел размером 1024 элемента.

Реализацию функций можно взять [отсюда](https://rosettacode.org/wiki/Binary_search" \l "C), а массив сгенерировать [тут](https://onlinetools.com/integer/create-integer-array).

**Постановка задачи для Windows:**

**Проект <Ваши Инициалы>Lib:**

Должен содержать код для создания статической библиотеки с заданным функционалом (см. выше). Код должен быть кроссплатформенным, так как будет применяться и для Linux/macOS (Для тех кто не берёт код из указанного выше источника).

Заголовочный файл библиотеки **MyLib.h**

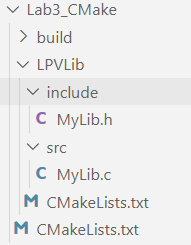
|  |
| --- |
| #ifndef MYLIB\_H  #define MYLIB\_H  #define LIB\_ARRAY\_SIZE 1024  #include <stdio.h>  extern const int lib\_array[LIB\_ARRAY\_SIZE];  int lpv\_bsearch(int\* a, int n, int x);  int lpv\_bsearch\_r(int\* a, int x, int i, int j);  #endif |

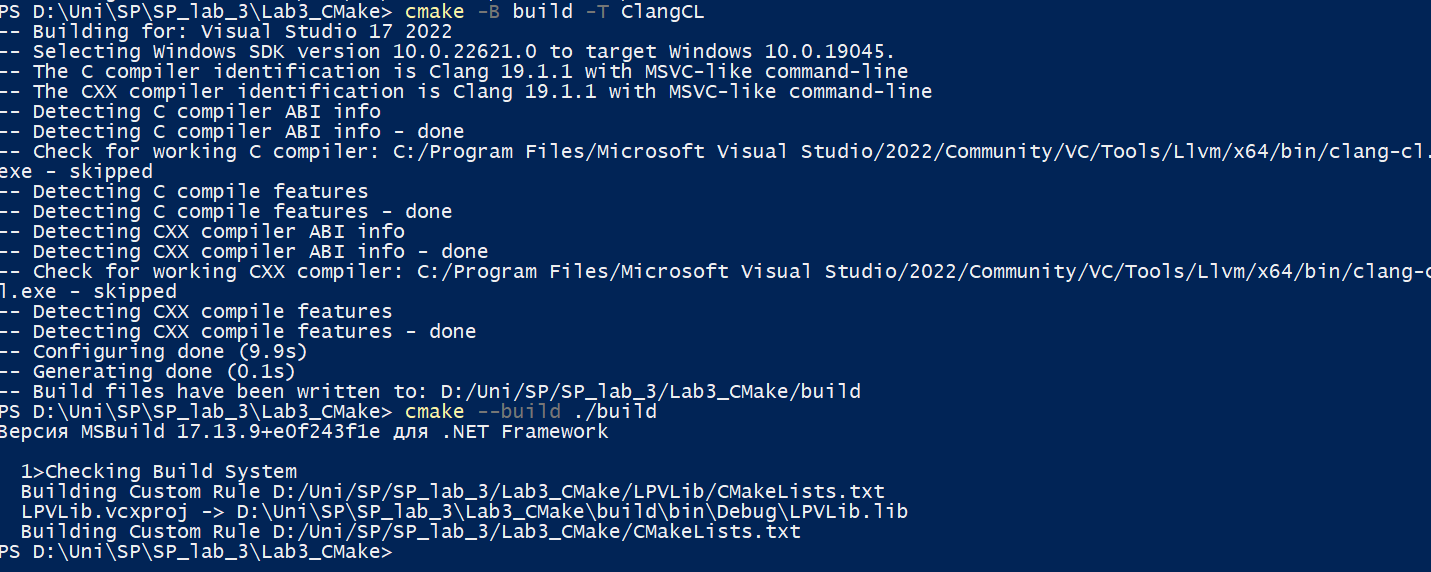
Основной файл **MyLib.c**

|  |
| --- |
| #include "MyLib.h"  int lpv\_bsearch(int\* a, int n, int x) {      int i = 0, j = n - 1;      while (i <= j) {          int k = i + ((j - i) / 2);          if (a[k] == x) {              return k;          }          else if (a[k] < x) {              i = k + 1;          }          else {              j = k - 1;          }      }      return -1;  }  int lpv\_bsearch\_r(int\* a, int x, int i, int j) {      if (j < i) {          return -1;      }      int k = i + ((j - i) / 2);      if (a[k] == x) {          return k;      }      else if (a[k] < x) {          return lpv\_bsearch\_r(a, x, k + 1, j);      }      else {          return lpv\_bsearch\_r(a, x, i, k - 1);      }  }  const int lib\_array[LIB\_ARRAY\_SIZE] = {       0,   1,   2,  … , 1022, 1023  }; |

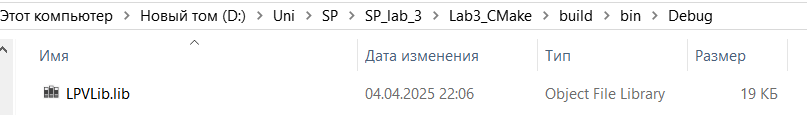
Сборка CMake:

|  |
| --- |
| cmake\_minimum\_required(VERSION 3.28)  include\_directories(${CMAKE\_SOURCE\_DIR}/LPVLib/include/)  add\_library(LPVLib STATIC src/MyLib.c) |





Получаем файл библиотеки:



**Проект Lab-03a:**

Разработать клиентское приложение которое будет использовать статическую библиотеку. Код также должен быть кроссплатформенным, так как и в этом случае, код будет применяться для Linux/macOS. Приложение должно быть консольным и может быть вызывано в виде:

***Lab-03a <Имя вызываемой функции> [Искомое число]***

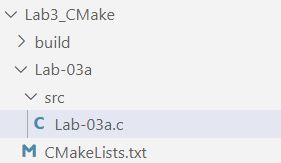
Если искомое число не задано, то оно запрашивается для ввода с консоли. В результате выполнения этого приложения должен быть выведен один из следующих результатов:

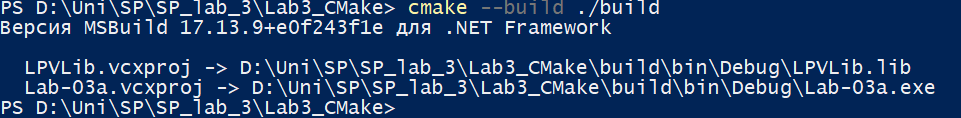
* Не указана вызываемая функция!
* <Имя вызванной функции>: Заданное число не найдено!
* <Имя вызванной функции>: Число <Х> найдено на позиции <Y>!

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <string.h>  #include "MyLib.h"  int main(int argc, char\* argv[]) {        if (argc < 2) {          printf("The function being called is not specified\n");          return 1;      }      char\* function\_name = argv[1];      int searchVal = -1;      if (argc > 2) {          searchVal = atoi(argv[2]);      }      else {          printf("Enter the desired number: ");          if (scanf("%d", &searchVal) != 1) {              printf("Invalid input. Expected a number.\n");              return 1;          }      }      int index = -1;      if (strcmp(function\_name, "bsearch") == 0) {          index = lpv\_bsearch(lib\_array, LIB\_ARRAY\_SIZE - 1, searchVal);      }      else if (strcmp(function\_name, "bsearch\_r") == 0) {          index = lpv\_bsearch\_r(lib\_array, searchVal, 0, LIB\_ARRAY\_SIZE - 1);      }      else {          printf("Unknown function: %s\n", function\_name);          return 1;      }      if (index == -1) {          printf("%s: The specified number was not found\n", function\_name);      }      else {          printf("%s: The number %d found in the position %d\n", function\_name, searchVal, index);      }      return 0;  } |

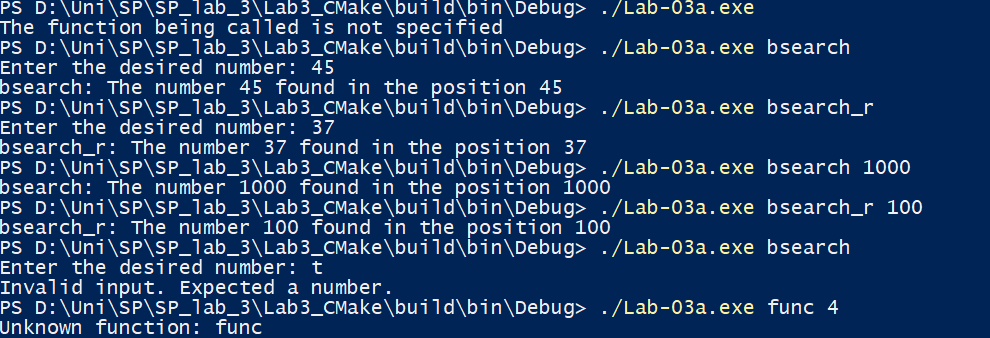
Сборка CMake:

|  |
| --- |
| cmake\_minimum\_required(VERSION 3.28)  include\_directories(${PROJECT\_SOURCE\_DIR}/LPVLib/include)  add\_executable(Lab-03a src/Lab-03a.c)  target\_link\_libraries(Lab-03a LPVLib) |



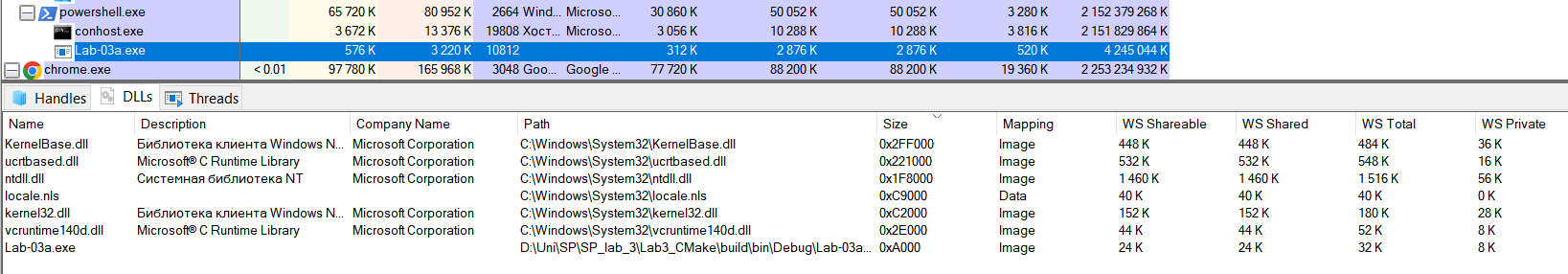


Результат:



Проверить факт подключения библиотеки через Process Explorer.

Вопрос: Является ли это возможным? Почему?



**Проект <Ваши Инициалы>DLib:**

Разработать динамическую библиотеку на языке C с заданным функционалом (см. выше). Также должна быть реализована функция DllMain c логированием событий в некоторый файл (путь к файлу должен быть абсолютным).

Экспорт функций и переменных организовать через .def файл так, чтобы:

* Одна из функций экспортировалась без имени;
* Одна из функций экспортировалась с именем отличающимся от оригинального.

