ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ
Государственное образовательное учреждение
высшего профессионального образования
«ТОМСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

В.Г.Гальченко

ЛАБОРАТОРНЫЙ ПРАКТИКУМ
СИСТЕМНОЕ ПРОГРАММИРОВАНИЕ В СРЕДЕ WIN32
СОЗДАНИЕ WINDOWS ПРИЛОЖЕНИЙ

Учебное пособие

Издательство
Томского политехнического университета
Томск 2009

УДК 681.3.07

Гальченко В.Г. Лабораторный практикум. Системное программирование в среде WIN32. Создание Windows приложений. Учебное пособие / В.Г.Гальченко. – Томск: Изд. ТПУ, 2009.– 68 с.

В пособии рассмотрены вопросы использования интерфейса программирования приложений (API) Win32 для разработки системных программ управления файловой системой, управления процессами и потоками, асинхронным вводом-выводом. В пособии приводятся примеры программирования процесса включения функции в динамическую библиотеку и вызова этой функции из динамической библиотеки. Кроме того, представлены примеры программирования пользовательского интерфейса с использованием библиотеки классов Microsoft Foundation Classes (MFC).

В качестве среды программирования используется Visual Studio C++.NET 2003. Все программы, которые представлены в лабораторных работах, отлажены под управление ОС Windows XP.

Пособие рассчитано на студентов старших курсов специальности «Прикладная математика и информатика», которые хотят освоить тонкости программирования приложений на языке Visual Studio C++.NET 2003.

Печатается по постановлению
Редакционно - издательского Совета
Томского политехнического университет

Рецензенты:
1.А.А.Шелестов – доцент кафедры АСУ, Томского университета систем управления и радиоэлектроники, кандидат технических наук.

2.Б.М.Шумилов – профессор кафедры прикладной математики Томского государственного архитектурного университета, доктор физико-математических наук.

Темплан 2009

П р е д и с л о в и е

Учебное пособие представляет собой лабораторный практикум программирования в среде Win32. В нем представлены примеры системного программирования приложений управления файловой системой, процессами, асинхронным вводом-выводом и т.д.

Лабораторный практикум состоит из 9 лабораторных работ, охватывающих основные разделы программирования в среде Win32 и программирования пользовательского интерфейса. В каждой лабораторной работе приводится теоретический раздел, в котором дается подробное описание используемых функций Win32, параметров функций и их возможные значения, результат выполнения данной функции и правила ее использования в программе. Программной средой выполнения лабораторной работы является Visual Studio C++.NET 2003. В связи с этим, в каждой лабораторной работе описывается процесс создания проекта, основные файлы проекта и технология компиляции файла программы. Использование функций Win32 для выполнения лабораторной работы приводится в программе, написанной на Visual Studio C++.NET 2003.

Выполнение лабораторной работы сводится к созданию исполняемого модуля и запуску программы из командной строки с соответствующими исходными данными.

После выполнения лабораторной работы производится защита, на которой студенты должны пояснить технологию создания проекта задания, выбор основных параметров проекта, использование функций Win32, процесс создания Windows приложения в каждой работе и ответить на вопросы, которые приводятся в задании.

Использование данного пособия поможет студентам освоить процесс использования функций Win32, программирование интерфейса и Windows приложений под управлением ОС Windows XP.

Лабораторная работа №1

Копирование файлов с использованием Win32

Последовательная обработка файлов – самая простая, наиболее обычная и наиболее необходимая функция любой файловой системы. Копирование файлов, часто с обновлением, и объединением отсортированных файлов – распространенные формы последовательной обработки. Простое копирование позволяет определить сильные и слабые стороны разных систем и перейти к Win32.

В данной работе изучается копирование файлов с использованием стандартной библиотеки на языке С, копирование файлов с использованием интерфейса программирования приложений (API) Win32, а также с использованием функции-полуфабриката Win32.

1. Копирование файлов с использованием библиотеки С

Программа Cpc

Обращение к программе производится из командной строки: Cpc файл 1, файл2 - копирование файлов с использованием библиотеки С, копирует файл1 в файл2 .

```
#include <stdio.h>
#include <errno.h>
#define BUF_SIZE 256
int main (int argc, char *argv [])
{
   FILE *in_file, *out_file; char rec
   [BUF_SIZE]; size_t bytes_in,
   bytes_out;
      if (argc != 3)
      {
      printf (“Использование: Cpc file1 File2\n”);
      return 1;
      }
   in_file = fopen (argv [1], “rb”);
   if (in_file == NULL)
   {
      perror (argv [1] ) ;
      return 2;
   }
```

```
    out_file = fopen (argv [2], “wb”);
    if (out_file == NULL)
        {
        perror (argv [2] ) ;
        return 3;
        }
    /* Обрабатываем входной файл по одной записи. */
     while ( (bytes_in = fread (rec, 1, BUF_SIZE, in_file) ) > 0)
{
        bytes_out = fwrite (rec, 1, bytes__in, out_file) ;
        if (bytes_out != bytes_in)
        {
           perror («Неисправимая ошибка записи.»);
           return 4;
        }
}
        fclose (in_file);
        fclose (out_file);
}
```

Программа запускается из командной строки.

2. Копирование файлов с использованием API Win32

Программа Cpw

Обращение к программе производится из командной строки: Cpw файл 1, файл 2 – копирование файлов с использованием API Win32 , копирует файл1 в файл2.

```
#include <windows.h>
#include <stdio.h>
#define  BUF_SIZE   256
int main   (int  argc,   LPTSTR argv   [])
{
   HANDLE  hIn,   hOut;
   DWORD  nIn,   nOut;
   CHAR Buffer   [BUF_ SIZE];
   if (argc != 3)
       {
       printf (“Использование: CpW file1 File2\n”);
       return 1;
       }
       hIn = CreateFile (argv [1], GENERIC_READ, 0, NULL,
   OPEN_EXISTING, 0, NULL);
       if (hIn = = INVALID_HANDLE_VALUE)
```

```
    {
    printf («Нельзя открыть входной файл. Ошибка: %x\n”,
       GetLastError ());
    return 2;
    }
    hOut = CreateFile (argv [2], GENERIC_WRITE, 0, NULL,
    CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL,
  NULL);
    if (hOut = =INVALID_HANDLE_VALUE)
    {
    printf («Нельзя открыть выходной файл. Ошибка: %x\n”,
       GetLastError () ) ;
    return 3;
    }
    while (ReadFile (hIn, Buffer, BUF_SIZE, &nIn, NULL) && nIn > 0)
    {
    WriteFile (hOut, Buffer, nIn, &nOut, NULL);
    if (nIn != nOut)
          {
       printf (“Неисправимая ошибка записи: %x\n”, GetLastError
       ( ) ) ;
       return 4;
          }
    }
    CloseHandle (hIn);
    CloseHandle (hOut);
    return 0;
}
```

Данный пример иллюстрирует некоторые особенности программирования в Win32.

1. <windows.h> включается всегда и содержит все определения функций и типов данных Win32.
2. Все объекты Win32 идентифицируются переменными типа HANDLE, и к большинству объектов применяется одна универсальная функция - CloseHandle.
3. Рекомендуется закрывать все открытые дескрипторы, когда они больше не требуются, чтобы освободить ресурсы. Тем не менее, при завершении процесса дескрипторы закрываются автоматически.
4. В Win32 определены многочисленные символические константы и флаги. Их имена обычно очень длинны и часто описывают их назначение. Типичное имя - INVALID_HANDLE_VALUE или GENERIC_READ.
5. Функции типа ReadFile и WriteFile возвращают логические значения, а не

количество байтов (которое передается в параметре). Это несколько изменяет
логику цикла. Конец файла обнаруживается по нулевому количеству байтов и
не приводит к неудаче операции.
6.Функция GetLastError позволяет в любой момент получить системные коды
ошибок в форме DWORD.

3. Копирование файлов с использованием функции полуфабриката Win32

Win32 содержит множество функций-полуфабрикатов, которые объединяют несколько функций для выполнения часто встречающихся задач. В некоторых случаях эти функции-полуфабрикаты могут улучшить быстродействие. Например, CopyFile значительно упрощает программу. В частности, здесь совершенно не нужно искать оптимальный размер буфера, который в двух предыдущих программах был произвольно задан в 256 байт.

Реализация программы копирования файлов для Win32 с использованием CopyFile достаточно проста и обеспечивает высокое быстродействие.

Обращение производится из командной строки: cpCF файл1 файл2 - копирует файл1 в файл2.

```
#include <windows.h>
#include <stdio.h>
int main (int argc, LPTSTR argv [])
{
  if (argc != 3)
      {
      printf ("Использование: cpCF file1 file2\n");
      return 1;
      }
      if (!CopyFile (argv [1], argv [2], FALSE))
      {
      printf ("Ошибка CopyFile: %x\n", GetLastError ());
      return 2;
      }
      return 0;
}
```

Приведенные программы копирования файлов демонстрируют многочисленные различия между библиотекой C и Win32. Примеры с использованием Win32 демонстрируют стиль программирования и со-

глашения Win32, но дают лишь первое представление о функциональных возможностях, доступных в Win32.

4.Содержание работы

1. Изучить программы копирования файлов.
2. Последовательно набрать и отладить программы копирования файлов в среде Visual Studio C++ .NET 2003.
3. Выполнить задание по копированию файла 1 в файл 2, выбрав произвольные имена файлов в различных программах.
4. Подготовить отчет по выполненной работе.
5. В отчете ответить на контрольные вопросы.

5. Порядок выполнения программ в среде Visual C++ .NET 2003

1. Запустить Microsoft Visual Studio C++ .NET 2003.
2. Создать проект под именем cpC. Для этого выполнить команды меню **File, New**, Project**.**

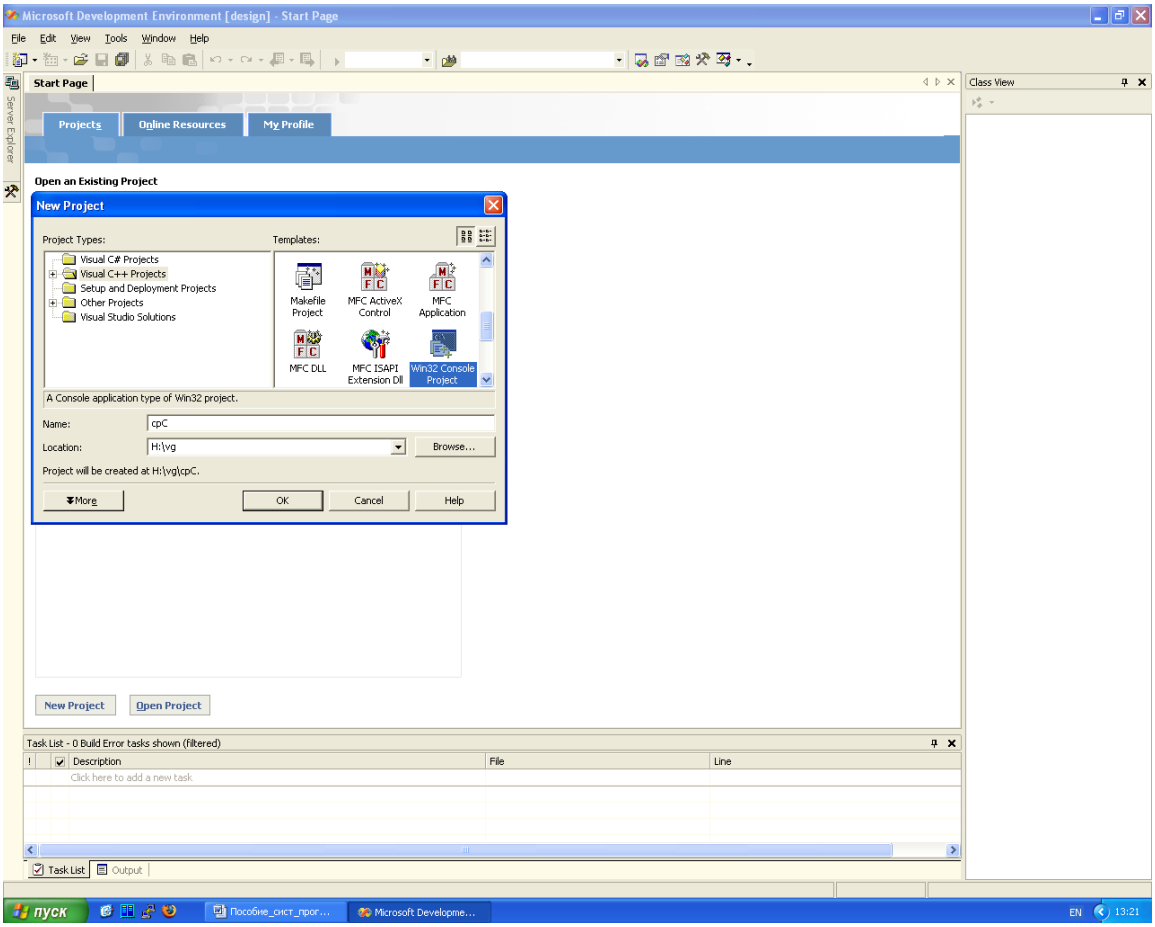


Рис.1. Создание нового проекта cpC консольного приложения

На появившейся вкладке **New Project** выбрать **Win32 Console Project**, в окне **Name:** набрать имя проекта **cpC**. В окне **Location**: набрать путь к папке, в которой будет записан проект и щелкнуть мышью OK. На появившейся вкладке щелкнуть мышью **Finish**.