MyLibD.h

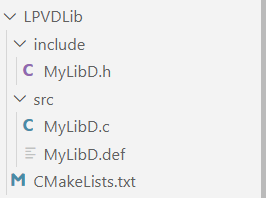
|  |
| --- |
| #ifndef MYLIB\_H  #define MYLIB\_H  #define LIB\_ARRAY\_SIZE 1024  extern const int lib\_array[LIB\_ARRAY\_SIZE];  int lpv\_bsearch(int\* a, int n, int x);  int lpv\_bsearch\_r(int\* a, int x, int i, int j);  #endif |

MyLibD.с

|  |
| --- |
| #include "MyLibD.h"  #include <windows.h>  #include <stdio.h>  #include <errno.h>  #include <time.h>  #define LOG\_FILE "D:\\Uni\\SP\\SP\_lab\_3\\Lab3\_CMake\\logs\\MyLibDlogs.txt"  void log\_message(const char\* message) {      if (GetDriveType("D:\\") == DRIVE\_NO\_ROOT\_DIR) {          OutputDebugStringA("D: drive not available");          return;      }      FILE\* logFile = fopen(LOG\_FILE, "a");      if (!logFile) {          OutputDebugStringA("Failed to open log file");          return;      }      time\_t now;      time(&now);      struct tm\* timeinfo = localtime(&now);        fprintf(logFile, "[%02d:%02d:%02d] ",             timeinfo->tm\_hour,             timeinfo->tm\_min,             timeinfo->tm\_sec);      fprintf(logFile, "%s\n", message);      fclose(logFile);  }  int lpv\_bsearch(int\* a, int n, int x) {      log\_message("lpv\_bsearch called");        int i = 0, j = n - 1;      while (i <= j) {          int k = i + ((j - i) / 2);          if (a[k] == x) {              log\_message("lpv\_bsearch: element found");              return k;          }          else if (a[k] < x) {              i = k + 1;          }          else {              j = k - 1;          }      }        log\_message("lpv\_bsearch: element not found");      return -1;  }  int lpv\_bsearch\_r(int\* a, int x, int i, int j) {      static int first\_call = 1;      if (first\_call) {          log\_message("lpv\_bsearch\_r called");          first\_call = 0;      }        if (j < i) {          log\_message("lpv\_bsearch\_r: element not found");          first\_call = 1;          return -1;      }        int k = i + ((j - i) / 2);      if (a[k] == x) {          log\_message("lpv\_bsearch\_r: element found");          first\_call = 1;          return k;      }      else if (a[k] < x) {          return lpv\_bsearch\_r(a, x, k + 1, j);      }      else {          return lpv\_bsearch\_r(a, x, i, k - 1);      }  }  BOOL APIENTRY DllMain(HMODULE hModule, DWORD ul\_reason\_for\_call, LPVOID lpReserved) {      switch (ul\_reason\_for\_call) {          case DLL\_PROCESS\_ATTACH:              log\_message("DLL loaded");              break;            case DLL\_PROCESS\_DETACH:              log\_message("DLL unloaded");              break;            case DLL\_THREAD\_ATTACH:              log\_message("Thread attached");              break;            case DLL\_THREAD\_DETACH:              log\_message("Thread detached");              break;      }        return TRUE;  }  const int lib\_array[LIB\_ARRAY\_SIZE] = {      0,   1,   2, ..., 1023  }; |

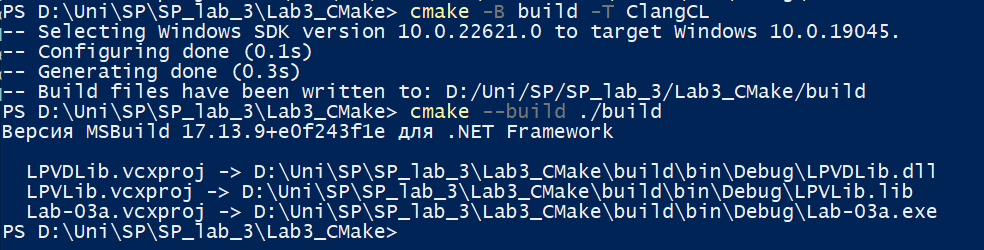
MyLibD.def

|  |
| --- |
| LIBRARY LPVDLib  EXPORTS  iterative\_bsearch = lpv\_bsearch @1  lpv\_bsearch\_r @2 NONAME  lib\_array @3 DATA |



Сборка:

|  |
| --- |
| cmake\_minimum\_required(VERSION 3.28)  include\_directories(${CMAKE\_SOURCE\_DIR}/LPVDLib/include)  add\_library(LPVDLib SHARED src/MyLibD.c)  target\_link\_options(LPVDLib PRIVATE "/DEF:${CMAKE\_SOURCE\_DIR}/LPVDLib/src/MyLibD.def") |



**Проект <Ваши Инициалы>DLib++:**

Разработать динамическую библиотеку на языке C++ с заданным функционалом (см. выше). Также должна быть реализована функция DllMain c логированием событий в некоторый файл (путь к файлу должен быть абсолютным).

Экспорт функций и переменных организовать через **\_\_declspec(dllexport)**. Одной из функций указать модификатор **extern “C”**.

MyLibDD.hpp

|  |
| --- |
| #ifndef MYLIBDD\_PP\_H  #define MYLIBDD\_PP\_H  #define LIB\_ARRAY\_SIZE 1024  #ifdef \_WIN32      #define DLL\_EXPORT \_\_declspec(dllexport)  #else      #define DLL\_EXPORT  #endif  #ifdef \_\_cplusplus  extern "C" {  #endif  extern "C" DLL\_EXPORT const int lib\_array[LIB\_ARRAY\_SIZE];  extern "C" DLL\_EXPORT int lpv\_bsearch(int\* a, int n, int x);  #ifdef \_\_cplusplus  }  DLL\_EXPORT int lpv\_bsearch\_r(int\* a, int x, int i, int j);  #endif  #endif |

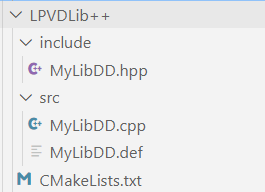
MyLibDD.cpp

|  |
| --- |
| #include "MyLibDD.hpp"  #include <windows.h>  #include <stdio.h>  #include <errno.h>  #include <time.h>  #define LOG\_FILE "D:\\Uni\\SP\\SP\_lab\_3\\Lab3\_CMake\\logs\\MyLibDDlogs.txt"  void log\_message(const char\* message) {      if (GetDriveType("D:\\") == DRIVE\_NO\_ROOT\_DIR) {          OutputDebugStringA("D: drive not available");          return;      }      FILE\* logFile = fopen(LOG\_FILE, "a");      if (!logFile) {          OutputDebugStringA("Failed to open log file");          return;      }      time\_t now;      time(&now);      struct tm\* timeinfo = localtime(&now);        fprintf(logFile, "[%02d:%02d:%02d] ",             timeinfo->tm\_hour,             timeinfo->tm\_min,             timeinfo->tm\_sec);      fprintf(logFile, "%s\n", message);      fclose(logFile);  }  int lpv\_bsearch(int\* a, int n, int x) {      log\_message("lpv\_bsearch called");        int i = 0, j = n - 1;      while (i <= j) {          int k = i + ((j - i) / 2);          if (a[k] == x) {              log\_message("lpv\_bsearch: element found");              return k;          }          else if (a[k] < x) {              i = k + 1;          }          else {              j = k - 1;          }      }        log\_message("lpv\_bsearch: element not found");      return -1;  }  int lpv\_bsearch\_r(int\* a, int x, int i, int j) {      static int first\_call = 1;      if (first\_call) {          log\_message("lpv\_bsearch\_r called");          first\_call = 0;      }        if (j < i) {          log\_message("lpv\_bsearch\_r: element not found");          first\_call = 1;          return -1;      }        int k = i + ((j - i) / 2);      if (a[k] == x) {          log\_message("lpv\_bsearch\_r: element found");          first\_call = 1;          return k;      }      else if (a[k] < x) {          return lpv\_bsearch\_r(a, x, k + 1, j);      }      else {          return lpv\_bsearch\_r(a, x, i, k - 1);      }  }  BOOL APIENTRY DllMain(HMODULE hModule, DWORD ul\_reason\_for\_call, LPVOID lpReserved) {      switch (ul\_reason\_for\_call) {          case DLL\_PROCESS\_ATTACH:              log\_message("DLL loaded");              break;            case DLL\_PROCESS\_DETACH:              log\_message("DLL unloaded");              break;            case DLL\_THREAD\_ATTACH:              log\_message("Thread attached");              break;            case DLL\_THREAD\_DETACH:              log\_message("Thread detached");              break;      }        return TRUE;  }  const int lib\_array[LIB\_ARRAY\_SIZE] = {      0,   1,   2,   ... , 1023  }; |

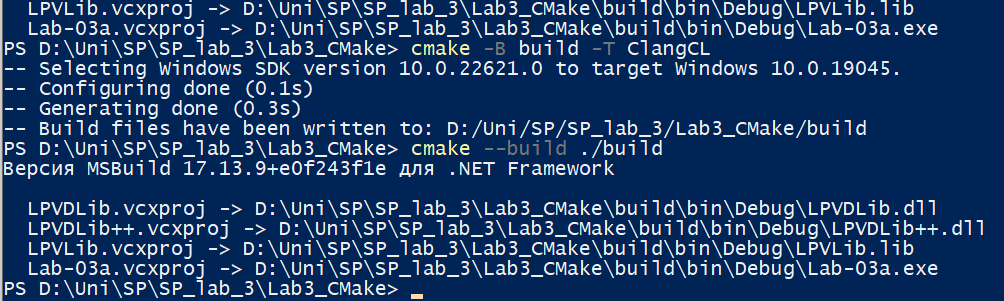
MyLibDD.def

|  |
| --- |
| LIBRARY MyLibDD  EXPORTS  iterative\_bsearch = lpv\_bsearch @1  recursive\_bsearch = lpv\_bsearch\_r @2  lib\_array @3 |

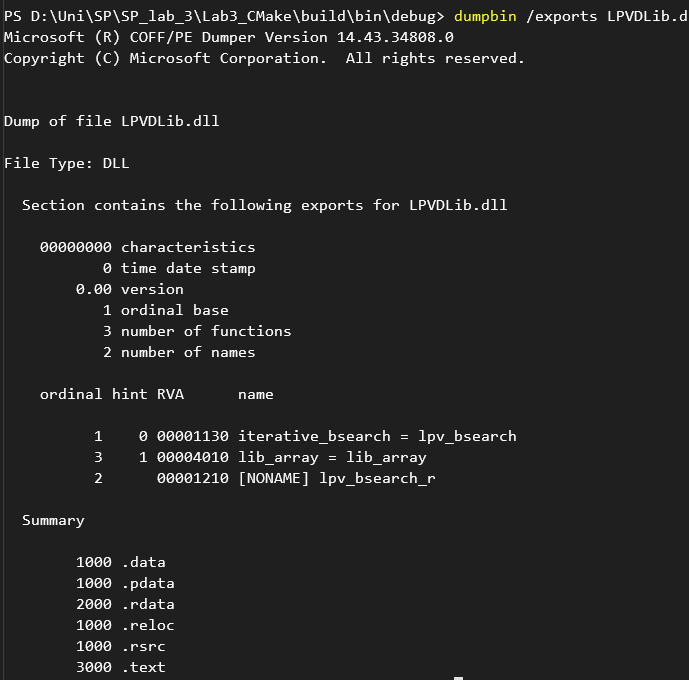
Сборка:

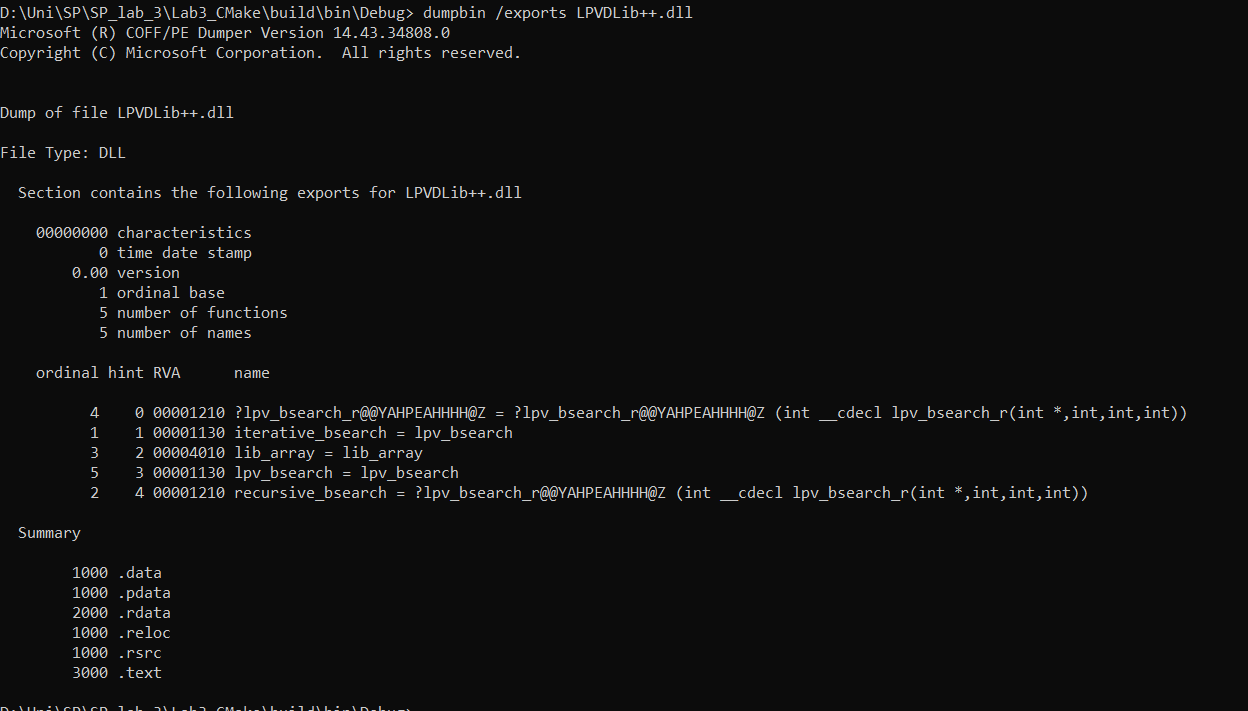


|  |
| --- |
| cmake\_minimum\_required(VERSION 3.28)  include\_directories(${CMAKE\_SOURCE\_DIR}/LPVDLib++/include)  add\_library(LPVDLib++ SHARED src/MyLibDD.cpp)  target\_link\_options(LPVDLib++ PRIVATE "/DEF:${CMAKE\_SOURCE\_DIR}/LPVDLib++/src/MyLibDD.def") |



Изучить таблицы экспорта обеих библиотек с помощью утилиты **dumpbin** (данная утилита доступна из Developer версии терминала Visual Studio) и предоставить скриншоты с их содержимым! Имя функции преобразованное компилятором C++ сохраните для дальнейших вызовов этой функции!





**Проект Lab-03b-Explicit:**

Разработать клиентское приложение которое сможет **явно** подключить любую из двух динамических библиотек. Приложение должно быть консольным и может быть вызывано в виде:

***Lab-03b-ex <Подключаемая библиотека> <Имя или номер вызываемой функции> [Искомое число]***

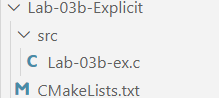
Параметры «Подключаемая библиотека» и «Имя вызываемой функции» должны быть непосредственно значениями передаваемыми в функции **LoadLibrary** и **GetProcAddress**.

Если искомое число не задано, то оно запрашивается для ввода с консоли (для наглядности работы динамической библиотеки поместите этот код непосредственно перед вызовом функции поиска). В результате выполнения этого приложения должен быть выведен один из следующих результатов:

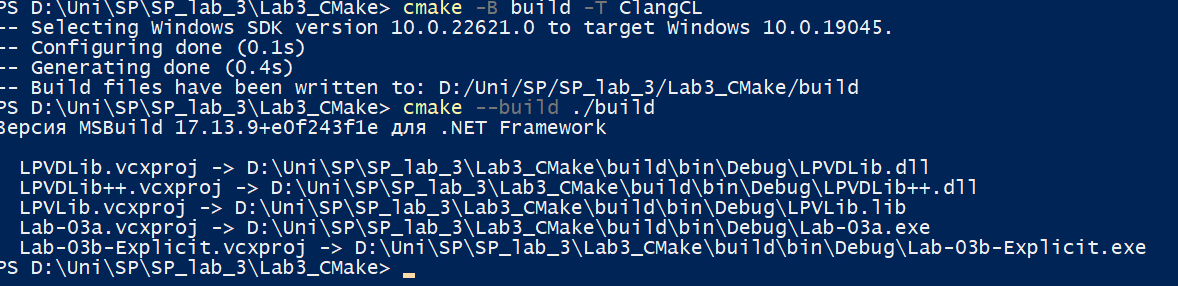
* Не указана загружаемая библиотека!
* Не указана вызываемая функция!
* Загружаемая библиотека не найдена!
* Искомая функция <Имя вызываемой функции> не найдена в библиотеке <Имя загруженной библиотеки>!
* <Имя вызванной функции>: Заданное число не найдено!
* <Имя вызванной функции>: Число <Х> найдено на позиции <Y>!

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <string.h>  #include <windows.h>  typedef int (\*BSearchFunc)(int\*, int, int);  typedef int (\*BSearchRecursiveFunc)(int\*, int, int, int);  typedef const int\* LibArray;  void printUsage() {  printf("Usage: .\\Lab-03b-Explicit.exe <Library> <Function|Number> [Number]\n");  printf("Available libraries: LPVDLib.dll, LPVDLib++.dll\n");  printf("Available functions:\n");  printf(" 1 or iterative\_bsearch - iterative binary search\n");  printf(" 2 or recursive\_bsearch - recursive binary search\n");  }  int main(int argc, char\* argv[]) {  if (argc < 3) {  if (argc < 2) {  printf("The downloadable library is not specified!\n");  }  else {  printf("The function being called is not specified!\n");  }  printUsage();  return 1;  }  char\* libName = argv[1];  char\* funcName = argv[2];  HMODULE hLib = LoadLibrary(libName);  if (!hLib) {  printf("The downloadable library was not found: %s (Error: %d)\n", libName, GetLastError());  return 1;  }  BSearchFunc iterativeSearch = NULL;  BSearchRecursiveFunc recursiveSearch = NULL;  if (strcmp(libName, "LPVDLib.dll") == 0) {  iterativeSearch = (BSearchFunc)GetProcAddress(hLib, "iterative\_bsearch");  recursiveSearch = (BSearchRecursiveFunc)GetProcAddress(hLib, (LPCSTR)2);  }  else if (strcmp(libName, "LPVDLib++.dll") == 0) {  iterativeSearch = (BSearchFunc)GetProcAddress(hLib, "iterative\_bsearch");  recursiveSearch = (BSearchRecursiveFunc)GetProcAddress(hLib, "?lpv\_bsearch\_r@@YAHPEAHHHH@Z");  }  if ((strcmp(funcName, "iterative\_bsearch") == 0 || strcmp(funcName, "1") == 0) && !iterativeSearch) {  printf("The iterative search function was not found in the library!\n");  FreeLibrary(hLib);  return 1;  }  if ((strcmp(funcName, "recursive\_bsearch") == 0 || strcmp(funcName, "2") == 0) && !recursiveSearch) {  printf("The recursive search function was not found in the library!\n");  FreeLibrary(hLib);  return 1;  }  LibArray lib\_array = (LibArray)GetProcAddress(hLib, "lib\_array");  if (!lib\_array) {  printf("Error retrieving the array from the library! (Error: %d)\n", GetLastError());  FreeLibrary(hLib);  return 1;  }  int target;  if (argc < 4) {  printf("Enter a number to search for: ");  if (scanf("%d", &target) != 1) {  printf("Error entering the number!\n");  FreeLibrary(hLib);  return 1;  }  }  else {  target = atoi(argv[3]);  }  int position = -1;  if (strcmp(funcName, "iterative\_bsearch") == 0 || strcmp(funcName, "1") == 0) {  position = iterativeSearch((int\*)lib\_array, 1024, target);  }  else if (strcmp(funcName, "recursive\_bsearch") == 0 || strcmp(funcName, "2") == 0) {  position = recursiveSearch((int\*)lib\_array, target, 0, 1023);  }  else {  printf("The required %s function is not supported!\n", funcName);  FreeLibrary(hLib);  return 1;  }  if (position == -1) {  printf("%s: The specified number was not found!\n", funcName);  }  else {  printf("%s: The number %d was found at position %d!\n", funcName, target, position);  }  FreeLibrary(hLib);  return 0;  } |