3.На экране появляется текст файла cpC.cpp, в котором находится заготовка программы консольного приложения.

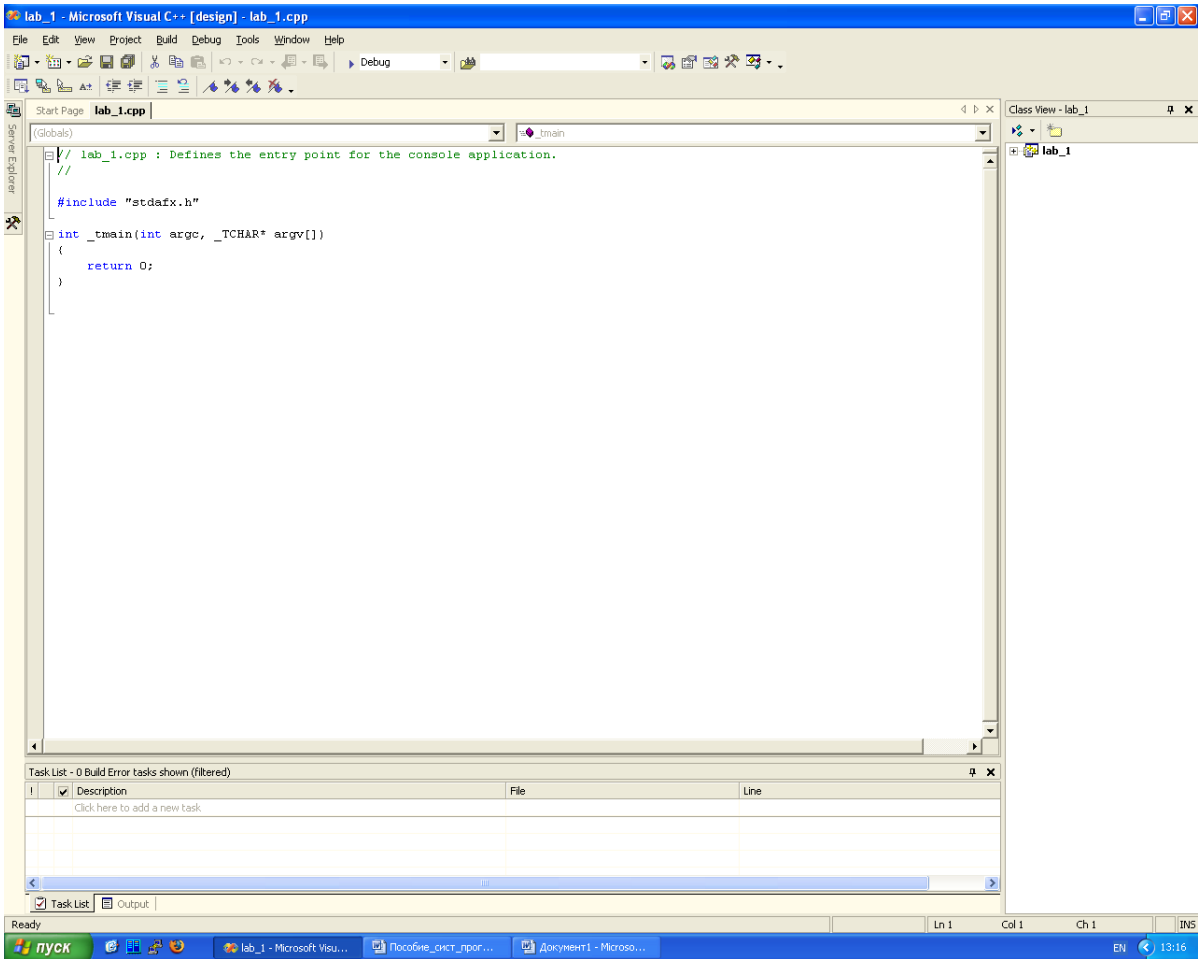


Рис 2. Содержимое файла cpC.cpp

4.После набора текста программы выполняется ее редактирование с использованием из меню **Build** команды **Build Solution**. В случае исправления всех ошибок в папке **Debug** проекта создается исполняемый файл с расширением .exe - **cpC.exe**, который необходимо выполнить.

5.Выполнение программы производится из командной строки.

Для этого можно использовать программу Far, при этом следует указать имя исходного файла, который должен быть создан до запуска программы cpC.exe, и имя файла, куда производится копирование.

6.Аналогично выполняется создание проектов cpW, cpCF, создание исполняемых файлов cpW.exe, cpCF.exe и выполнение программ копирования файлов с использованием Win32 и фунции-полуфабриката Win32.

6. Содержание отчета

В отчет должны быть включены следующие разделы:

1.Общие сведения об интерфейсе прикладного программирования API Win32.

2.Особенности использования API Win32 для копирования файлов.

3.Листинги различных программ копирования файлов.

4.Результаты копирования файлов, выполненных с использованием стандартной библиотеки C, с использованием Win32 и с использованием функции-полуфабриката Win32.

7.Контрольные вопросы

1. Что такое API Win32?
2. Какие операционные системы обслуживает API Win 32?
3.Какие особенности имеет API Win 32?
4. Какие преимущества программирования дает API 32?
5. Какой основной тип переменных используется в Win 32?
6. Для управления каких систем могут быть написаны программы с использованием Win32?
7. Что означает строка int main (int argc, LPTSTR argv []?
8. Поясните, какую функцию выполняет данный оператор: hIn = CreateFile (argv [1], GENERIC_READ, 0, NULL, OPEN_EXISTING, 0, NULL);
9. Поясните, какую функцию выполняет данный оператор: hOut = CreateFile (argv [2], GENERIC_WRITE, 0, NULL, CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL);
10.Поясните, какую функцию выполняет данный оператор: while (ReadFile (hIn, Buffer, BUF_SIZE, &nIn, NULL) && nIn > 0)
11. Поясните, какую функцию выполняет данный оператор: WriteFile (hOut, Buffer, nIn, &nOut, NULL);
12. Поясните, какую функцию выполняет данный оператор: if (!CopyFile (argv [1], argv [2], FALSE)).

Лабораторная работа №2

Вывод списка файлов и их атрибутов в заданном каталоге

1. Управление каталогами

Создание или удаление каталога выполняется двумя простыми функциями.

```
BOOL CreateDirectory (
    LPCTSTR lpszPath,
    LPSECURITY_ATTRIBUTES  lpsa)
BOOL RemoveDirectory (LPCTSTR  lpszPath)
```

lpszPath - указывает на строку с завершающим нулем, содержащую имя каталога, который должен быть создан или удален. Второй параметр (атрибуты безопасности) пока должен быть равен NULL. Удалить можно только пустой каталог.

Каждый процесс имеет текущий, или рабочий, каталог. Программист может и узнать текущий каталог, и установить его. Функция SetCurrentDirectory устанавливает каталог:

```
BOOL SetCurrentDirectory (LPCTSTR  lpszCurDir)
```

lpszCurDir - путь к новому текущему каталогу. Это может быть относительный путь или полностью уточненный путь, начинающийся либо с буквы диска и двоеточия, например D:\.

Если путь к каталогу - просто обозначение привода (например, A: или C:), текущим становится рабочий каталог указанного диска. Например, если рабочие каталоги установлены в последовательности

```
C:\MSDEV
INCLUDE
```

то в результате текущий каталог будет C: \MSDEV\INCLUDE.

Функция GetCurrentDirectory возвращает в буфер, заданный программистом, полностью уточненный путь.

```
DWORD GetCurrentDirectory (
    DWORD cchCurDir,
    LPTSTR lpszCurDir)
```

Возвращаемое значение - длина строки возвращенного пути или требуемый размер буфера, если он недостаточно велик; нуль при неудаче.

cchCurDir - длина в символах (не в байтах) буфера для имени каталога. Длина должна учитывать завершающий нулевой символ,

lpszCurDir - указывает на буфер, в который записывается строка пути.

Если буфер слишком мал для пути, возвращаемое значение сообщает его необходимую величину. Поэтому, при проверке на

неудачное выполнение функции должен проверяться как нулевой результат функции, так и результат, превышающий значение параметра cchCurDir.

Атрибуты файлов и работа с каталогами

В каталоге можно искать файлы и другие каталоги, удовлетворяющие указанному образцу имени, а также получать атрибуты файлов. Для этого необходим дескриптор поиска, который возвращается функцией FindFirstFile. Для получения нужных файлов служит функция FindNextFile, а для завершения поиска - FindClose.

```
HANDLE FindFirstFile (
        LPCTSTR lpszSearchFile,
        LPWIN32_FIND_DATA lpffd);
```

Возвращаемое значение: дескриптор поиска .
INVALID_HANDLE_VALUE указывает на неудачу.

FindFirstFile ищет соответствие имени как среди файлов, так и среди подкаталогов. Возвращаемый дескриптор используется в последующем поиске.

lpszSearchFile указывает на каталог или полное имя, которое может содержать подстановочные знаки (? и *).

lpffd указывает на структуру WIN32_FIND_DATA, которая содержит информацию о найденном файле или каталоге.

Структура WIN32_FIND_DATA определена следующим образом:

```
typedef struct WIN32_FIND_DATA {
DWORD dwFileAttributes;
FILETIME ftCreationTime;
FILETIME ftLastAccessTime;
FILETIME ftLastWriteTime;
DWORD nFileSizeHigh;
DWORD nFileSizeLow;
DWORD dwReserved0 ;
DWORD dwReservedl;
TCHAR cFileName [MAX_PATH];
TCHAR cAlternateFilename [14 ];}
WIN32 _FIND_DATA;
```

dwFileAttributes можно проверить на соответствие значениям, приведенным в описании CreateFile. Далее следуют три значения времени (время создания файла, время последнего доступа и последней записи). Поля размера файла, содержащие его текущую длину, не требуют пояснений,

cFileName - это не полное имя, а собственно имя файла,

cAlternateFile - версия имени файла для DOS формата 8.3 (включая точку). Это то же сокращение имени файла, которое отображается в проводнике Windows. Оба имени - строки с завершающим нулем.

Часто требуется найти в каталоге файлы, имена которых удовлетворяют образцу, содержащему подстановочные знаки ? и *. Для этого нужно получить из FindFirstFile дескриптор поиска, который содержит информацию об искомом имени, и вызвать функцию FindNextFile.

```
BOOL FindNextFile (
    HANDLE hFindFile,
    LPWIN32_FIND_DATA lpffd);
```

Функция FindNextFile возвращает FALSE либо при недопустимых параметрах, либо если соответствие данному образцу уже нельзя найти. В последнем случае GetLastError возвращает ERROR_NO_MORE_FILES.

Когда поиск закончен, необходимо закрыть дескриптор поиска. Нельзя использовать для этого функцию CloseHandle. (Это редкое исключение из правила, по которому CloseHandle пригодна для закрытия всех дескрипторов; закрытие дескриптора поиска вызовет исключительную ситуацию) Для закрытия дескрипторов поиска предназначена функция FindClose:

```
BOOL FindClose (
    HANDLE hFindFile);
```

Функция GetFileInformationByHandle позволяет получить ту же информацию для определенного файла, указав его открытый дескриптор.

Функции FindFirstFile и FindNextFile позволяют получать следующую информацию об атрибутах файла: флаги атрибутов, три штампа времени и размер файла. Для работы с атрибутами есть несколько других функций, включая ту, что позволяет их устанавливать, и они могут работать непосредственно с дескриптором открытого файла без просмотра каталога или указания имени файла. Другие функции служат для получения остальных атрибутов.

Функция GetFileAttributes по указанному имени файла и каталога возвращает только атрибуты.

```
DWORD GetFileAttributes (
    LPCTSTR lpszFileName);
```

Возвращаемое значение: атрибуты файла или OxFFFFFFFF в случае неудачи.

Атрибуты можно проверить на соответствие комбинациям определенных значений. Некоторые атрибуты, такие как флаг временного файла, устанавливаются при вызове CreateFile. Значения атрибутов могут быть следующими:
FILE_ATTRIBUTE_DIRECTORY
FILE_ATTRIBUTE_NORMAL
FILE_ATTRIBUTE_READONLY
FILE_ATTRIBUTE_TEMPORARY

Функция SetFileAttributes изменяет эти атрибуты в указанном по имени файле.

Теперь поясним функции управления файлами и каталогами. В листинге приведена версия программы вывода содержимого каталога, которая показывает время изменения файла и его размер, хотя может отображать только младшую часть значения размера файла.

Программа ищет в каталоге файлы, которые удовлетворяют образцу поиска. Для каждого найденного файла программа показывает имя и, если указана опция -1, атрибуты файла.

Большая часть листинга посвящена обходу каталога. Каждый каталог программа проходит дважды: один раз для обработки файлов и второй - для обработки подкаталогов, обеспечивая рекурсивную работу, когда указана опция – R.

Программа правильно работает при указании абсолютного пути, например:
<Имя программы> -R -1 C:\test*.txt. Программа обеспечит вывод списка файлов с расширением .txt и их атрибутов из каталога C:\test и подкаталогов.