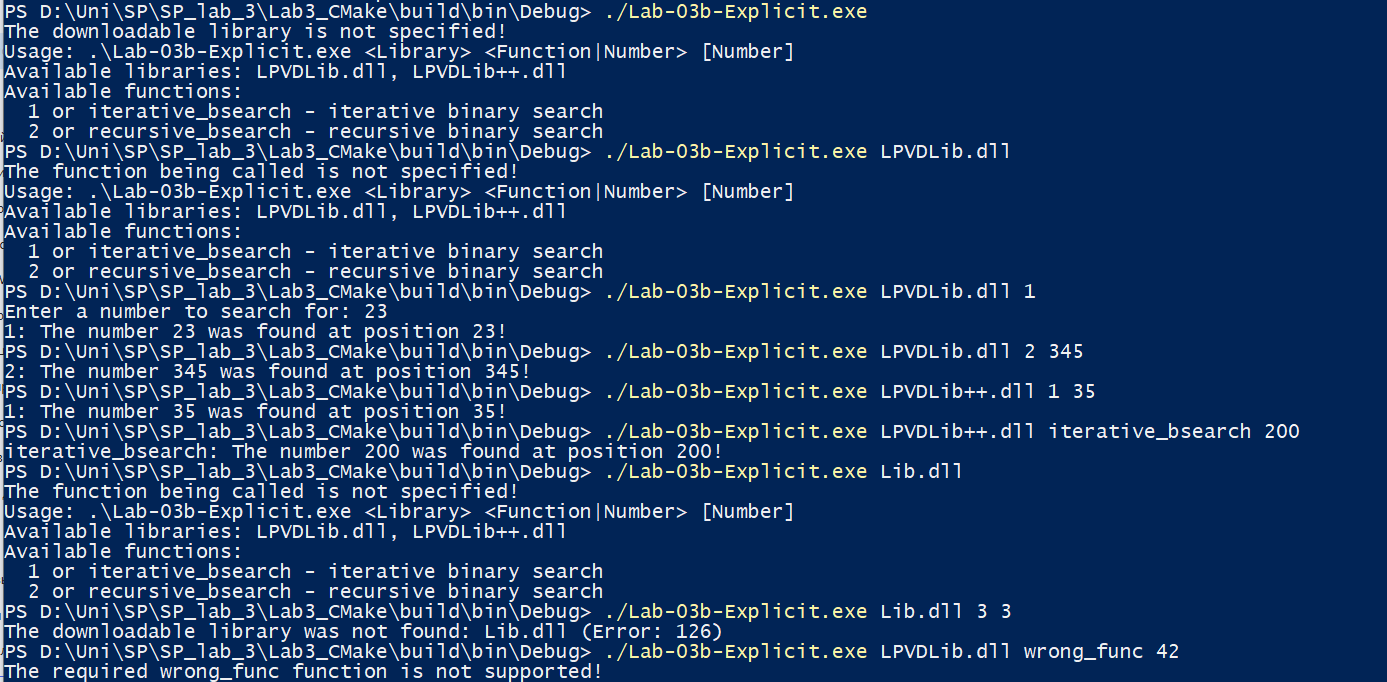
Сборка:

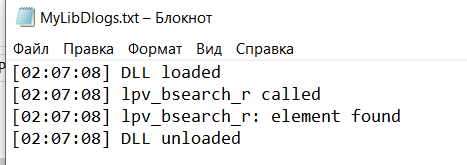


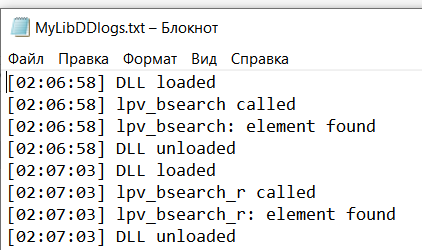
|  |
| --- |
| cmake\_minimum\_required(VERSION 3.28)  add\_executable(Lab-03b-Explicit WIN32 src/Lab-03b-ex.c) |



Результат:

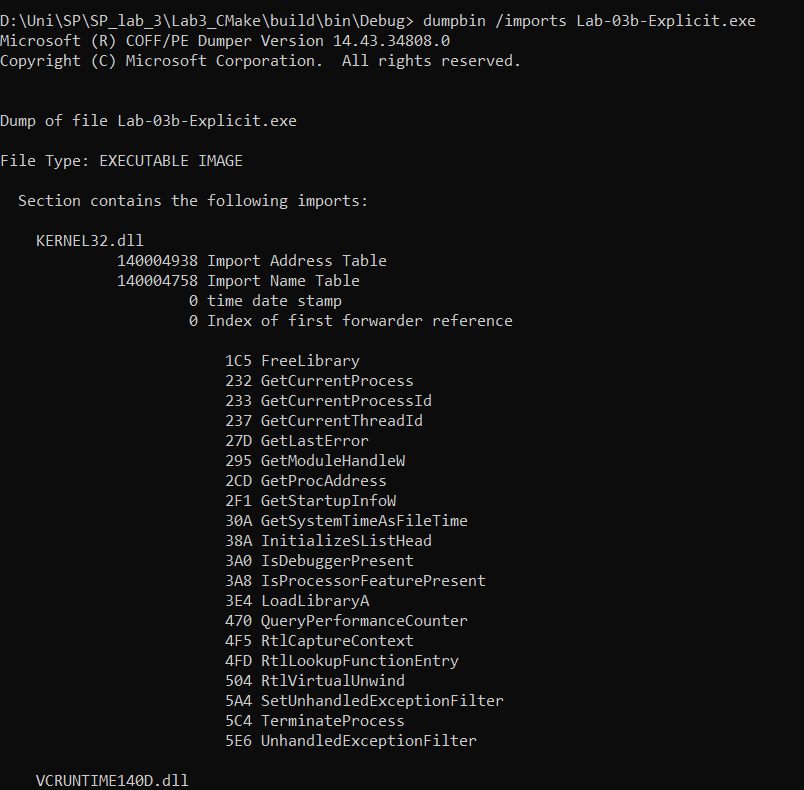




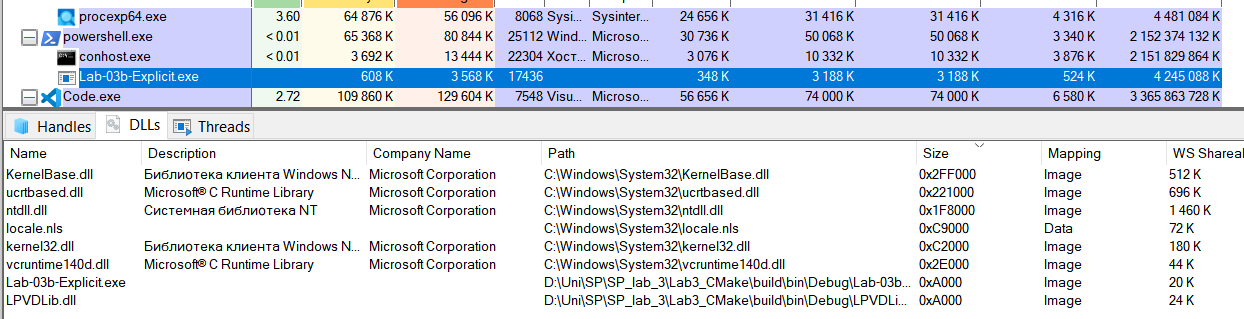


Изучить таблицу импорта приложения с помощью утилиты dumpbin и предоставить скриншот с её содержимым!

**Вопрос**: Что оказалось в таблице импорта? Почему?



Проверить факт подключения библиотеки через Process Explorer. (Для этого не указывайте число в аргументах при запуске приложения и дайте приложению войти в режим ожидания ввода с консоли).



**Проект Lab-03b-Implicit:**

Разработать клиентское приложение которое неявно подключает одну любую из двух динамических библиотек. Приложение должно быть консольным и может быть вызывано в виде:

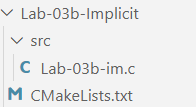
***Lab-03b-im <Имя вызываемой функции> [Искомое число]***

Если искомое число не задано, то оно запрашивается для ввода с консоли. В результате выполнения этого приложения должен быть выведен один из следующих результатов:

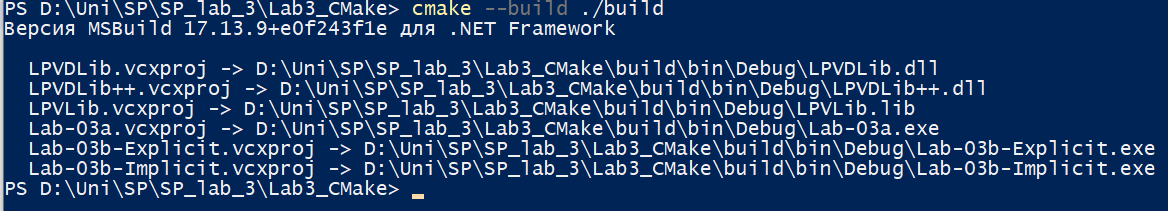
* Не указана вызываемая функция!
* <Имя вызванной функции>: Заданное число не найдено!
* <Имя вызванной функции>: Число <Х> найдено на позиции <Y>!

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <string.h>  #include <windows.h>  \_\_declspec(dllimport) int iterative\_bsearch(int\*, int, int);  \_\_declspec(dllimport) int lpv\_bsearch\_r(int\*, int, int, int);  \_\_declspec(dllimport) const int lib\_array[1024];  void print\_usage() {      printf("Usage: Lab-03b-im <Function Name> [Target Number]\n");  }  int main(int argc, char\* argv[]) {      if (argc < 2) {          printf("Error: No function specified!\n");          print\_usage();          return 1;      }      char\* function\_name = argv[1];      int target = 0;      if (argc < 3) {          printf("Enter the number to search for: ");          if (scanf\_s("%d", &target) != 1) {              printf("Error: Invalid input for number! Please enter a valid integer.\n");              return 1;          }      }      else {          char\* endptr;          target = strtol(argv[2], &endptr, 10);          if (\*endptr != '\0') {              printf("Error: Invalid target number format! Please provide a valid integer.\n");              return 1;          }      }      int position = -1;      if (strcmp(function\_name, "iterative\_bsearch") == 0) {          position = iterative\_bsearch((int\*)lib\_array, 1024, target);      }      else if (strcmp(function\_name, "bsearch\_r") == 0) {          position = lpv\_bsearch\_r((int\*)lib\_array, target, 0, 1023);      }      else {          printf("Error: Function '%s' not found in the library!\n", function\_name);          print\_usage();          return 1;      }      if (position == -1) {          printf("%s: The number %d was not found in the array!\n", function\_name, target);      }      else {          printf("%s: The number %d was found at position %d!\n", function\_name, target, position);      }      return 0;  } |

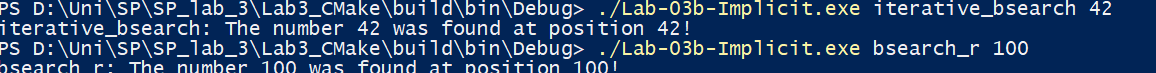
Сборка:



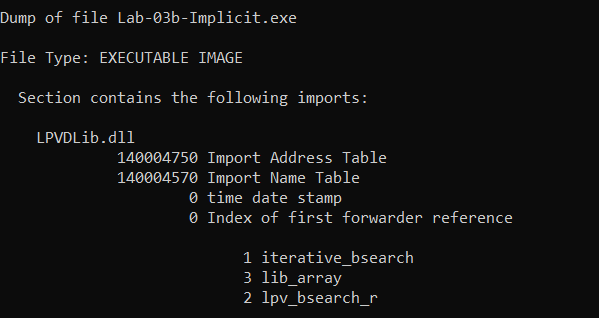
|  |
| --- |
| cmake\_minimum\_required(VERSION 3.28)  include\_directories(${CMAKE\_SOURCE\_DIR}/LPVDLib/include)  add\_executable(Lab-03b-Implicit WIN32 src/Lab-03b-im.c)  target\_link\_libraries(Lab-03b-Implicit LPVDLib) |



Результат:

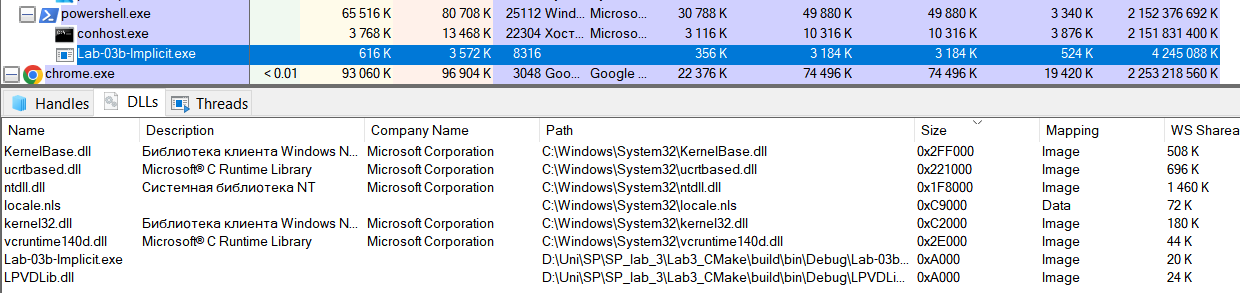


Изучить таблицу импорта приложения с помощью утилиты dumpbin и предоставить скриншот с её содержимым!



Вопрос: Что оказалось в таблице импорта? Почему?

Проверить факт подключения библиотеки через Process Explorer. (Для этого не указывайте число в аргументах при запуске приложения и дайте приложению войти в режим ожидания ввода с консоли).



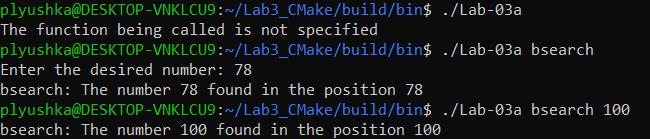
**Постановка задачи для Linux:**

**Проект <Ваши Инициалы>Lib:**

Собрать статическую библиотеку из проекта с аналогичным названием на Windows.

**Проект Lab-03a:**

Собрать приложение из проекта с аналогичным названием на Windows. Проверить работоспособность приложения.



**Проект Lib<Ваши Инициалы>d:**

Разработать разделяемую библиотеку на языке C с заданным функционалом (см. выше). Также должна быть реализованы функции constructor и destructor с логированием событий в некоторый файл (путь к файлу должен быть абсолютным).

LibLPVd.h

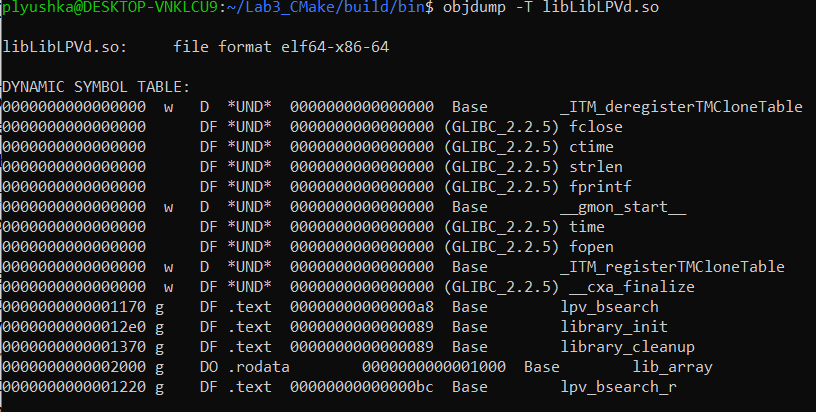
|  |
| --- |
| #ifndef LIB\_LPV\_D\_H  #define LIB\_LPV\_D\_H  #define LIB\_ARRAY\_SIZE 1024  extern const int lib\_array[LIB\_ARRAY\_SIZE];  int lpv\_bsearch(int\* a, int n, int x);  int lpv\_bsearch\_r(int\* a, int x, int i, int j);  void \_\_attribute\_\_((constructor)) library\_init(void);  void \_\_attribute\_\_((destructor)) library\_cleanup(void);  #endif |

LibLPVd.c

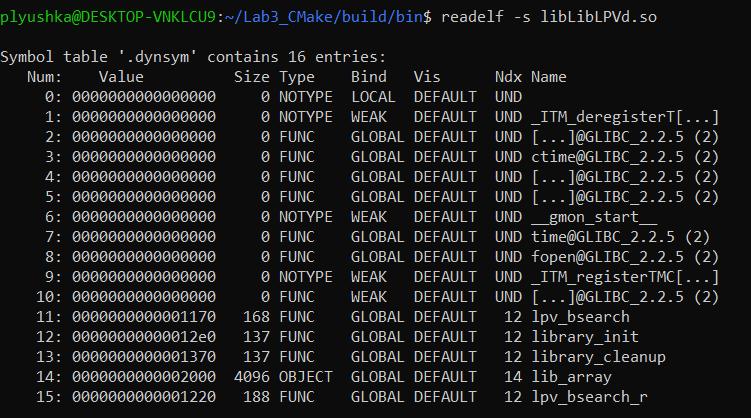
|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <string.h>  #include <time.h>  #include <unistd.h>  #include "LibLPVd.h"  #define LOG\_FILE "/tmp/log.txt"  const int lib\_array[LIB\_ARRAY\_SIZE] = {      0,   1,   2,  … , 1022, 1023  };  int lpv\_bsearch(int\* a, int n, int x) {      int i = 0, j = n - 1;      while (i <= j) {          int k = i + ((j - i) / 2);          if (a[k] == x) {              return k;          }          else if (a[k] < x) {              i = k + 1;          }          else {              j = k - 1;          }      }      return -1;  }  int lpv\_bsearch\_r(int\* a, int x, int i, int j) {      if (j < i) {          return -1;      }      int k = i + ((j - i) / 2);      if (a[k] == x) {          return k;      }      else if (a[k] < x) {          return lpv\_bsearch\_r(a, x, k + 1, j);      }      else {          return lpv\_bsearch\_r(a, x, i, k - 1);      }  }  void \_\_attribute\_\_((constructor)) library\_init(void) {      FILE\* log = fopen(LOG\_FILE, "a");      if (log) {          time\_t now = time(NULL);          char\* time\_str = ctime(&now);          time\_str[strlen(time\_str) - 1] = '\0';          fprintf(log, "[LIB INIT] %s\n", time\_str);          fclose(log);      }  }  void \_\_attribute\_\_((destructor)) library\_cleanup(void) {      FILE\* log = fopen(LOG\_FILE, "a");      if (log) {          time\_t now = time(NULL);          char\* time\_str = ctime(&now);          time\_str[strlen(time\_str) - 1] = '\0';          fprintf(log, "[LIB EXIT] %s\n", time\_str);          fclose(log);      }  } |

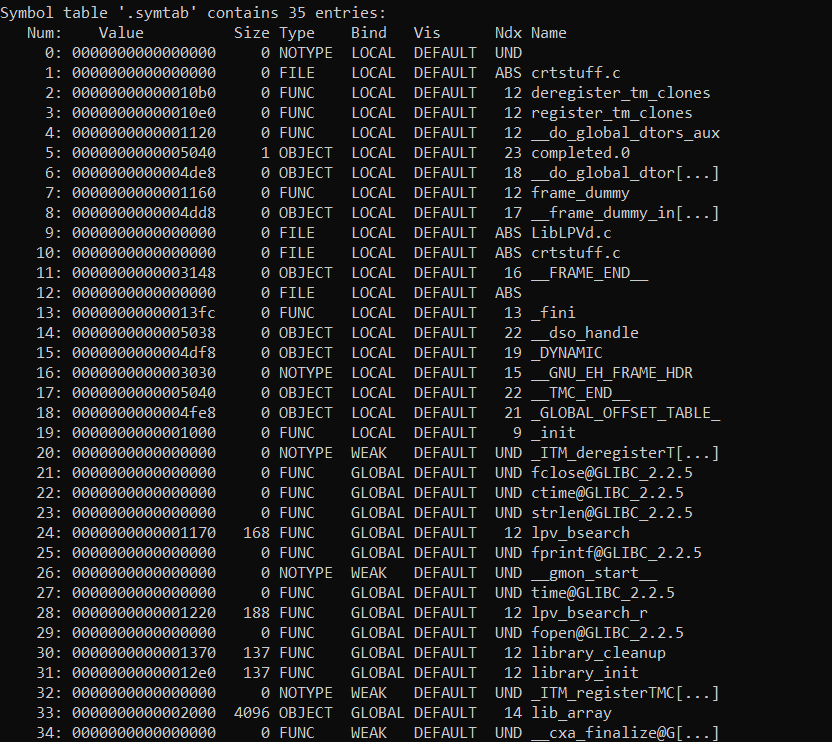
Просмотреть список экспортируемых символов через утилиту objdump/readelf и предоставить скриншот с его содержимым!

objdump -T libLibLPVd.so



readelf -s libLibLPVd.so





**Проект Lab-03c:**

Разработать клиентское приложение которое сможет явно подключать разделяемую библиотеку. Приложение должно быть консольным и может быть вызывано в виде:

***Lab-03с <Подключаемая библиотека> <Имя или номер вызываемой функции> [Искомое число]***

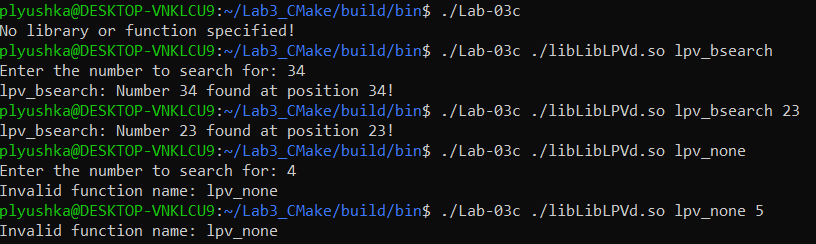
Параметры «Подключаемая библиотека» и «Имя вызываемой функции» должны быть непосредственно значениями передаваемыми в функции **dlopen** и **dlsym**.

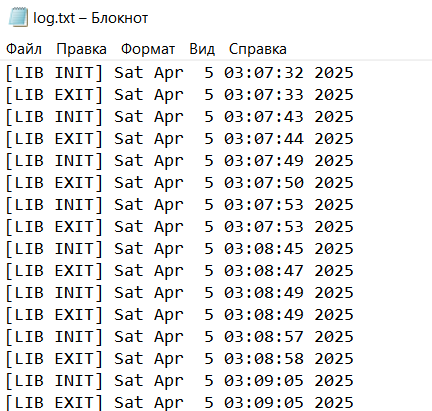
Если искомое число не задано, то оно запрашивается для ввода с консоли (для наглядности работы динамической библиотеки поместите этот код непосредственно перед вызовом функции поиска). В результате выполнения этого приложения должен быть выведен один из следующих результатов:

* Не указана загружаемая библиотека!
* Не указана вызываемая функция!
* Загружаемая библиотека не найдена!
* Искомая функция <Имя вызываемой функции> не найдена в библиотеке <Имя загруженной библиотеки>!
* <Имя вызванной функции>: Заданное число не найдено!
* <Имя вызванной функции>: Число <Х> найдено на позиции <Y>!