Программа вывод списка файлов и обход каталога

```
//Директивы препроцессора
#include "stdafx.h"
//#include "EvryThng.h"
//Далее приводится текст библиотеки «EvryThng.h»
#include "stdafx.h"
#define WIN32_LEAN_AND_MEAN
#define NOATOM
#define NOCLIPBOARD
#define NOCOMM
#define NOCTLMGR
#define NOCOLOR
#define NODEFERWINDOWPOS
#define NODESKTOP
```

```
#define NODRAWTEXT
#define NOEXTAPI
#define	NOGDICAPMASKS
#define	NOHELP
#define	NOICONS
#define	NOTIME
#define	NOIMM
#define	NOKANJI
#define	NOKERNEL
#define	NOKEYSTATES
#define	NOMCX
#define	NOMEMMGR
#define	NOMENUS
#define	NOMETAFILE
#define	NOMSG
#define	NONCMESSAGE
#define	NOPROFILER
#define	NORASTEROPS
#define	NORESOURCE
#define	NOSCROLL
#define	NOSERVICE
#define	NOSHOWWINDOW
#define	NOSOUND
#define	NOSYSCOMMANDS
#define	NOSYSMETRICS
#define	NOSYSPARAMS
#define	NOTEXTMETRIC
#define	NOVIRTUALKEYCODES
#define	NOWH
#define	NOWINDOWSTATION
#define	NOWINMESSAGES
#define	NOWINOFFSETS
#define	NOWINSTYLES
#define	OEMRESOURCE
/* Envirmnt.h - определяются UNICODE _MT. */
/* Лучше и проще определить UNICODE в проекте. */
/* Вызовите команды Project... Settings ... C/C++ и в окне
Project Options внизу добавьте текст /D "UNICODE". */
 /* Сделайте то же для _MT и _STATIC_LIB. */
//#define UNICODE
#undef UNICODE
```

```
#ifdef UNICODE
#define _UNICODE
#endif
#ifndef UNICODE
#undef _UNICODE
#endif
//#define _MT
#define _STATICLIB
/* Определяйте _STATICLIB при формировании статической
библиотеки или компоновке с ней. */
//#undef_STATICLIB
#include <windows.h>
#include <tchar.h>
#include <stdio.h>
#include <io.h>
#include <stdlib.h>
#include <search.h>
#include <string.h>
/*  "Support.h"  */
/* Определения всех символьных констант и служебных функций
общего применения для всех программ примеров. */
/* ЛУЧШЕ ОПРЕДЕЛИТЬ СИМВОЛЫ UTILITY_EXPORTS И
_STATICLIB НЕ ЗДЕСЬ, А В ПРОЕКТЕ, ОДНАКО ИХ ОПИСАНИЯ
ПРИВОДЯТСЯ ЗДЕСЬ. */
/* Символ "UTILITY_EXPORTS" генерируется в Dev Studio при
создании проекта DLL с именем "Utility" и определяется в командной
строке C. */
//#define UTILITY_EXPORTS Закомментировано; определяйте
//       этот символ в проекте.
#define UTILITY_EXPORTS
#ifdef UTILITY_EXPORTS
#define LIBSPEC _declspec (dllexport)
#else
#define LIBSPEC _declspec (dllimport)
#endif
#define EMPTY _T ("")
#define YES _T ("y")
#define NO _T ("n")
#define CR OxOD
#define LF OxOA
#define TSIZE sizeof (TCHAR)
```

```
/* Пределы и константы. */
#define TYPE_FILE 1   /* Используется в Is, rm и IsFP. */
#define TYPE_DIR 2
#define TYPE_DOT 3
#define MAX_OPTIONS 20
/* Максимальное кол-во опций командной строки. */
#define MAX_ARG 1000
/* Максимальное кол-во параметров командной строки. */
#define MAX_COMMAND_LINE MAX_PATH+50
/* Максимальный размер командной строки. */
/* Функции общего применения. */
LIBSPEC BOOL ConsolePrompt (LPCTSTR, LPTSTR, DWORD, BOOL);
LIBSPEC BOOL PrintStrings (HANDLE, ...);
LIBSPEC BOOL PrintMsg (HANDLE, LPCTSTR);
LIBSPEC VOID ReportError (LPCTSTR, DWORD, BOOL);
```

//Начало программы

```
BOOL TraverseDirectory (LPCTSTR, DWORD, LPBOOL);
DWORD FileType (LPWIN32_FIND_DATA);
BOOL ProcessItem (LPWIN32_FIND_DATA, DWORD, LPBOOL);

DWORD Options (int argc, LPCTSTR argv [], LPCTSTR OptStr, ...)
/* argv - командная строка. Опции, начинающиеся с '-', если они есть,
помещаются в argv [1],  argv [2], ....
OptStr - текстовая строка, содержащая все возможные опции во
взаимно-однозначном соответствии с адресами булевых переменных в
списке параметров-переменных (...).
 Эти флаги устанавливаются тогда и только тогда, когда символ
соответствующей  опции встречается в argv [I], argv [2], ...
Возвращаемое значение - индекс в  argv первого параметра, не
принадлежащего  к опциям. */
{
va_list pFlagList;
LPBOOL pFlag;
int iFlag = 0, iArg;
va_start (pFlagList, OptStr) ;
while ((pFlag = va_arg (pFlagList, LPBOOL)) != NULL && iFlag < (int)
_tcslen (OptStr))
{
*pFlag = FALSE;
for (iArg = 1; !(*pFlag) && iArg < argc && argv [iArg] [0] = = '-';iArg++)
```

```
*pFlag = _memtchr (argv [iArg], OptStr [iFlag],
_tcslen (argv [iArg])) != NULL; iFlag++;
}
va_end (pFlagList);
for (iArg = 1; iArg < argc && argv [iArg] [0] = = '-'; iArg++);
return  iArg;
}

int  _tmain (int argc, LPTSTR argv [])
{
BOOL Flags [MAX_OPTIONS], ok = TRUE;
TCHAR PathName [MAX_PATH + 1], CurrPath [MAX_PATH + 1];
LPCTSTR  pFileName;
int i, FileIndex;


FileIndex = Options (argc, argv, _T ("R1"), &Flags [0], &Flags[1],
&Flags[2], NULL);
/* "Разбор" образца поиска на "путь" и "имя файла" */
GetCurentDirectory (MAX_PATH, CurrPath);
printf("argc=%d Index=%d\n",argc, FileIndex);
printf("STR=%s \n",CurrPath);
/* Сохраняем текущий путь. */
if (argc < FileIndex +1)
/* Путь не указан. Текущий каталог. */
ok = TraverseDirectory (_T ("*"), MAX_OPTIONS, Flags);
else
for (i = FileIndex; i < argc; i++)
{
/* Обрабатываем все пути в командной строке. */
        pFileName=argv[i];

ok = TraverseDirectory (pFileName, MAX_OPTIONS, Flags) && ok;
SetCurrentDirectory (CurPath);
 /* Восстанавливаем каталог. */
}
return ok ? 0 : 1;
}
static BOOL TraverseDirectory (LPCTSTR PathName, DWORD NumFlags,
LPBOOL Flags)
/* Обход каталога; выполняем Processltem для каждого найденного
```

```
соответствия. */
/* PathName: относительный или абсолютный путь для обхода. */
{
HANDLE SearchHandle;
WIN32_FIND_DATA FindData;
BOOL Recursive = Flags [0];
DWORD FType, iPass;
TCHAR CurrPath [MAX_PATH +1];
GetCurrentDirectory (MAX_PATH, CurrPath) ;
for (iPass = 1; iPass <= 2; iPass++)
{
        /* 1-й проход: список файлов. */
        /* 2-й проход: обход каталогов (если указана опция -R). */
        SearchHandle = FindFirstFile (PathName, &FindData);
        do
 {
FType = FileType (&FindData); /* Файл или каталог? */
if (iPass ==1) /* Выводим имя и атрибуты. */
ProcessItem (&FindData, MAX_OPTIONS, Flags);
if (Type == TYPE_DIR && iPass == 2 && Recursive)
{
        /* Обрабатываем подкаталог. */
_tprintf (_T ("\n%s\\%s:"), CurrPath, FindData.cFileName);
        /* Подготовка к обходу каталога. */
SetCurrentDirectory (FindData.cFileName);
TraverseDirectory (_T ("*"), NumFlags, Flags);
/* Рекурсивный вызов. */
SetCurrentDirectory (_T ("."));
}
}
while (FindNextFile (SearchHandle, &FindData));
FindClose (SearchHandle);
}
return TRUE;
}
static BOOL ProcessItem (LPWIN32_FIND_DATA pFileData, DWORD
NumFlags, LPBOOL Flags)
/* Вывод атрибутов файла или каталога. */
{
const TCHAR FileTypeChar [] = {' ','d'};
DWORD FType = FileType (pFileData);
```

```
BOOL Long = Flags [1];
SYSTEMTIME LastWrite;

if (FType != TYPE_FILE && FType != TYPE_DIR) return FALSE;
_tprintf (_T ("\n"));
if (Long)
{
/* Указана ли в командной строке опция "-1"? */
 _tprintf (_T ("%c"), FileTypeChar [FType - 1]);
 _tprintf (_T ("%10d"), pFileData->nFileSizeLow) ;
FileTimeToSystemTime (&(pFileData->ftLastWriteTime),&LastWrite);
        _tprintf (_T (" %02d/%02d/%04d %02d:%02d:%02d "),
LastWrite.wMonth, LastWritewDay,
LastWrite.wYear, LastWrite.wHour,
LastWrite.wMinute, LastWrite.wSecond);
}
 _tprintf (_T ("%s"), pFileData->cFileName);
 return TRUE;
}

static DWORD FileType (LPWIN32_FIND_DATA pFileDta)
 /* Поддерживаемые типы: TYPE_FILE (файл);
TYPE_DIR (каталог); TYPE DOT (каталог . или .. */
{
BOOL IsDir;
DWORD FType;
FType = TYPE_FILE;
IsDir = (pFileData->dwFileAttributes & FILE_ATTRIBUTE_DIRECTORY)
!= 0;
if (IsDir)
if (lstrcmp (pFileData->cFileName, _T (".")) = = 0
|| lstrcmp (pFileData->cFileName, _T ("..")) == 0)
        FType = TYPE_DOT;
else FType = TYPE_DIR;
return FType;
}
```

2.Содержание работы

1.Изучить программу вывода списка файлов и обход каталога.

2.Последовательно набрать и отладить программу вывода списка файлов и обход каталогов в среде Visual Studio C++ 2003.

3.Выполнить задание:

- вывод списка файлов по заданному шаблону из папки Debug;
- вывод списка файлов по заданному шаблону с аттрибутами, задавая ключ -1;
- вывод списка файлов по заданному шаблону и папки, созданной в папке Debug, запуская программу из командной строки, используя ключи -1, -R.

Файлы для поиска можно создать в текущей директории с помощью программы Far. Также с помощью Far можно создать новую директорию с файлами для поиска по шаблону.

4.Изучить основные функции, которые используются в данной программе.

5.Подготовить отчет по выполненной работе.

6.В отчете ответить на контрольные вопросы.

3. Порядок выполнения программ в среде Visual Studio C++ .NET 2003

1. Запустить Microsoft Visua Studio C++ .NET 2003.

2. Создать проект под именем **lab2**. Для этого выполнить команды меню **File, New**, **Project.** На появившейся вкладке выбрать **Win32 Console Project**, в окне **Name:** набрать имя проекта **lab2** и щелкнуть мышью OK. На появившейся вкладке щелкнуть мышью **Finish**. Если файла lab2.cpp на экране нет, то в меню **View** выбрать **Solution Explorer**. В папке **1ab2 files** выбрать папку **Source Files** и открыть файл программы **lab2**.**cpp**.

3.После набора текста программы выполняется ее компиляция с использованием меню **Build** команды **Rebuild Solution**. В случае исправления всех ошибок в папке **Debug** проекта создается исполняемый файл с расширением **lab2.exe**, который необходимо выполнить.

4. Выполнение указанной программы производится из командной строки для 3-х вариантов: поиск файлов в папке Debug по заданному шаблону; поиск файлов по заданному шаблону с выводом атрибутов файла, задавая ключ -1; поиск файлов позаданному шаблону в директории, созданной в папке Debug.

4. Содержание отчета

В отчет должны быть включены следующие разделы:

1.Общие сведения об атрибутах файлов и каталогов.

2.Методы получения атрибутов файлов и каталогов

3.Листинг программы вывода списка файлов и каталогов.

4.Пояснение программы вывода списка файлов и каталогов.

5.Письменные ответы на вопросы.

5. Контрольные вопросы

1. Поясните задачу функции: FindFirstFile и ее параметры.
2. Поясните задачу функци: FindNextFile и ее параметры.
3. Поясните задачу функции: FindClose и ее параметры.
4. Какую задачу выполняет функция GetFileAttributes и ее параметры?
5. Какова структура параметра lhffd?
6. Какой параметр функции FindFirstFile указывает на каталог?
7. Какие существуют функции для работы со временем?
8. Какие значения атрибутов имеют файлы и какой функцией они устанавливаются?
9. Какой функцией можно изменить атрибуты файлов?
10. Какая функция создает имена для временных файлов?