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <dlfcn.h>  #include <string.h>  int my\_bsearch(int \*a, int n, int x);  int my\_bsearch\_r(int \*a, int x, int i, int j);  int main(int argc, char \*argv[]) {      if (argc < 3) {          fprintf(stderr, "No library or function specified!\n");          return 1;      }      const char \*lib\_name = argv[1];      const char \*func\_name = argv[2];      void \*handle = dlopen(lib\_name, RTLD\_LAZY);      if (!handle) {          fprintf(stderr, "Failed to load library: %s\n", dlerror());          return 1;      }      int x;      if (argc == 3) {          printf("Enter the number to search for: ");          if (scanf("%d", &x) != 1) {              fprintf(stderr, "Error reading number!\n");              dlclose(handle);              return 1;          }      } else {          x = atoi(argv[3]);      }      const int \*lib\_array = dlsym(handle, "lib\_array");      if (!lib\_array) {          fprintf(stderr, "Failed to find array in the library %s: %s\n", lib\_name, dlerror());          dlclose(handle);          return 1;      }      int array\_size = 1024;      int position = -1;      if (strcmp(func\_name, "lpv\_bsearch") == 0) {          int (\*search\_function\_i)(int \*, int, int) = (int (\*)(int \*, int, int))dlsym(handle, func\_name);          if (!search\_function\_i) {              fprintf(stderr, "Function %s not found in the library %s: %s\n", func\_name, lib\_name, dlerror());              dlclose(handle);              return 1;          }          position = search\_function\_i(lib\_array, array\_size, x);      }      else if (strcmp(func\_name, "lpv\_bsearch\_r") == 0) {          int (\*search\_function\_r)(int \*, int, int, int) = (int (\*)(int \*, int, int, int))dlsym(handle, func\_name);          if (!search\_function\_r) {              fprintf(stderr, "Function %s not found in the library %s: %s\n", func\_name, lib\_name, dlerror());              dlclose(handle);              return 1;          }          position = search\_function\_r(lib\_array, x, 0, array\_size - 1);      }      else {          fprintf(stderr, "Invalid function name: %s\n", func\_name);          dlclose(handle);          return 1;      }      if (position == -1) {          printf("%s: The number was not found!\n", func\_name);      } else {          printf("%s: Number %d found at position %d!\n", func\_name, x, position);      }      dlclose(handle);      return 0;  } |

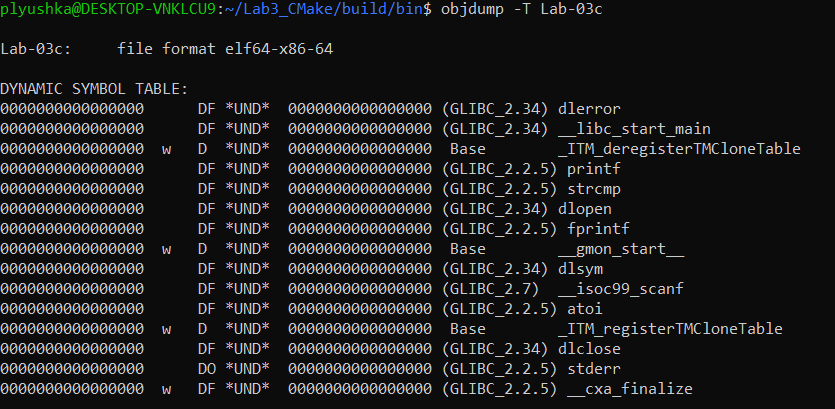
Результат:



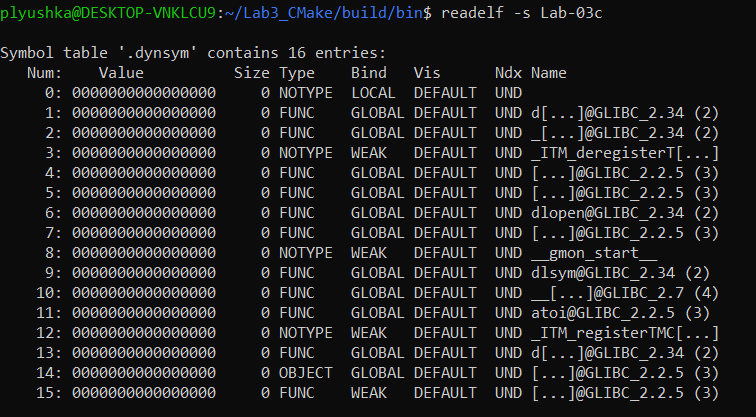


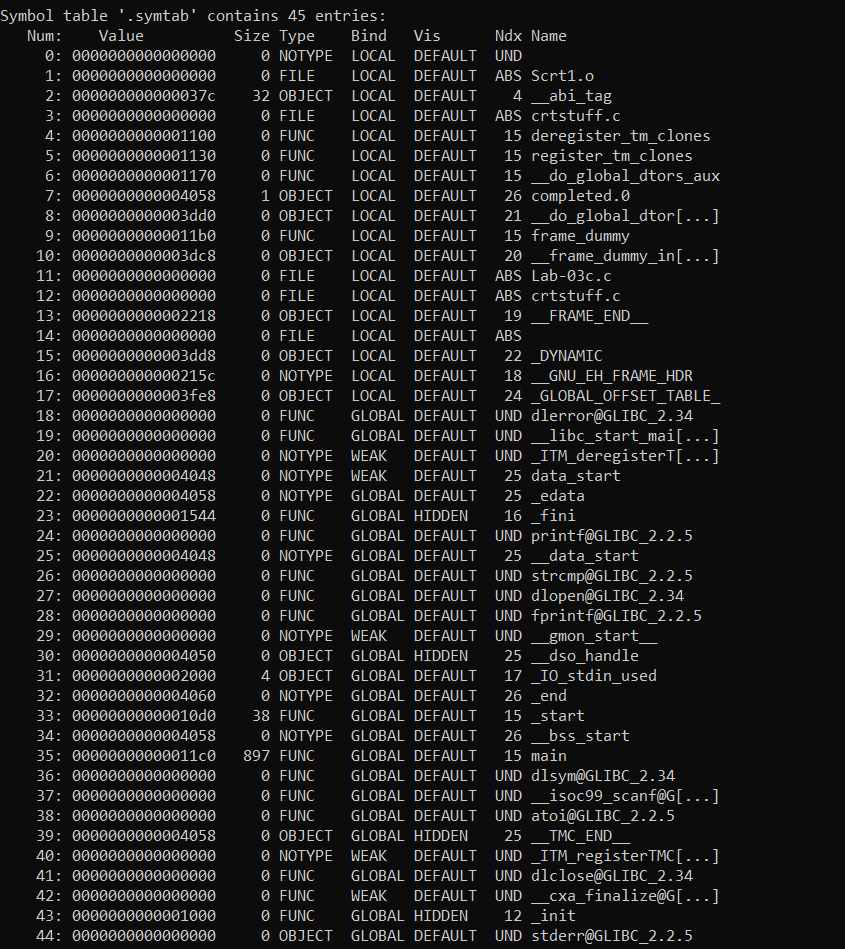
Просмотреть список импортируемых символов через утилиту **objdump/readelf** и предоставить скриншот с его содержимым!

objdump -T Lab-03c

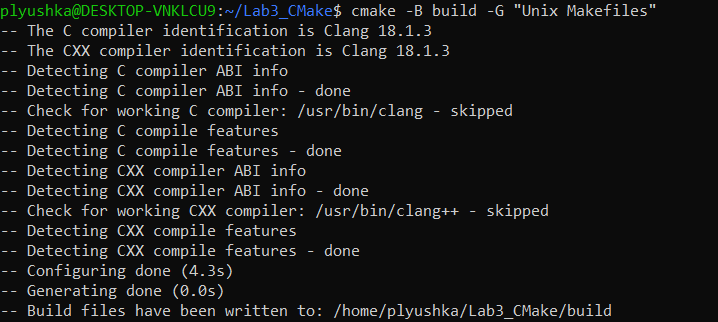


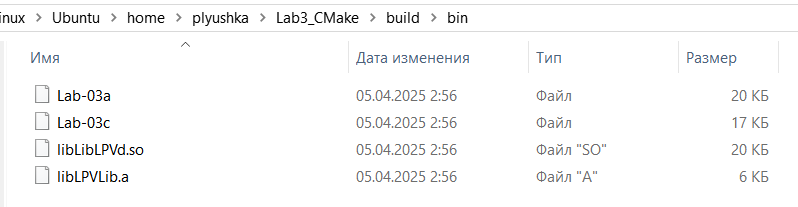
readelf -s Lab-03c





Сборка:





**Сборка Clang**

**Windows**

1. **Статические**: LPVLib, Lab-03a

clang -target x86\_64-pc-win32 -I.\include\ -c -o .\bin\LPVLib.o .\src\MyLib.c



llvm-ar cr .\bin\LPVLib.lib .\bin\LPVLib.o



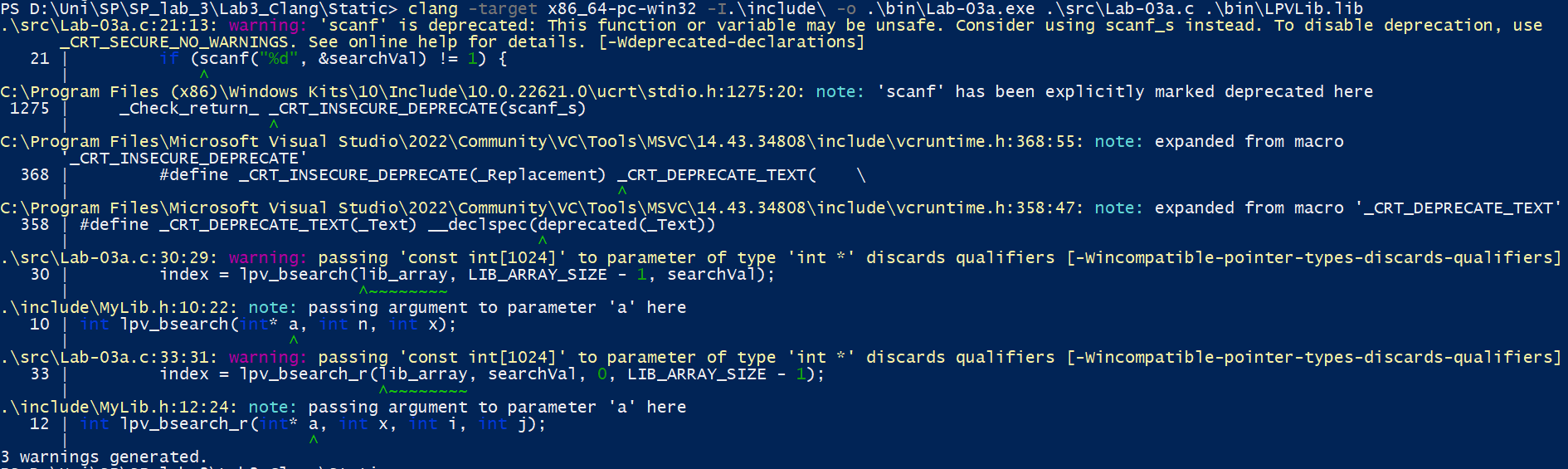
Индексирование

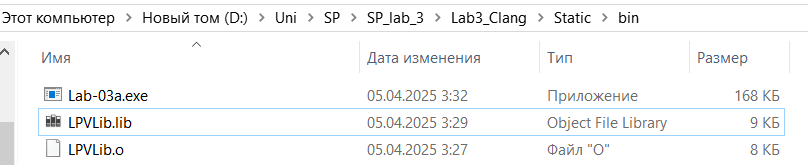
llvm-ranlib .\bin\LPVLib.lib

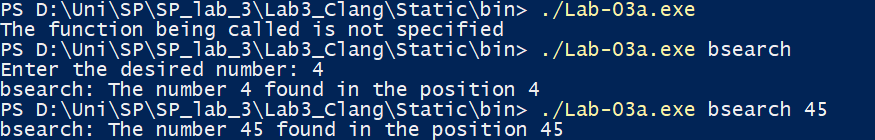


Приложение

clang -target x86\_64-pc-win32 -I.\include\ -o .\bin\Lab-03a.exe .\src\Lab-03a.c .\bin\LPVLib.lib