Лабораторная работа №3

Копирование нескольких файлов в стандартный вывод

1. Стандартные устройства Win32

Win32 имеет три стандартных устройства для ввода, вывода и сообщения об ошибках, для которых требуются дескрипторы.

Имеется функция для получения дескрипторов стандартных устройств.

HANDLE GetStdHandle (DWORD nStdHandle)

Возвращаемое значение: допустимый дескриптор, если функция завершается успешно; иначе INVALID _HANDLE _VALUE.

Параметры GetStdHandle

nStdHandle должен иметь одно из следующих значений:

- STD_INPUT_HANDLE;
- STD_OUTPUT_HANDLE;
- STD_ERROR_HANDLE.

Стандартные устройства обычно назначены консоли и клавиатуре. Стандартный ввод-вывод можно перенаправлять.

GetStdHandle не создает новый дескриптор стандартного устройства и не дублирует прежний. Последовательные вызовы с одним и тем же параметром возвращают одно и то же значение дескриптора. Закрытие дескриптора стандартного устройства делает это устройство недоступным для последующего использования, поэтому в программах часто открывают стандартное устройство, работают с ним, но не закрывают. Назначение стандартных устройств производится функцией:

BOOL SetStdHandle (DWORD nStdHandle, HANDLE hHandle)

Возвращаемое значение: TRUE или FALSE, в зависимости от успеха, или неудачи.

Параметры SetStdHandle

Возможные значения nStdHandle - те же, что в GetstdHandle. Параметр hHandle определяет открытый файл, который должен быть стандартным устройством.

Обычный метод перенаправления стандартного ввода-вывода в пределах процесса состоит в том, чтобы использовать последовательность SetStdHandle и GetStdHandle. Полученный в результате дескриптор используется в последующих операциях ввода-вывода.

Два имени файлов зарезервированы для консольного ввода (с клавиатуры) и консольного вывода: CONIN$ и CONOUT$. Первоначально стандартный ввод, вывод и выдача ошибок назначены на консоль. Консоль можно использовать независимо от любого перенаправления этих стандартных устройств; для этого нужно просто открыть дескрипторы CONIN$ ИЛИ CONOUT$ С ПОМОЩЬЮ CreateFile.

Для консольного ввода-вывода можно применить ReadFile и WriteFile, но лучше использовать специальные функции ReadConsole и WriteConsole. Основные их преимущества состоят в том, что эти функции обрабатывают универсальные символы (TCHAR), а не байты, а также учитывают режим консоли, установленный функцией SetConsoleMode.

BOOL SetConsoleMode (HANDLE hConsole, DWORD fdevMode)

Возвращаемое значение: TRUE, если функция завершается успешно.

hConsole- определяет буфер ввода или экран,который должен иметь аттрибут доступа GENERIC_WRITE,

fdevMode- определяет режим обработки символов. Значение каждого флага указывает, применяется ли этот флаг к консольному вводу или выводу.

Программа копирования нескольких файлов в стандартный вывод

```
/*При наличии в командной строке запуска программы имени одного или нескольких файлов программа перенаправляет их содержимое на экран. Если имя файла отсутствует, данные вводятся с клавиатуры и перенаправляются на экран */
#include "stdafx.h"
```

```
#include "EvryThng.h"
#define BUF_SIZE 0x200
static VOID CatFile (HANDLE, HANDLE);
DWORD Options (int argc, LPCTSTR argv [], LPCTSTR OptStr, ...)
/* argv - командная строка. Опции, начинающиеся с '-', если они есть, помещаются в argv [1],  argv [2], ....
OptStr - текстовая строка, содержащая все возможные опции  во взаимно-однозначном  соответствии с адресами булевых переменных в списке  параметров-переменных (...).  Эти флаги устанавливаются тогда и только тогда,  когда символ соответствующей  опции встречается в argv [1], argv [2], ...
  Возвращаемое значение - индекс в  argv первого параметра, не принадлежащего  к опциям. */
{
va_list pFlagList;
LPBOOL pFlag;
int iFlag = 0, iArg;
va_start (pFlagList, OptStr) ;
while ((pFlag = va_arg (pFlagList, LPBOOL)) != NULL && iFlag < (int) _tcslen (OptStr))
{
*pFlag = FALSE;
for (iArg = 1; !(*pFlag) && iArg < argc && argv [iArg] [0] == '-';iArg++)
*pFlag = _memtchr (argv [iArg], OptStr [iFlag],
_tcslen (argv [iArg])) != NULL; iFlag++;
}
va_end (pFlagList);
for (iArg = 1; iArg < argc && argv [iArg] [0] == '-'; iArg++);
return  iArg;
}

VOID ReportError (LPCTSTR UserMessage, DWORD ExitCode, BOOL PrintErrorMsg)
/*Универсальная функция для сообщения о системных ошибках. */
{
DWORD eMsgLen, LastErr = GetLastError ();
LPTSTR lpvSysMsg;
HANDLE hStdErr = GetStdHandle (STD_ERROR_HANDLE);
PrintMsg (hStdErr, UserMessage);
if (PrintErrorMsg)
{
```

```
eMsgLen = FormatMessage
(FORMAT_MESSAGE_ALLOCATE_BUFFER                                    |
FORMAT_MESSAGE_FROM_SYSTEM, NULL, LastErr,
MAKELANGID (LANG_NEUTRAL, SUBLANG_DEFAULT),
(LPTSTR) &lpvSysMsg, 0, NULL);
PrintStrings (hStdErr, _T ("\n"), lpvSysMsg, _T ("\n"), NULL);
/* Освобождаем блок памяти, содержащий сообщение об ошибке. */
HeapFree (GetProcessHeap (), 0, lpvSysMsg);
}
if (ExitCode > 0)
        ExitProcess (ExitCode);
else
        return;
}
 BOOL PrintStrings (HANDLE hOut,...)
{
/* Записывает сообщения в дескриптор вывода. */
DWORD MsgLen, Count;
LPCTSTR pMsg;
va_list pMsgList; /* Текущая строка сообщения. */
va_start (pMsgList, hOut); /* Начало обработки сообщений. */
while ((pMsg = va_arg (pMsgList, LPCTSTR)) != NULL)
{
MsgLen = _tcslen (pMsg);
/* WriteConsole работает только с дескрипторами консоли. */
if (!WriteConsole (hOut, pMsg, MsgLen, &Count, NULL)
/* Вызываем WriteFile только при неудаче WriteConsole */
&& !WriteFile (hOut, pMsg, MsgLen * sizeof (TCHAR), &Count, NULL))
return FALSE;
}
va_end (pMsgList);
return TRUE;
}
BOOL PrintMsg (HANDLE hOut, LPCTSTR pMsg)
/* Версия PrintStrings для одного сообщения. */
{
return PrintStrings (hOut, pMsg, NULL);
}

int _tmain (int argc, LPTSTR argv [])
{
```

```
HANDLE hInFile, hStdIn = GetStdHandle (STD_INPUT HANDLE);
HANDLE hStdOut = GetStdHandle (STD OUTPUT_HANDLE);
BOOL DashS;
int iArg, iFirstFile;
/* DashS устанавливается, только если указана опция -s. */
/* iFirstFile - индекс первого входного файла в argv []. */
iFirstFile = Options (argc, argv, _T ("s"), &DashS, NULL);
if (iFirstFile == argc)
{
/* В списке аргументов нет файлов. */
/* Используется стандартный ввод. */
CatFile (hStdln, hStdOut);
return 0;
}
/* Обрабатываем каждый входной файл. */
for (iArg = iFirtFile; iArg < argc; iArg++)
{
hInFile = CreateFile (argv [iArg], GENERIC_READ,
0, NULL, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);
if (hInFile ==INVALID_HANDLE VALUE && !DashS)
ReportError (_T ("Cat - ошибка открытия файла"), 1, TRUE);
CatFile (hlnFile, hStdOut);
CloseHandle (hlnFile);
}
return 0;
}
/* Функция, которая непосредственно выполняет работу:
читает входные данные и копирует их в стандартный вывод. */
static VOID CatFile (HANDLE hInFile, HANDLE hOutFile)
{
DWORD nIn, nOut;
BYTE Buffer [BUF_SIZE];
while (ReadFile (hInFile, Buffer, BUF SIZE, &nIn, NULL)
&& (nIn != 0)
&& WriteFile (hOutFile, Buffer, nIn, &nOut, NULL)) ;
return;
}
```

2.Содержание работы

1.Изучить программу копирования нескольких файлов в стандартный вывод.

2.Как указано в работе №2 создать проект консольного приложения и присвоить ему соответствующее имя.
3.Набрать и отладить программу копирования нескольких файлов в стандартный вывод.
4.В текущем директории с помощью программы Far создать папку и поместить в ней два текстовых файла.
5.Запустить программу из командной строки, передавая содержимое файлов на стандартный вывод.
6.Запустить программу из командной строки без указания имени файла, используя стандартный ввод с клавиатуры.
7.Изучить основные функции, которые используются в данной программе: BOOL Options, VOID CatFile, VOID ReportError, BOOL PrintStrings, BOOL PrintMsg.
8.Подготовить отчет по выполненной работе.
9.В отчете ответить на контрольные вопросы.

3. Порядок выполнения программ в среде Visual Studio C++ .NET 2003

1. Запустить Microsoft Visual Studio .NET C++ 2003.
2. Создать проект под именем **lab3**. Для этого выполнить команды меню **File, New**. На появившейся вкладке выбрать **Win32 Console Project**, в окне **Name:** набрать имя проекта **lab3** и щелкнуть мышью OK. На появившейся вкладке щелкнуть мышью по кнопке **Finish**. После щелчка по папке Project в правой части окна MDE отобразится ее содержимое: 3 папки – **Source Files** (исходные файлы), **Header Files** (заголовочные файлы) и **Resource Files** (файлы ресурсов), а также файл ReadMe.txt. Откройте папку **Source Files** и дважды щелкните на файл .cpp. Тогда откроется окно, в котором выполняется набор текста программы.
3.После набора текста программы выполняется ее редактирование с использованием меню **Build** команды **Rebuild Solution**. В случае исправления всех ошибок в папке **Debug** проекта создается исполняемый файл **lab3.exe**, который необходимо выполнить. Выполнение указанной программы производится из командной строки, задавая при этом после имени программы имя файла, который копируется на стандартное устройство вывода.

4. Содержание отчета

В отчет должны быть включены следующие разделы:
1.Общие сведения о стандартных устройствах ввода-вывода.
2.Способы получения дескрипторов стандартных устройств.

3.Листинг программы копирования файлов в стандартный вывод.
4.Пояснение процедур программы BOOL Options, VOID CatFile, VOID ReportError, BOOL PrintStrings, BOOL PrintMsg.
5.Письменные ответы на вопросы.

5. Контрольные вопросы

1.Поясните функцию: GetStdHandle и ее параметры.
2.Поясните функцию: SetStdHandle и ее параметры.
3.Поясните функцию: SetConsoleMode и ее параметры.
4.Какую задачу выполняет функция BOOL Options?
5.Какую задачу выполняет функция VOID CatFile?
6.Какую задачу выполняет функция VOID ReportError ?
7.Какую задачу выполняет функция BOOL PrintStrings?
8.Как составляется командная строка при перенаправлении вводимых с клавиатуры символов на экран?
9.Как составляется командная строка при выводе содержимого двух файлов на экран?

Лабораторная работа №4

Последовательная обработка файлов с использованием отображения в память

1.Создание объекта отображения файла

Объекты отображения файлов в память могут получать имена, поэтому они доступны другим процессам для разделения памяти. Также отображаемый объект имеет защиту, атрибуты безопасности и размер. Для создания объекта отображения в память используется функция:

```
HANDLE  CreateFileMapping (
    HANDLE  hFile,
    LPSECURITY_ATTRIBUTES   lpsa,
    DWORD  fdwProtect;
    DWORD dwMaximumSizeHigh,
    DWORD dwMaximumSizeLow,
    LPCTSTR  lpszMapName);
```

Возвращаемоезначение:дескриптор отображения файла или NULL при неудаче.

hFile — дескриптор открытого файла с флагами защиты, совместимыми с параметром fdwProtect.

Тип LPSECURITY_ATTRIBUTES позволяет защитить объект отображения. Параметр fdwProtect определяет доступ к отображенному

файлу с помощью описанных ниже флагов. Для специальных целей разрешены дополнительные флаги.

Установленный флаг PAGE_READONLY означает, что программа может только читать страницы в отображенной области и не может записывать или исполнять их. Файл hFile должен быть открыт с правом доступа GENERIC_READ.

- Флаг PAGE_READWRITE предоставляет полный доступ к объекту, если файл hFile открыт с правами доступа GENERIC_READ и GENERIC_WRITE.
- Флаг PAGE_WRITECOPY определяет, что при изменении содержимого отображенной памяти собственная (для данного процесса) копия записывается в файл подкачки, а не в исходный файл.
- Параметры dwMaximumSizeHigh и dwMaximumSizeLow определяют размер объекта отображения. Если указан нуль, используется текущий размер; при использовании файла подкачки унеобходимо обязательно определять его размер. Если ожидается, что размер файла увеличится, необходимо использовать ожидаемый размер, и при необходимости будет немедленно установлен нужный размер файла. Нельзя отображать область файла за указанной границей - объект отображения не может расти.
- lpszMapName указывает имя объекта отображения, что позволяет другим процессам совместно использовать объект. Регистр символов в имени не учитывается. Если разделение памяти не используется, указывается значение NULL.