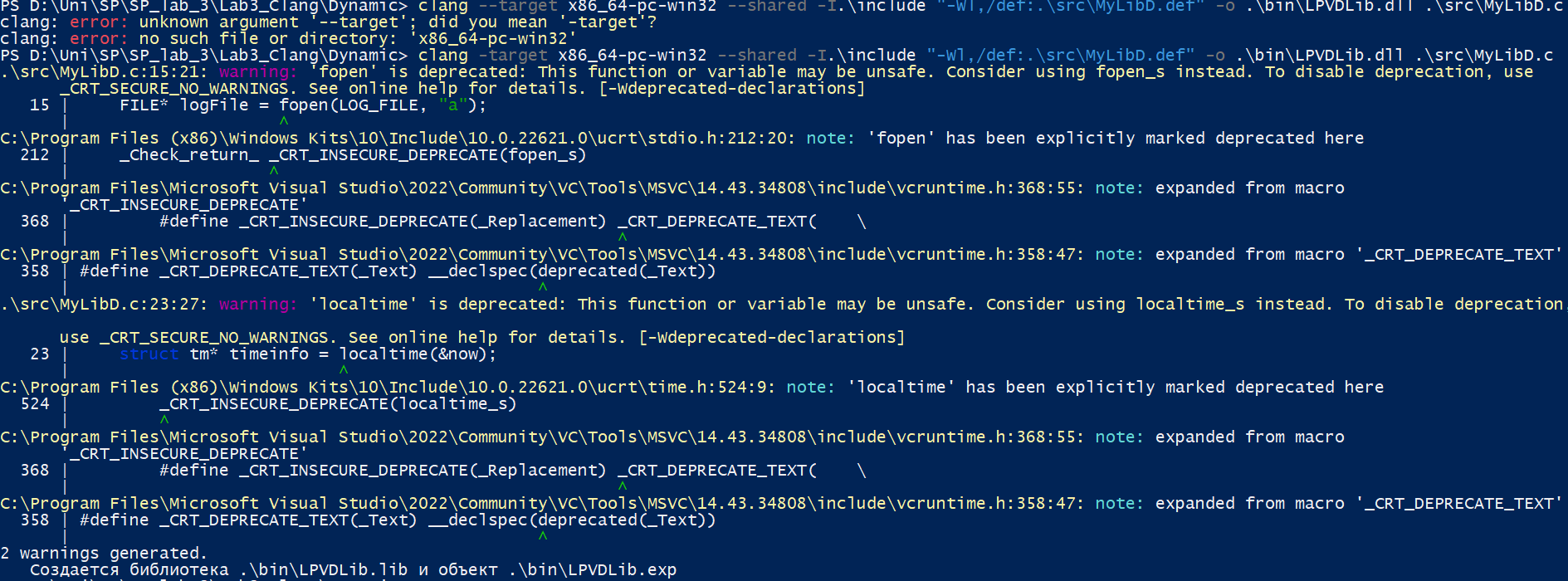




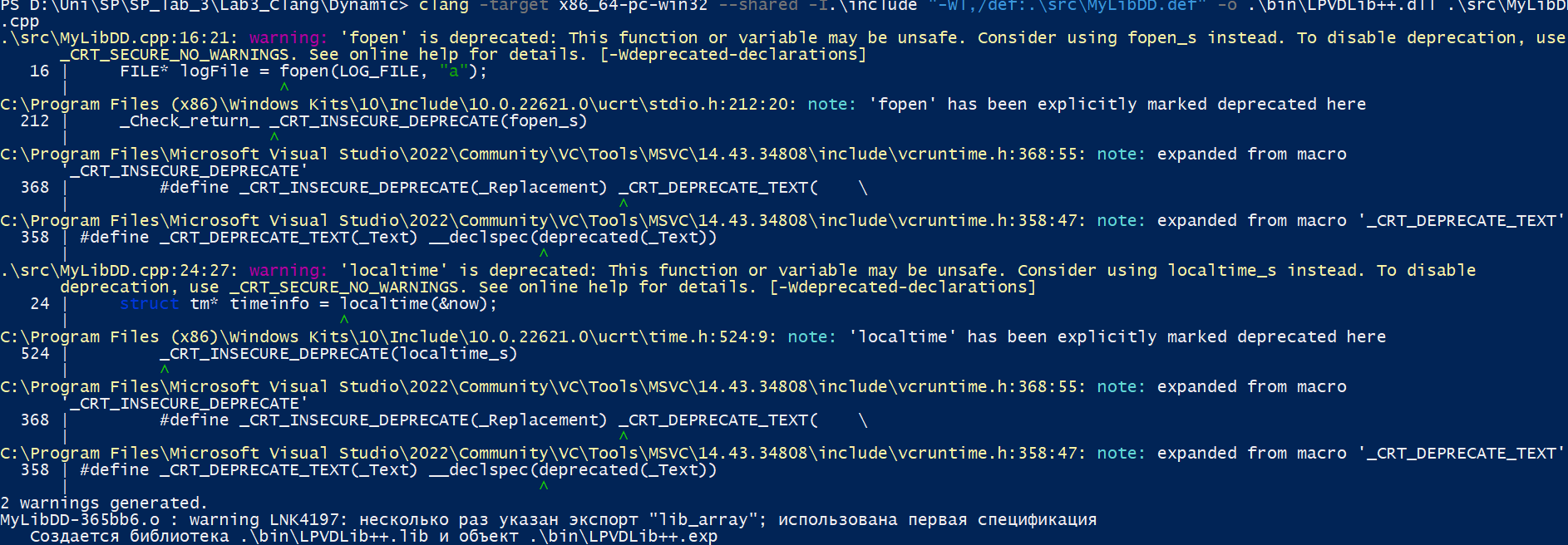


1. **Динамические**: LPVDLib, LPVDLib++, Lab-03b-Explicit, Lab-03b-Implicit

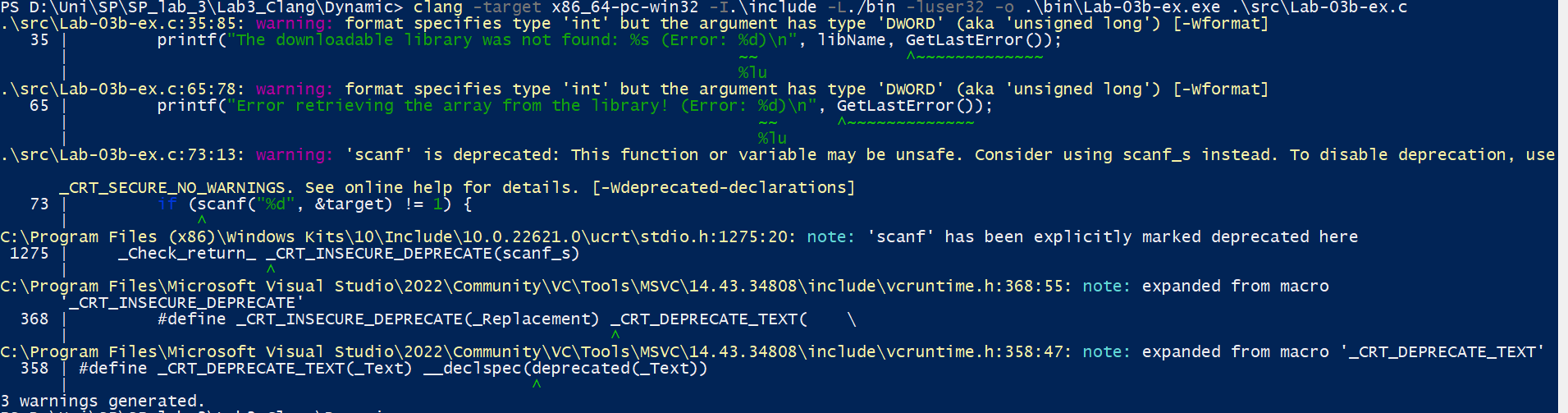
clang -target x86\_64-pc-win32 --shared -I.\include “-Wl,/def:.\src\MyLibD.def” -o .\bin\LPVDLib.dll .\src\MyLibD.c



clang -target x86\_64-pc-win32 --shared -I.\include “-Wl,/def:.\src\MyLibDD.def” -o .\bin\LPVDLib++.dll .\src\MyLibDD.c

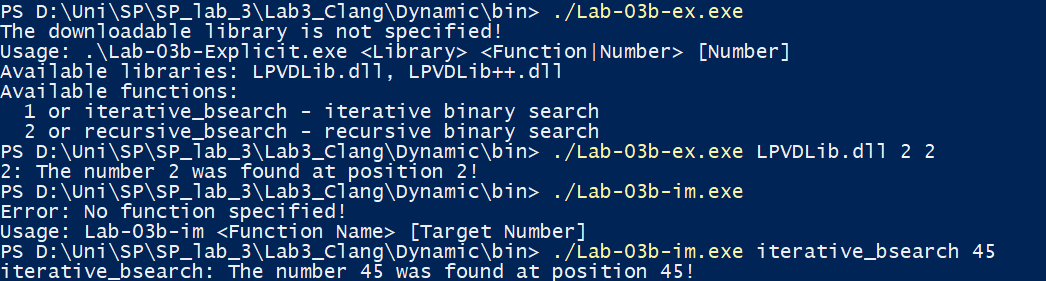


clang -target x86\_64-pc-win32 -I.\include -L./bin -luser32 -o .\bin\Lab-03b-ex.exe .\src\Lab-03b-ex.c



clang -target x86\_64-pc-win32 -I./include -L./bin -lLPVDlib -o .\bin\Lab-03b-im.exe .\src\Lab-03b-im.c