Об ошибке сообщает возвращаемое значение NULL (а не INVALID_HANDLE_VALUE).

Указав имя существующего объекта отображения, можно получить дескриптор отображения файла. Имя должно быть получено предшествующим вызовом функции CreateFileMapping. Два процесса могут совместно использовать память, разделяя отображение файла. Первый процесс создает отображение файла, а следующий открывает это отображение, используя имя. Если названного объекта не существует, открыть его не удастся.

```
HANDLE OpenFileMapping (
    DWORD dwDesiredAccess,
    BOOL bInheritHandle,
    LPCTSTR lpName);
```

Возвращаемое значение: дескриптор отображения…….файла или NULL при неудаче.

dwDesiredAccess использует тот же набор флагов, что и параметр fdwProtect функции CreateFileMaping.

lpName - имя, полученное вызовом функции CreateFileMapping.

Наследование дескриптора - параметр bInheritHandle.

Функция CloseHandle уничтожает дескрипторы отображения.

2.Выделение виртуального адресного пространства для объекта отображения

Следующий этап - выделение виртуального адресного пространства и отображение его в файл через объект отображения. С точки зрения программиста, такое выделение памяти аналогично действию функции HeapAlloc, но намного грубее, более крупными частями. Возвращается указатель на выделенный блок (или образ файла); отличие состоит в том, что выделенный блок отображается в указанный пользователем файл, а не в файл подкачки обмена.

```
LPVOID MapViewOfFile(
    HANDLE hMapObject,
    DWORD  fdwAccess,
    DWORD dwOffsetHigh,
    DWORD dwOffsetLow,
    SIZE_T  cbMap);
```

Возвращаемое значение: начальный адрес блока (образ файла) или NULL при неудаче.

hMapObject - указывает объект отображения файла, полученный от функций CreateFileMapping или OpenFileMapping.

Значение параметра fdwAccsess должно соответствовать правам доступа объекта отображения. Возможны три значения флагов:

FILE_MAP_WRITE,

FILE_MAP_READ,

FILE_MAP_ALL_ACCESS.

Параметры dwOffsetHigh и dwOffsetLow определяют начальное положение области отображения. Начальный адрес должен быть кратным 64К. Для отображения от начала файла используется нулевое значение смещения.

cbMap указывает размер отображаемой области в байтах. Его нулевое значение при вызове функции MapViewOfFile указывает на то, что весь файл должен быть отображен.

Освобождение образа файла выполняется так же, как и освобождение памяти, выделенной в куче, т.е. функцией HeapFree.

3.Программа работы с файлом, отображенным в память

Основным преимуществом отображения в память является возможность использования удобных алгоритмов для обработки файлов.

В приведенной программе осуществляется отображение файла в память. Имя файла задается при запуске программы из командной строки. Кроме того, в командной строке задается имя файла, в котором размещается фрагмент данных для поиска его в исходном файле. Программа «просматривает» исходный файл и выдает результат сравнения. В случае, если заданный фрагмент находится, то выдается соответствующее сообщение.

Затем исходный файл отправляется на стандартный вывод функцией _tprintf.

Листинг программы

```
#include "stdafx.h"
#include "EvryThng.h"
#define BUF_SIZE 160
#define rmaxf1 40
#define rmaxf2 10
#include <process.h>
#include <stdarg.h>
#include <string.h>
void KCompare (CHAR , CHAR , int);
void KCompare ( CHAR string1[rmaxf1],  CHAR string2[rmaf2], int KEY)
{
        char tmp[20]; int result;
        result=_tcsncmp (( CHAR *) string1, ( CHAR *) string2, KEY);
        if (result==0)
        { strcpy( tmp, "equal to" );
        printf( "Result:\t\tString 1 is %s string 2\n\n", tmp );
        }
        else
        { strcpy( tmp, "noequal to" );
        printf( "Result:\t\tString 1 is %s string 2\n\n", tmp );
        }
}
int _tmain (int argc, LPTSTR argv [])
{
HANDLE hFile, hFile2, hMap;
LPVOID  pFile;
```

```
CHAR string1[rmaxf1], string2[rmaxf2];
CHAR Buffer1[BUF_SIZE], Buffer2[BUF_SIZE];
DWORD FsLow, nf2;
int i, j, k, KEY;
BOOL f2;
LPCTSTR pF;
hFile = CreateFile (argv[1], GENERICREAD, 0, NULL,
OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);
FsLow = GetFileSize (hFile, NULL);
printf("Fs=%d\n",(int)FsLow);
hMap = CreateFileMapping (hFile, NULL, PAGE_READONLY,
0, 0, NULL);
pFile = MapViewofFile (hMap, FILE_MAP_READ, 0, 0, 0);
pF=pFile;
for (i=0; i< (int)FsLow; i++)
{
Buffer1[i]=*pF;
pF++;
}
for (i=0; i< (int)FsLow; i++)
printf("%c",Buffer1[i]);
printf("\n");
hFile2 = CreateFile (argv[2], GENERIC_READ, 0, NULL,
OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);
f2=ReadFile(hFile2, Buffer2, BUF_SIZE, &nf2, NULL);
KEY=GetFileSize (hFile2, NULL);
printf("KEY=%d\n", (int)KEY);
for (i=0; i< (int)KEY; i++)
printf("%c",Buffer2[i]);
printf("\n");
for (i=0; i< (int)KEY; i++)
string2[i]=Buffer2[i];
        k=0;
        do
        {
        i=k;
        do
        {
        for (j=0; j<(int)KEY;j++)
        {
        string1[j]=Buffer1[j+i];
```

```
        printf("%c",string1[j]);
        }
        printf("\n");
        getchar();
        KCompare(string1, string2, (int)KEY);
        i=i+KEY;
        }
        while (i<(int)FsLow-(int)KEY+1);
        k=k+1;
        }
        while (k<(int)FsLow-(int)KEY+1);
/* Выводим исходный файл. */
_tprintf(_T("%s"),pFile);
UnmapViewOfFile (pFile);
CloseHandle (hMap);
CloseHandle (hFile);
CloseHandle (hFile2);
return 0;
}
```

4.Содержание работы

1.Изучить основные функции отображения объекта в память: CreateFileMapping, OpenFileMapping, MapViewOfFile и их параметры.
2.Изучить функцию выделения виртуального адресного пространства и отображения его в файл через объект отображения MapViewOfFile.
3.Набрать и отладить программу сортировки отображенного в память файла в среде Visual Studio C++ 2003.
4.Выполнить программу, задавая исходные данные из командной строки. В командной строке указывается имя программы, через пробел имя исходного файла для поиска фрагмента, затем имя файла, в котором находится фрагмент.
5.Разобраться с работой программы и объяснить ее функционирование.
6.Подготовить отчет по выполненной работе.
7.В отчете ответить на контрольные вопросы.

5.Порядок выполнения программ в среде Visual Studio C++ .NET 2003

1. Запустить Microsoft Visual Studio C++ .NET 2003.
2. Создать проект консольного приложения под именем **lab4**.
3. Набрать текст программы отображения файла в памяти и поиска в нем заданного шаблона.

4.Провести отладку программы, создать исполняемый файл.
5. Запустить программу из командной строки, задавая имя файла имя исходного файла и имя файла, в котором находится фрагмент для поиска.

6. Содержание отчета

В отчет должны быть включены следующие разделы:
1.Общие сведения о механизме отображения объекта в памяти.
2.Основные функции отображения объекта в памяти и их параметры.
3.Листинг программы, с отображением файла в память.
4.Пояснения работы программы.
5.Письменные ответы на вопросы.

7.Контрольные вопросы

1.Какую фунцию выполняет CreateFileMapping?
2.Какую фунцию выполняет OpenFileMapping?
3.Какую фунцию выполняет MapViewOfFile??
4.Поясните работу функции KeyCompare.
5.Как запускается программа использования файла, отображенного в памяти?
6.Поясните процесс работы программы поиска в исходном файле заданного фрагмента.

Лабораторная работа №5

Использование динамических библиотек для создания приложений

Наиболее прямой путь создания программы - собрать исходный код всех функций, скомпилировать его и скомпоновать все в единый исполняемый образ. Эта монолитная модель с одним исполняемым образом проста, но обладает рядом недостатков.

Динамические библиотеки могут быть использованы для создания разделяемых библиотек. Одной динамической библиотекой пользуются несколько программ, а в память загружается только одна ее копия. Все программы отображают адресные пространства своих процессов в код библиотеки, при этом каждый поток будет иметь собственную копию неразделяемой памяти в стеке.

1.Неявное связывание

Неявное связывание, или связывание во время загрузки - более простой из двух способов. При использовании языка Microsoft Visual C++ для этого необходим ряд шагов:

1. Функции для новой библиотеки собираются и компонуются как библиотека DLL, а не как, например, консольное приложение.
2. В процессе компоновки создается библиотечный файл .LIB, который является суррогатом для основного кода. Этот файл необходимо поместить в каталог вызывающей программы.
3. Процесс компоновки создает и файл .DLL, содержащий исполняемый образ. Обычно этот файл помещается в тот же каталог, что и приложение, которое будет его использовать, а приложение загружает .DLL во время инициализации.

2. Экспорт и импорт функций

Наиболее существенное изменение, необходимое для переноса функции в динамическую библиотеку - это ее определение как экспортируемой. Это достигается путем использования модификатора для функции, помещаемой в динамическую библиотеку как показано ниже:

```
_declspec  (dllexport)
DWORD MyFunction  ( . . . ) ;
```

_declspec – модификатор функции,

MyFunction – функция, помещаемая в библиотеку.

В этом случае процесс компоновки создаст файл библиотеки .DLL и файл .LIB. Файл .LIB - это суррогат библиотеки, который должен быть связан с вызывающей ее программой для разрешения внешних ссылок и создания действительных связей с файлом .DLL во время загрузки и подключаемый на стадии компановки.

Вызывающая функцию программа должна определить импортируемую функцию путем использования модификатора

```
_declspec (dllimport)
DWORD MyFunction (...);
```

При компоновке вызывающей библиотеку программы, т.е. перед созданием .exe файла, необходимо в меню Project -->Setting на вкладке Link в окне Project_Options набрать путь и имя файла библиотеки MyFunction.lib. После этого необходимо убедиться в том, что файл библиотеки .DLL доступен ей. Обычно это обеспечивается помещением файла .DLL в тот же каталог, в котором находится исполняемый файл.

3.Явное связывание

Явное связывание, или связывание во время выполнения, несколько сложнее и требует от программы специального запроса для загрузки библиотеки (функция LoadLibrary) или ее выгрузки (функция FreeLibrary). Затем программа получает адрес нужной точки входа и использует его как указатель в вызове функции. В вызывающей программе функция не объявляется; вместо этого необходимо объявить переменную как указатель на функцию. Поэтому при компоновке программы нет потребности в файле библиотеки. Необходимы три функции: LoadLibrary, GetProcAddress и FreeLibrary.

HINSTANCE LoadLibrary (LPCTSTR lpLibFileName);

Возвращаемый дескриптор (типа HINSTANCE, а не HANDLE) не примет значения NULL в случае неудачи. Расширение .DLL в имени файла не требуется. С помощью функции LoadLibrary можно загрузить и файл типа .EXE. Так как динамические библиотеки разделяемые, система ведет учет ссылок на каждую DLL (их число увеличивается при вызове функции LoadLibrary), поэтому не требуется отображения фактически существующего файла. Даже если файл DLL найден, функция LoadLibrary выполнится неудачно в случае, если библиотека неявно связана с другой библиотекой, которая не может быть найдена.

Аналогичная функция LoadLibraryEx имеет несколько флагов, используемых для указания альтернативных путей поиска и загрузки библиотеки как файла данных.

После работы с данным экземпляром или для загрузки его другой версии необходимо освободить дескриптор библиотеки, освобождая тем самым ресурсы, включая и виртуальное адресное пространство, занятое библиотекой. Если счетчик ссылок указывает на то, что библиотека используется другим процессом, она остается загруженной в память:

BOOL FreeLibrary (HINSTANCE hLibModule);

После загрузки динамической библиотеки и до ее выгрузки можно получить адреса точек входа с использованием функции GetProcAddress.

FARPROC GetProcAddress (HMODULE module, LPCSTR lpProcName);

Параметр hModule, несмотря на другой тип (тип HINSTANCE определен как HMODULE), является экземпляром, получаемым от функций LoadLibrary или GetModuleHandle, которые здесь не описаны. Параметр lpProcName, который не может быть строкой стандарта Unicode, является именем точки входа. В случае неудачи возвращаемое функцией значение будет NULL.

Используя функцию GetModuleFileName, можно получить имя файла, связанного с дескриптором hModule. И наоборот, по заданному имени файла (файл типа .exe или .dll) функция GetModuleHandle возвращает дескриптор, связанный с файлом, если тот был загружен текущим процессом.

Функция входа в библиотеку также выполняется при создании или завершении процессом новых потоков.

Ниже приводится пример программы преобразования файла из формата ASCII в формат Unicode, используя создание библиотеки с неявным связыванием.

4.Программа использования функции из динамической библиотеки

Основная программа

```
#include   "EvryThng.h"
_declspec (dllimport)
BOOL Asc2Un(LPCTSTR, LPCTSTR, BOOL);
int  _tmain (int argc, LPTSTR argv [])
{
BOOL Output;
/*  Вызываем функцию. */
Output=Asc2Un(argv [1], argv [2], FALSE);
return 0;
}
```

Программа включения в DLL библиотеку функции Asc2Un

```
#include "EvryThng.h"
#define BUF_SIZE 256
_declspec (dllexport)
BOOL Asc2Un (LPCTSTR, LPCTSTR, BOOL);

BOOL Asc2Un (LPCTSTR fin, LPCTSTR fOut, BOOL bFaillfExists)
{
/* Функция копирования файла ASCII в Unicode на базе CopyFile. */
HANDLE hIn, hOut;
DWORD fdwOut, nIn, nOut, iCopy;
CHAR aBuffer [BUF_SIZE];
WCHAR uBuffer [BUF_SIZE];
BOOL WriteOK = TRUE;
hIn = CreateFile (fin, GENERIC_READ, 0, NULL, OPEN_EXISTING,
FILE_ATTRIBUTE_NORMAL, NULL);
/* Определяем действие CreateFile, если выходной файл уже существует.*/
```

```
fdwOut = CREATE_NEW||CREATE_ALWAYS;
hOut = CreateFile (fOut, GENERIC_WRITE, 0, NULL,
fdwOut, FILE_ ATTRIBUTE_NORMAL, NULL);
while (ReadFile (hIn, aBuffer, BUF_SIZE, &nIn, NULL) && nIn > 0 &&
WriteOK)
{
for (iCopy = 0; iCopy < nIn; iCopy++)
/* Преобразование каждого символа. */
uBuffer [iCopy] = (WCHAR) aBuffer [iCopy];
   WriteOK = WriteFile (hOut, uBuffer, 2 * nIn, &nOut, NULL); }
CloseHandle (hIn);
CloseHandle (hOut);
return WriteOK;
}
```

5.Содержание работы

6. Изучить основные способы создание динамических библиотек с неявным и явным связыванием.
7. Изучить процедуру экспорта и импорта функций в динамическую библиотеку при неявном связывании.
8. Изучить функции, которые используются при явном связывании: LoadLibrary, FreeLibrary и GetProcAddress.
9. Скопировать и отладить программу с использованием библиотеки с неявным связыванием.
10. Создать .DLL модуль для функции Asc2Un, листинг которой приведен ниже основной программы.
11. Разобраться с работой основной программы и объяснить ее функционирование.
7. Подготовить отчет по выполненной работе.
8. В отчете ответить на контрольные вопросы.

6. Порядок выполнения программ в среде Visual Studio C++ .NET 2003

1. Запустить Microsoft Visual Studio C++ /NET 2003.
2. Создать проект с именем **Asc2Un** формирования функции **Asc2Un** в динамическую библиотеку, используя вкладку **MFC DLL**. Для этого нужно в меню **File** выбрать **New**, затем **Project** и активировать иконку **MFC DLL**, в строке **Nam**e: набрать **Asc2Un**, в стоке **Location**: указать путь к папке, куда бубет записан проект и нажать кнопку **Finish**.

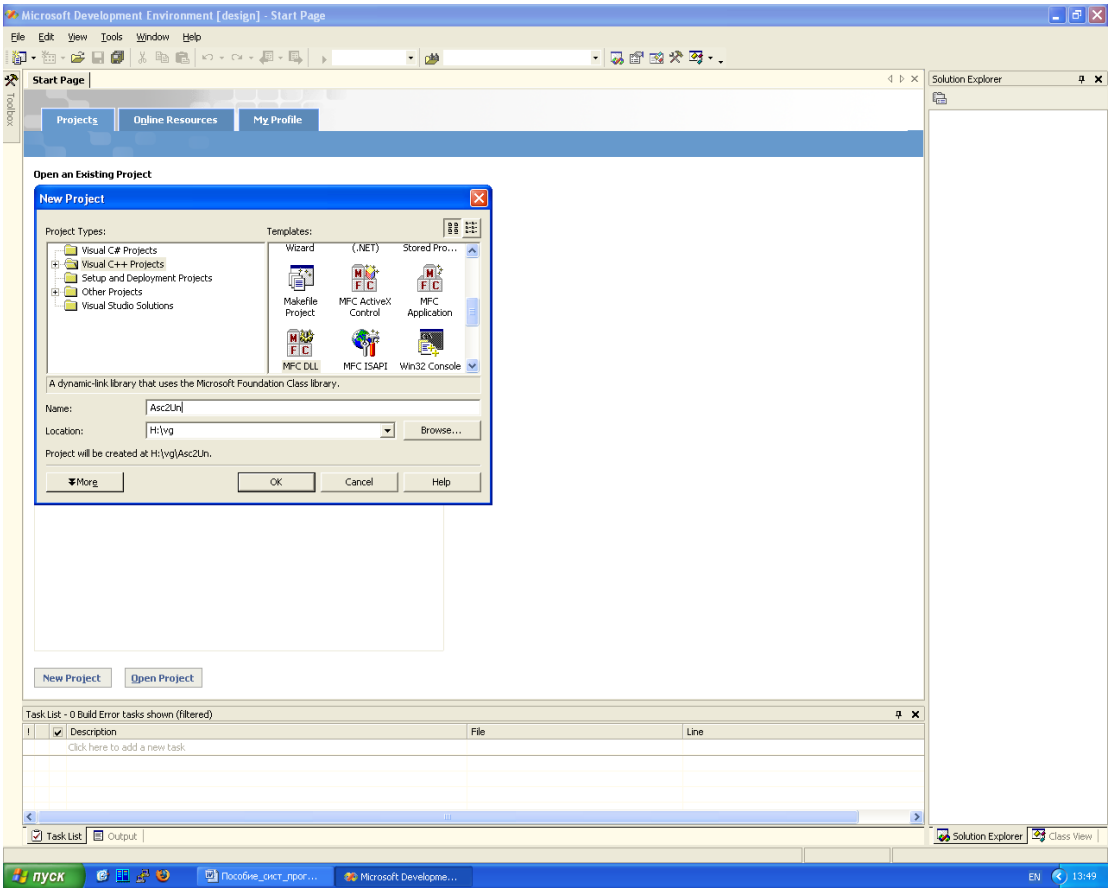


Рис.5. Проект Asc2Un MFC DLL

3.На экране должен появиться текст файла **Asc2Un.cpp**. Если его нет, то нужно войти в меню **View** и выбрать строку **Solution Explorer** и в папке **Asc2Un** найти файл **Asc2Un.cpp**.

4. Набрать программу функци Asc2Un и разместить в файле Asc2Un.cpp после **#endif**. Создать динамическую библиотеку .DLL. Для этого следует выбрать в меню **Project**, **Properties** вкладку **C/C++** ,**Code Generation** . В окне **Runtime Library** выбрать **Multithreaded Debug DLL**.

5. Создать проект для консольного приложения Win32 **Console Project** для основной программы и набрать основную программу.

6. При создании .exe файла основной программы в меню **Project**, **Properties** открыть **C/C++** выбрать **Command Line** и в окне **Additional Options** вписать H:\путь\...\Asc2Un.lib.

7. Перед запуском основной программы скопировать в папку Debug проекта консольного приложения файл Asc2Un.dll.

8. Запустить программу из командной строки, задавая имя файла, в котором находится исходный файл и далее имя файла, куда следует поместить преобразованный файл.

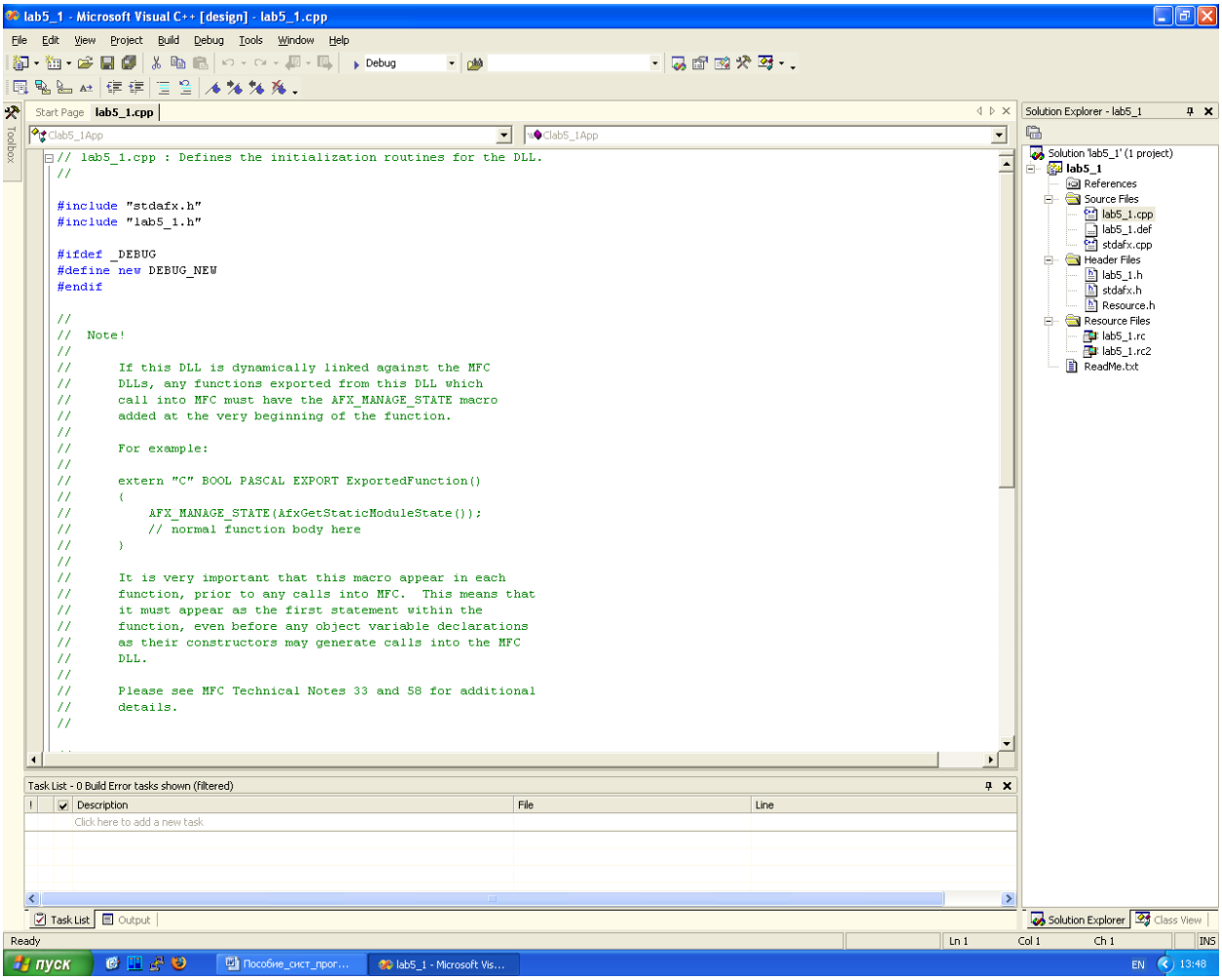


Рис.6. Содержимое файла Файл Asc2Un.cpp

7.Содержание отчета

В отчет должны быть включены следующие разделы:

1. Общие сведения о создании динамической библиотеки с неявным и явным связыванием.
2. Пояснить механизм экспорта и импорта функций в библиотеку с использованием модификаторов _declspec (dllexport), _declspec (dllimport).
3. Пояснить функции LoadLibrary, FreeLibrary и GetProcAddress.
4. Привести листинг основной программы и функции преобразования файла Asc2Un и результаты преобразования файла.
5. Письменные ответы на вопросы.

8.Контрольные вопросы

1.Какие способы создания динамических библиотек существуют?

2.Опишите процесс неявного связывания функций в динамической библиотеке.
3.Как осуществляется процесс экспорта, импорта функций в динамическую библиотеку при неявном связывании?
4.Опишите функции, которые используются при явном связывании функций в библиотеке (LoadLibrary, FreeLibrary, GetProcAddress).
5.Как создается .DLL файл для функции Asc2Un?
6.Как в вызывающей программе импортируется функция из динамической библиотеки?
7.Поясните работу программы преобразования файла из кодировки ASCII в кодировку Unicode.

Лабораторная работа №6

Многопроцессная обработка данных

Процессы и потоки в Windows

Каждый процесс содержит один или более потоков. Поток в Windows - основная единица выполнения. Планирование потоков проводится на основе обычных факторов: доступности ресурсов, таких как процессоры и физическая память, приоритетов, справедливости распределения ресурсов и т.д. Windows 2000 и NT поддерживают симметричную многопроцессную обработку, поэтому потоки могут распределяться по отдельным процессам.

Все потоки процесса совместно используют код, глобальные переменные, строки окружения и ресурсы. Каждый поток планируется независимо. Поток включает следующие элементы:

- Стек вызовов процедур, прерываний, обработчиков исключений и автоматических данных.
- Локальная память потока (Thread Local Storage - TLS) - массивы указателей,

которые дают процессу возможность выделять память для создания собственного уникального окружения данных.