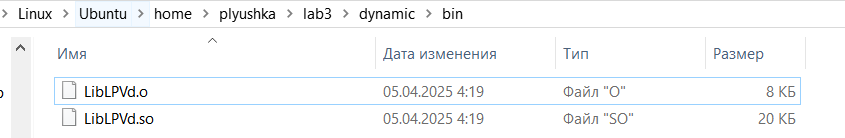
**Linux**

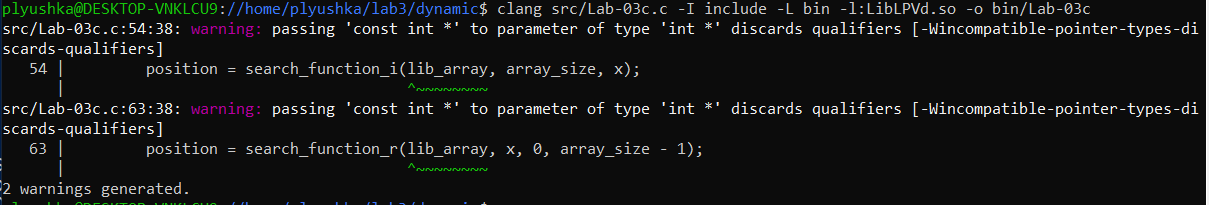
1. **Динамическая**: LibLPVd, Lab-03c

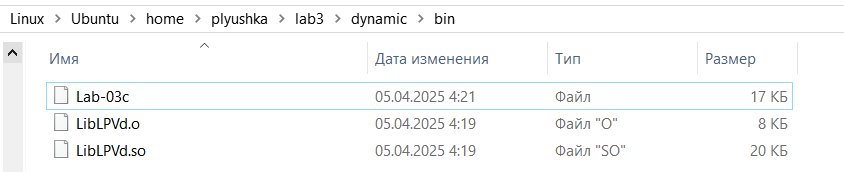
clang -fPIC -I include -c src/LibLPVd.c -o bin/LibLPVd.o

clang -shared -o bin/LibLPVd.so bin/LibLPVd.o





clang src/Lab-03c.c -I include -L bin -l:LibLPVd.so -o bin/Lab-03c



LD\_LIBRARY\_PATH=. ./Lab-03c LibLPVd.so lpv\_bsearch 2

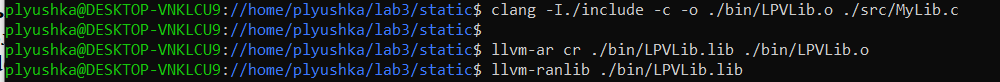


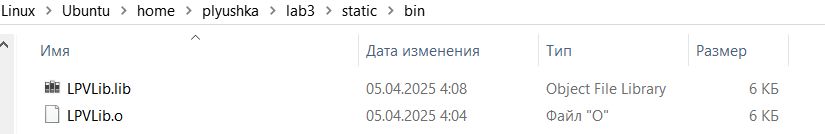
1. **Статическая**: LPVLib, Lab-03a

clang -I./include -c -o ./bin/LPVLib.o ./src/MyLib.c

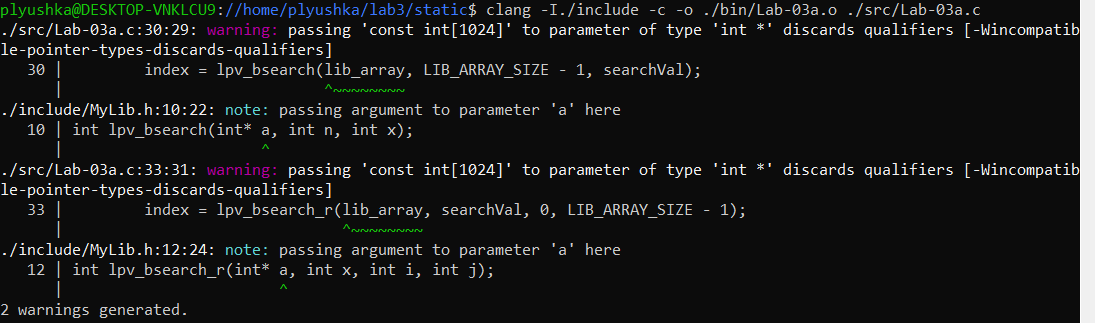
llvm-ar cr ./bin/LPVLib.lib ./bin/LPVLib.o

llvm-ranlib ./bin/LPVLib.lib



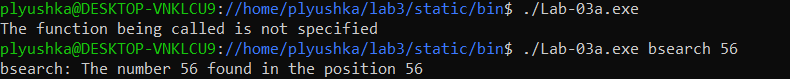


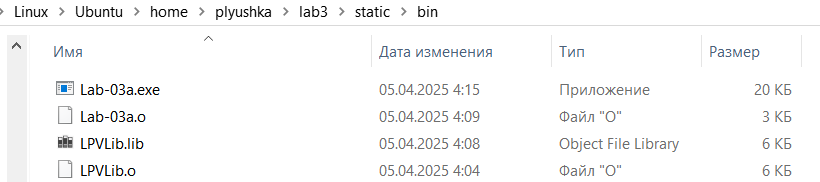
clang -I./include -c -o ./bin/Lab-03a.o ./src/Lab-03a.c



Clang -o ./bin/Lab03a.exe ./bin/Lab03a.o ./bin/LPVLib.lib







**Требования к выполненной работе:**

* Для генерации и сборки проекта использовать как средство CMake, так и clang/clang++ напрямую;
* Для обоих случаев выучить команды (консольные или в CMakeLists.txt) которые используются для сборки библиотек или приложений использующих библиотеки!
* Для обоих случаев подготовить серию скриншотов со сборкой библиотек и приложений!

**При использовании CMake должно быть выполнено следующее:**

* Сборка должна быть «out-of-source»;
* Для Windows использовать генератор «Visual Studio»/«Ninja» для Linux – «Unix Makefiles», для macOS – «XCode»;
* При сборке проектов всегда должен использоваться clang;
* Все проекты для обеих ОС являются частью одного большого проекта. Добавить в конфигурационные файлы условия сборки отдельных проектов для разных ОС. Конечные файлы генерируются в общий каталог!

Вопросы для контроля:

1. **Что такое библиотека?**

**Библиотека** – файл, содержащий несколько объектных файлов, которые будут использоваться вместе на стадии сборки (линковки) программы.

1. **Что такое статическая библиотека?**

**Статическая библиотека** – файл, содержащий копии всех помещенных в него объектных файлов. В архиве хранятся атрибуты, права доступа и тд.

1. **Как подключить статическую библиотеку?**

**Cmake**

add\_library(lib\_name STATIC lib\_name.c )

target\_link\_libraries(program\_name lib\_name)

**Clang**

clang -target x86\_64-pc-win32 -I.\include -o .\bin\ program\_name.exe .\src\ lib\_name.c .\bin\ lib\_name.lib

clang -o ./bin/ program\_name ./bin/ program\_name.o -L./bin –lnan (lib\_name)

1. **На каком этапе сборки происходит непосредственно загрузка кода?**

Для статичесих библиотек на этапе комппоновки.

Для динамических при выполнении программы.

1. **Что такое динамическая библиотека?**

**Динамическая библиотека** – копия объектного модуля разделенная между всеми программами, которые его используют. Они не компилируются в исходный файл, копия библиотеки загружается в память при запуске.

1. **Какой механизм лежит в основе работы динамических библиотек?**

**Отложенная загрузка (Lazy Loading**). Код библиотеки не включается в исполняемый файл на этапе компиляции. Вместо этого программа загружает библиотеку во время выполнения при первом вызове её функций.

**Разделение кода между процессами.** Динамическая библиотека загружается в память один раз, и несколько программ могут использовать её одновременно. Это экономит оперативную память и упрощает обновления (достаточно заменить одну DLL/SO).

**Динамическое связывание (Dynamic Linking).** В отличие от статических библиотек (\*.a, \*.lib), где код встраивается в исполняемый файл, динамические библиотеки подключаются через:

* Таблицу импорта (указатели на функции из DLL).
* Динамический загрузчик ОС (например, ld.so в Linux, LoadLibrary() в Windows).

**Разрешение символов (Symbol Resolution).** При загрузке библиотеки ОС ищет требуемые функции (символы) и связывает их с вызовами в программе.

1. **Назовите два способа подключения динамической библиотеки? Кратко поясните порядок подключений.**

**Неявное**

* Программа компилируется с заголовочным файлом (.h) библиотеки.
* Компоновщик (линкер) использует файл импорта (\*.lib в Windows, \*.so в Linux), чтобы разрешить символы.
* При запуске программы ОС автоматически загружает библиотеку (DLL/SO).
* Если библиотека не найдена — программа не запустится.
* Функции вызываются напрямую, как если бы они были частью программы.

**Явное**

* Программа вручную загружает библиотеку через API ОС:
* Windows: LoadLibrary() → GetProcAddress()
* Linux: dlopen() → dlsym()
* Функции вызываются через указатели, а не напрямую.
* После использования библиотека выгружается (FreeLibrary() / dlclose()).

1. **Что такое библиотека импорта?**

Как можно заметить в коде и заголовочном файле библиотеки применяются модификаторы:

**[\_\_declspec(dllimport)](https://learn.microsoft.com/en-us/cpp/cpp/dllexport-dllimport?view=msvc-170)** – такой модификатор означает, что данная переменная или функция импортируются из DLL

**[\_\_declspec(dllexport)](https://learn.microsoft.com/en-us/cpp/cpp/dllexport-dllimport?view=msvc-170)** – такой модификатор означает, что данная переменная или функция экспортируется из DLL

Если он указан перед переменной, прототипом функции или C++-классом, компилятор С/С++ встраивает в конечный OBJ-файл дополнительную информацию. Она понадобится компоновщику при сборке DLL из OBJ-файлов.

Обнаружив такую информацию, компоновщик создает LIB-файл со списком идентификаторов, экспортируемых из DLL. Этот LIB-файл нужен при сборке любого EXE-модуля, ссылающегося на такие идентификаторы и называется библиотекой импорта.

В библиотеке импорта содержатся ссылки на все экспортируемые из динамической библиотеки имена

1. **Для чего нужен extern “C”?**

extern "C" — это ключевое слово в C++, которое указывает компилятору использовать C-стиль именования и линковки для функций и переменных. Влияет только на линковку, но не на синтаксис (внутри extern "C" можно использовать C++-код).

Name Mangling (декорирование имён) в C++. В C++ компилятор изменяет имена функций (name mangling), чтобы поддерживать:

Перегрузку функций (функции с одним именем, но разными параметрами).

Пространства имён (namespace).

Методы классов.

Проблема: Если C++-функция экспортируется из DLL без extern "C", её имя в таблице экспорта будет искажено, и другие программы (особенно на C) не смогут её найти.

1. **Функции жизненного цикла динамических библиотек в Windows и Linux?**

**Windows**

**Явное**

LoadLibrary() / LoadLibraryEx() - Загружает DLL в память процесса.

GetProcAddress() - Возвращает указатель на функцию из DLL

FreeLibrary() - Освобождает DLL из памяти.

**Неявное**

DLL\_PROCESS\_ATTACH: break;

DLL\_PROCESS\_DETACH: break;

**Linux**

**Явное**

dlopen() - Загружает .so в память. ()RTLD\_LAZY — разрешает символы по мере использования. RTLD\_NOW — проверяет все символы сразу.

dlsym() - Возвращает указатель на функцию.

dlclose() - Уменьшает счётчик ссылок и выгружает .so, если он больше не нужен.

**Неявное**

\_\_attribute\_\_((constructor))

\_\_attribute\_\_((destructor)).