- Параметр стека, полученный от потока, создавшего данный, и обычно уникальный для каждого потока,
- Структура контекста, управляемая ядром, со значениями аппаратных регистров.

1.Создание процессов

Функция CreateProcess - основная функция для управления процессами в Win32. Она создает процесс с одним потоком. Так как процесс требует наличия кода, в вызове функции CreateProcess необходимо указывать имя исполняемого файла программы.

Функция CreateProcess для обеспечения гибкости и мощности имеет десять параметров. Функция CreateProcess создает новый процесс с первичным потоком, функция CloseHandle лишь удалит ссылку на поток. Структура функции CreateProcess имеет следующий вид:

```
BOOL CreateProcess   (
        LPCTSTR lpszImageName,
        LPTSTR lpszCommandLine,
        LPSECURITY_ATTRIBOTES  lpsaProcess,
        LPSECURITY_ATTRIBUTES  lpsaThread,
        BOOL fInheritHandles,
        DWORD  fdwCreate,
        LPVOID  lpvEnvironment,
        LPCTSTR  lpszCurDir,
        LPSTARTUPINFO  lpsiStartInfo,
        LPPROCESS_INFORMATION  lppiProcInfo);
```

Возвращаемое значение: TRUE только в случае, если процесс и поток были успешно созданы.

lpszImageName, lpszCommandLine указывают на исполняемую программу и параметры командной строки так, как это указано в разделе определение исполняемого образа и командной строки,

lpsaProcess и lpsaThread указывают на структуры атрибутов безопасности процесса и потока. В случае если их значение NULL, используются атрибуты по умолчанию,

fInheritHandles определяет, должен ли новый процесс наследовать копии открытых наследуемых дескрипторов (файлов, отображений и т.д.) процесса, вызвавшего функцию. Унаследованные дескрипторы имеют те же атрибуты, что и их оригиналы;

fdwCreate может объединять несколько флагов:

- CREATE_SUSPENDED первичный поток находится в состоянии ожидания и будет запущен только после вызова функции ResumeThread.
- DETACHED_PROCESS и CREATE_NEW_CONSOLE - взаимоисключающие флаги, которые нельзя установить оба сразу. Первый флаг создает процесс без консоли, а второй предоставляет консоль новому процессу. Если ни один из этих флагов неустановлен, процесс наследует консоль родителя.

• CREATE_NEW_PROCESS_GROUP определяет новый процесс как корневой для новой группы процессов. Если процессы группы совместно используют одну консоль, то все они получают сигнал управления консолью (<Ctrl+C> или <Ctrl+Break>).

lpvEnvironment указывает блок окружения для нового процесса. Если этот параметр имеет значение NULL, то используется окружение родительского процесса. Блок окружения содержит строки имен и значений, такие как путь поиска.

lpszCurDir определяет диск и каталог для нового процесса. Если его значение NULL, то будет использован рабочий каталог родительского процесса.

lpsiStartInfo указывает вид основного окна программы и дескрипторы стандартных устройств для нового процесса. Обычно используется информация родителя, полученная вызовом функции GetStartupInfo. Перед вызовом функции CreateProcess структуру STARTUPINFO можно также обнулить, а затем для указания стандартных дескрипторов ввода, вывода и сообщений об ошибках установить стандартные значения соответствующих полей (hStdln, hStdOut и hStdErr) структуры STARTUPINFO. Для того чтобы это работало, необходимо присвоить другому полю структуры STARTUPINFO - dwFlags значение STARTF_USESTDHANDLES, а затем установите все дескрипторы, необходимые новому процессу. Убедитесь, что дескрипторы могут быть унаследованы и в функции CreateProcess установлены флаги fInheritProcess.

lppiProcinfo определяет структуру, в которую будут помещены дескрипторы и идентификаторы для созданных процесса и потока. Структура PROCESS_INFORMATION определена так:

```
typedef struct PROCESS_INFORMATION (
    HANDLE hProcess;
    HANDLE hThread,
    DWORD dwProcessId;
    DWORD dwThreadld;
    ) PROCESS_INFORMATION;
```

Определение исполняемого образа и командной строки

Параметры lpszImageName и lpszCommandLine определяют имя исполняемого образа, руководствуясь приведенными ниже правилами.

- Если значение параметра lpszImageName не NULL, то это имя исполняемого файла. В случае если оно содержит пробелы, можно использовать кавычки.
- В противном случае имя исполняемого файла - первый элемент

параметра lpszCommandLine.

Обычно указывают только параметр lpszCommandLine, присваивая параметру lpszImageName значение NULL. Тем не менее существуют подробные правила для параметра lpszImageName.

- Если lpszImageName имеет значение не NULL, то он указывает имя загрузочного файла. Необходимо указывать полный путь и имя файла. Можно указать только имя файла, тогда будут использованы текущие диск и каталог без дополнительного поиска. В имя файла включают его расширение (.EXE, .BAT и т.п.).
- Если параметр lpszImageName имеет значение NULL, то за имя программы принимается первый элемент параметра lpszCommandLine, отделенный пробелом. Если это имя не содержит полного пути, происходит поиск в следующей последовательности:

1. Каталог образа текущего процесса.
2. Текущий каталог.
3. Системный каталог Windows, полученный посредством вызова функции GetSystemDirectory.
4. Каталог Windows, полученный посредством вызова функции GetWindowsDirectory.
5. Каталоги, указанные в переменной окружения PATH.

Новый процесс может получить командную строку, используя обычный механизм argv, или выполнить функцию GetCommandLine для получения всей командной строки в одной строковой переменной.

Отметим, что командная строка не является константой. Это следует из того факта, что параметры argv для главной программы тоже не константы. Программа может изменять свои параметры, поэтому желательно делать изменения в копии строки параметров.

2.Программа параллельной обработки файлов

Приведенная в этом примере программа создает процесс для поиска по образцу в файлах: по одному процессу на каждый файл. Этот способ можно применять в любой программе, использующей стандартный вывод

Программа выполняет следующую обработку:

- В каждом файле, от F1 до FN, проводится поиск с использованием отдельных процессов, запускающих один и тот же исполняемый файл. Программа создает командную строку в виде grep pattern FK (pattern – это шаблон для поиска в файлах). При запуске программы из командной строки он задается в виде текста в кавычках, например “ dot “.
- Дескриптор временного файла, определенный как наследуемый,

назначается полю hstdOut структуры запуска нового процесса.

- Используя функцию WaitForMultipleObjects, программа ожидает завершения всех процессов поиска.
- После того как поиск закончен, по порядку выводятся результаты (временные файлы). Во временный файл также выводятся результаты при выполнении команды cat.
- Функция WaitForMultipleObjects ограничена в числе дескрипторов величиной MAXIMUM_WAIT_OBJECTS , поэтому она вызывается несколько раз.
- Чтобы установить, найден ли образец определенным процессом, программа определяет код завершения процесса функцией GetExitCodeProcess.

Листинг программы параллельной обработки файлов

```
#include "stdafx.h"
#include "EvryThng.h"
#define BUF_SIZE 256
static VOID cat(LPTSTR, LPTSTR);
int _tmain (DWORD argc, LPTSTR argv [])
/*  Создает отдельный процесс для каждого файла из командной строки.
результатов предоставляется временный файл в текущем каталоге. */
{
HANDLE hTempFile;
BOOL prov;
TCHAR outFile[MAX_PATH + 100];
SECURITY_ATTRIBUTES StdOutSA =
/* права доступа для наследуемого дескриптора. */
{sizeof
(SECURITY_ATTRIBUTES), NULL, TRUE};
TCHAR CommandLine [MAX_PATH + 100];
STARTUPINFO StartUpSearch, StartUp;
PROCESS_INFORMATION ProcessInfo;
DWORD iProc, ExCode;
HANDLE *hProc;
/* Указатель на массив дескрипторов процессов. */
typedef  struct
{TCHAR TempFile [MAX_PATH];}
PROCFILE;
PROCFILE  *ProcFile;
/* Указатель на массив имен временных файлов. */
```

```
/* Информация запуска для каждого дочернего процесса поиска и для
процесса, который будет выводить результаты. */
GetStartupInfo (&StartUpSearch);
GetStartupInfo (&StartUp);
/* Зарезервировать место для массива структур данных процессов,
содержащих дескриптор процесса и имя временного файла. */
ProcFile = (PROCFILE *)malloc ((argc - 2) *sizeof (PROCFILE));
hProc = (HANDLE *)malloc ((argc-2) *sizeof (HANDLE));
/* Создать отдельный процесс "grep" для каждого файла из командной
строки*/
for (iProc = 0; iProc < argc - 2; iProc++)
{
/* Создать командную строку вида grep argv [1] argv [iProc + 2] */
_stprintf (CommandLine, _T ("%s%s%s"),
_T ("grep "), argv [1] , argv [iProc + 2]);
_tprintf(_T("%s\n"),CommandLine);
if (GetTempFileName (_T ("."), _T ("gtm"), 0,
ProcFile [iProc].TempFile) == 0)
/* Указываем стандартный вывод для процесса поиска. */
hTempFile = /* Этот дескриптор наследуемый */
CreateFile (ProcFile [iProc].TempFile, GENERIC_WRITE,
FILE_SHARE_READ | FILE_SHARE_WRITE, &StdOutSA,
CREATE_ALWAYS,
FILE_ATTRIBUTE_NORMAL, NULL);
StartUpSearch.dwFlags = STARTF_USESTDHANDLES;
StartUpSearch.hStdOutput = hTempFile;
StartUpSearch.hStdError = hTempFile;
StartUpSearch.hStdInput = GetStdHandle (STD_INPUT_HANDLE) ;
 /* Создаем процесс для выполнения командной строки. */
CreateProcess (NULL, CommandLine, NULL, NULL,
TRUE, 0, NULL, NULL, &StartUpSearch, &ProcessInfo);
/* Закрываем ненужные дескрипторы */
CloseHandle (hTempFile) ;
CloseHandle (ProcessInfo.hThread) ;
/* Сохраняем дескриптор процесса. */
hProc [iProc] = ProcessInfo.hProcess;
}
/* Все процессы выполняются, подождем их завершения. */
for (iProc = 0; iProc < argc-2; iProc += MAXIMUM_WAIT_OBJECTS)
WaitForMultipleObjects (min (MAXIMUM_WAIT_OBJECTS, argc - 2
-iProc),
```

```
&hProc[iProc] , TRUE, INFINITE);
/* Результирующие файлы отсылаются на стандартный вывод с
помощью "cat". */
for (iProc = 0; iProc < argc - 2; iProc++)
{
      printf("Proc= %d\n", iProc);
      prov=GetExitCodeProcess (hProc [iProc], &ExCode);
      if(ExCode != 0) DeleteFile (ProcFile [iProc].TempFile);
if (GetExitCodeProcess (hProc [iProc], &ExCode) && ExCode == 0)
{
/* Образец найден - показать результаты. */
if (argc > 3) _tprintf (_T ("%s : \n" ), argv [iProc+2]);
fflush (stdout) ;
/* требуется для использования стандартного вывода
несколькими процессами */
 _stprintf (outFile, _T("%s"),ProcFile[iProc].TempFile);

cat( argv[iProc+2], (LPTSTR) outFile);
 _stprintf (CommandLine, _T("%s%s"),
_T ("cat "), ProcFile[iProc].TempFile);
_tprintf(_T("%s\n"),CommandLine);
CreateProcess (NULL, CommandLine, NULL, NULL,
TRUE, 0, NULL, NULL, &StartUp, &ProcessInfo);
WaitForSingleObject (ProcessInfo.hProcess, INFINITE);
CloseHandle (ProcessInfo.hProcess);
CloseHandle (ProcessInfo.hThread);
}
CloseHandle (hProc [iProc] );
/*DeleteFile (ProcFile [iProc].TempFile); */
}
free (ProcFile);
free (hProc) ;
return 0;
}
static VOID cat(LPTSTR hInFile, LPTSTR hOutFile)
{
CopyFile( hInFile,  hOutFile, FALSE);
return;
}
```

3.Содержание работы

1.Изучить основные функции управления процессами: CreateProcess, OpenProcess, ExitProcess и их параметры.
2.Изучить функции завершения процессов WaitForSingleObject, WaitForMultipleObjects и возможные возвращаемые значения в случае успешного выполнения этих функций.
3.Набрать и отладить программу параллельной обработки файлов в среде Visual Studio C++ .NET 2003.
4.Выполнить программу, задавая исходные данные из командной строки. В командной строке указывается имя программы и через пробел имена файлов для параллельной обработки.
5.Разобраться с работой программы и объяснить ее функционирование.
6.Подготовить отчет по выполненной работе.
7.В отчете ответить на контрольные вопросы.

4.Порядок выполнения программ в среде Visual Studio C++ .NET 2003

1. Запустить Microsoft Visual Studio C++ .NET 2003.
2. Создать проект под именем lab6.
3. В файл lab6.cpp набрать текст программы.
4. Провести отладку программы, создать исполняемый файл.
5. Запустить программу из командной строки, задавая шаблон поиска и имена файлов, в которых следует найти шаблон.

5.Содержание отчета

В отчет должны быть включены следующие разделы:
1. Общие сведения о механизме управления процессами.
2. Основные функции управления процессами и их параметры.
3. Листинг программы параллельной обработки файлов.
4. Пояснения работы программы.
5. Письменные ответы на вопросы.

6.Контрольные вопросы

1.Какую фунцию выполняет CreateProcess?
2.Какую фунцию выполняет OpenProcess?
3.Какую фунцию выполняет ExitProcess ?
4.Какую фунцию выполняет WaitForSingleObject?
5.Как запускается программа параллельной обработки файлов?
6.Поясните работу программы параллельной обработки файлов.

Лабораторная работа №7

Расширенный ввод-вывод с процедурами завершения

Использованию объектов синхронизации есть альтернатива, которая заключается в том, что вместо ожидания потоком сигнала завершения от события или дескриптора, система может вызвать указанную пользователем процедуру завершения, когда операция ввода-вывода закончится. Эта процедура завершения может запустить следующую операцию ввода-вывода и выполнить любое другое действие.

Как программа может задать процедуру завершения?

В ReadFile или WriteFile не остается никаких параметров или структур данных, в которых можно было бы сохранить адрес процедуры. Однако существует семейство расширенных функций ввода-вывода, отличающихся суффиксом Ex, которые имеют дополнительный параметр для адреса процедуры завершения. Расширенные функции чтения и записи - ReadFileEx и WriteFileEx. Кроме того, следует использовать одну из пяти ожидающих функций предупреждения.

- WaitForSingleObjectEx
- WaitForMultipleObjectsEx
- SleepEx
- SignalObjectAndWait
- MsgWaitForMultipleObjectsEx

Расширенный ввод-вывод иногда называется вводом-выводом с предупреждением

Расширенные функции чтения и записи можно применять с дескрипторами открытых файлов, именованных каналов и почтовых ячеек, если при открытии (создании) был указан флаг FILE_FLAG_OVERLAPPED. Хотя ввод-вывод с перекрытием и расширенный ввод-вывод отличаются друг от друга, этот флаг устанавливает атрибут дескриптора для обоих типов асинхронного ввода-вывода.

1.Функции ReadFileEx и WriteFileEx

```
BOOL ReadFileEx (
    HANDLE  hFile,
    LPVOID lpBuffer
    DWORD  nNumberOfBytesToRead,
    LPOVERLAPPED lpOverlapped,.
```

```
    LPOVERLAPPED_COMPLETION_ROUTINE  lpcr);

BOOL WriteFileEx  (
    HANDLE hFile,
    LPVOID lpBuffer,
    DWORD nNumberOfBytesToWrite,
    LPOVERLAPPED lpOverlapped,
    LPOVERLAPPED_COMPLETION_ROUTINE  lpcr);
```

Эти две функции содержат дополнительный параметр для указания процедуры завершения.

Структуры перекрытия можно задать, но указывать в них элемент hEvent нет необходимости, так как система его проигнорирует. В то же время этот элемент может принести пользу, если фиксировать в нем данные идентификации.

В отличие от ReadFile и WriteFile, расширенные функции не требуют параметров, определяющих количество передаваемых байтов. Эта информация передается в процедуру завершения, которая должна быть включена в программу.

Процедура завершения принимает параметры количества байтов, кода ошибки и структуры перекрытия. Последний параметр нужен для того, чтобы процедура могла определить, какая из нескольких незаконченных операций завершается. В этом случае надо учитывать такие же ограничения по повторному использованию или уничтожению структур, как и при вводе-выводе с перекрытием.

```
VOID WINAPI  FileIOCompletionRoutine (
    DWORD fdwError,
    DWORD cbTransferred,
    LPOVERLAPPED  lpo);
```

Как и в случае с CreateThread, где также указывалось имя функции, здесь FileIOCompletionRoutine- метка-заполнитель, а не фактическое имя функции.

fdwError может принимать лишь значения 0 (успех) и ERROR_HANDLE_EOF (при попытке чтения после конца файла). Структура перекрытия используется из завершенного вызова ReadFileEx или WriteFileEx.

Прежде чем система вызовет процедуру завершения, должно произойти следующее.

1. Операция ввода-вывода должна завершиться.
2. Вызывающий поток должен находиться в состоянии ожидания предупреждения, которое информирует систему о том, что она должна выполнить процедуру завершения из имеющейся очереди.

Чтобы перейти в состояние ожидания утверждения, поток должен сделать явный вызов одной из трех ожидающих функций предупреждения. Таким образом, поток гарантирует, что процедура завершения не будет вызвана преждевременно.

Когда эти два условия выполняются, запускаются процедуры завершения, которые были помещены в очередь как результат завершения ввода-вывода. Процедуры завершения выполняются в том же потоке, который вызвал первоначальный ввод-вывод и который находится в состоянии ожидания предупреждения. Поэтому поток должен переходить в это состояние только в том случае, если выполнение процедур завершения не приведет к нежелательным результатам

2.Ожидающие функции предупреждения

Существует пять ожидающих функций предупреждения, три из которых имеют непосредственное отношение к нашему изложению и описаны здесь. Каждая из них имеет флаг fAlertable, который должен быть установлен в TRUE. Эти функции являются расширением обычных функций Wait и Sleep.

```
DWORD  WaitForSingleObjectEx   (
      HANDLE  hObject,
      DWORD dwTimeOut,
      BOOL fAlertable);

DWORD                WaitforMultipleObjectEx                (
DWORD  cObjects,
      LPHANDLE  lphObjects,
      BOOL   fWaitAll,
      DWORD dwTimeOut,
      BOOL fAlertable);

DWORD SleepEx (
      DWORD dwTimeOut,
      BOOL fAlertable);
```

Тайм-ауты, как обычно, задаются в миллисекундах. Эти три функции возвращают управление, как только наступает любая из следующих ситуаций:

- дескриптор или дескрипторы вызывают удовлетворение одной из двух функций ожидания обычным образом;
- истекает тайм-аут;

• все процедуры завершения в очереди потока завершаются, и флаг fAlertable установлен.

Со структурами перекрытия ReadFileEx и WriteFileEx не связано никаких событий, так что дескрипторы в вызове функции ожидания не будут иметь прямого отношения к операциям ввода-вывода. Функция SleepEx, с другой стороны, не связана с объектом синхронизации и наиболее проста в использовании. В этой функции обычно указывается значение тайм-аута INFINITE, в результате чего она возвращает управление только по окончании всех процедур завершения в очереди.

3.Программа преобразование файла с использованием расширенного ввода-вывода

Программа lab7.exe обеспечивает преобразование файла ASCII в Unicode с использованием асинхронного ввода-вывода с помощью процедуры завершения. Многие переменные сделаны глобальными, для того чтобы они были доступны для процедур завершения. Запуск программы производится из командной строки в виде: lab7.exe filel file2.

```
#include "stdafx.h"
#include "EvryThng.h"
#define MAX_OVRLP 4
#define REC_SIZE 64
#define UREC_SIZE 2 * REC_SIZE

static VOID WINAPI ReadDone (DWORD, DWORD, LPOVERLAPPED);
static VOID WINAPI WriteDone (DWORD, DWORD, LPOVERLAPPED);
/* Первая структура перекрытия предназначена для чтения, а вторая для записи. Структуры и буфера выделяются для каждой незавершенной операции. */
OVERLAPPED OverLapIn [MAX_OVRLP],OverLapOut [MAX_OVRLP];
CHAR AsRec [MAX_OVRLP][REC_SIZE];
WCHAR UnRec [MAX_OVRLP] [REC_SIZE];
HANDLE hInputFile, hOutputFile;
LONGLONG nRecord, nDone;
LARGE_INTEGER FileSize;
LARGE_INTEGER CurPosIn, CurPosOut;
DWORD ic;

int _tmain (int argc, LPTSTR argv [])
{
hInputFile = CreateFile (argv [1], GENERIC_READ,
0, NULL, OPEN_EXISTING, FILE_FLAG_OVERLAPPED, NULL);
```

```
hOutputFile = CreateFile (argv [2], GENERIC_WRITE,
0, NULL, CREATE ALWAYS, FILE_FLAG_OVERLAPPED, NULL);
FileSize.LowPart = GetFileSize (hInputFile,
(LPDWORD)FileSize.HighPart);
Record = FileSize.QuadPart / REC_SIZE;
printf("nR=%d\n", (int)nRecord);
if ((FileSize.QuadPart % REC_SIZE) != 0) nRecord++;
CurPosIn.QuadPart =0;
printf("nR2=%d\n", (int)nRecord);
for (ic = 0; ic < MAX_OVRLP; ic++)
{
printf("ic=%d\n", (int)ic);
OverLapIn [ic].hEvent = (HANDLE) ic; /* Перезагрузка события. */
OverLapOut [ic].hEvent = (HANDLE) ic;
OverLapIn [ic].Offset = CurPosIn.LowPart;
OverLapIn [ic].OffsetHigh = CurPosIn.HighPart;
if (CurPosIn.QuadPart < FileSize.QuadPart)
ReadFileEx (hInputFile, AsRec [ic], REC_SIZE, &OverLapIn [ic],
ReadDone);
CurPosIn.QuadPart += (LONGLONG) REC SIZE;
}
 /* Все операции чтения выполняются. Вводим состояние ожидания
завершения и продолжаем, пока не будут обработаны все записи. */
nDone = 0;
while (nDone < 2 * nRecord)
SleepEx (ININITE, TRUE);
CloseHandle (InputFile);
CloseHandle (hOutputFile);
_tprintf (_T ("ASCII in Unicode is completed\n"));
return 0;
}
static VOID WINAPI ReadDone (DWORD Code, DWORD nBytes,
LPOVERLAPPED pOv)
{
/* Чтение завершено. Преобразуем данные и начинаем запись. */
DWORD i;
Done++;
printf("nD1=%d\n", (int)nDone);
/* Обработка записи и начало операции записи. */
ic = (DWORD) (pOv->hEvent);
CurPosIn.LowPart = OverLapIn [ic].Offset;
```

```
CurPosIn.HighPart = OverLapIn [ic].OffsetHigh;
CurPosOut.QuadPart = (CurPosIn.QuadPart / REC_SIZE) * UREC_SIZE;
OverLapOut [ic].Offset = CurPosOut.LowPart;
OverLapOut [ic].OffsetHigh = CurPosOut.HighPart;
/* Преобразование записи ASCII в Unicode. */
for (i = 0; i < nBytes; i++)
UnRec [ic] [i] = AsRec [ic] [i];
WriteFileEx (hOutputFile, UnRec [ic], nBytes*2,
 &OverLapOut [ ic ], WriteDone);
/* Подготовка структуры перекрытия к следующему чтению. */
CurPosIn.QuadPart += REC_SIZE * (LONGLONG) (MAX_OVRLP);
OverLapIn [ic].Offset = CurPosIn.LowPart;
OverLapIn [ic].OffsetHigh = CurPosIn.HighPart;
return;

static VOID WINAPI WriteDone (DWORD Code, DWORD nBytes,
LPOVERLAPPED pOv)
{
/* Запись завершена. Начинаем следующее чтение. */
/*LARGE_INTEGER CurPosIn;*/
/*DWORD ic;*/
nDone++;
printf("nD2=%d\n", (int)nDone);
ic = (DWORD) (pOv->hEvent);
CurPosIn.LowPart = OverLapIn [ic].Offset;
CurPosIn.HighPart = OverLapIn [ic].OffsetHigh;
if (CurPosIn.QuadPart < FileSize.QuadPart)
{
ReadFileEx (hInputFile, AsRec [ic], REC_SIZE, &OverLapIn [ic],
ReadDone);
}
return;
}
```

4.Содержание работы

1.Изучить расширенные функции ввода-вывода: ReadFileEx, WriteFileEx и их параметры.
2.Изучить ожидающие функции предупреждения WaitForSingleObjecEx, WaitForMultipleObjectEx, SleepEx и их параметры.
3.Отладить программу преобразования файлов из ASCII в Unicode с

использованием расширенного ввода-вывода среде Visual C++ .NET.
4.Выполнить программу, задавая исходные данные из командной строки.
5.Разобраться с работой программы и объяснить ее функционирование.
6.Пояснить выводимые результаты.
7.Подготовить отчет по выполненной работе.
8.В отчете ответить на контрольные вопросы.

5.Порядок выполнения программ в среде Visual Studio C++ .NET 2003.

1.Запустить Microsoft Visual Studio C++ .NET 2003/
2.Создать проект в консольном приложении под именем lab7.
3.В файл lab7.cpp набрать текст программы.
4. Провести отладку программы, создать исполняемый файл.
5. Запустить программу из командной строки, задавая имя исходного файла в кодировке ASCII и имя файла, в который помещается преобразованный файл.

6.Содержание отчета

В отчет должны быть включены следующие разделы:
1.Общие сведения о расширенном вводе-выводе с использованием процедур завершения.
2.Основные функции расширенного ввода-вывода и их параметры.
3.Процедуры завершения расширенного ввода-вывода.
4.Листинг программы преобразования файла в кодировке ASCII в Unicode.
5.Результаты работы программы.
6.Письменные ответы на вопросы.

7.Контрольные вопросы

1.Какую фунцию выполняетReadFileEx?
2.Какую фунцию выполняетWriteFileEx?
3.Какую фунцию выполняет SleepEx?
4.Какую фунцию выполняет WaitForSingleObjectEx?
5.Поясните работу процедуры WINAPI ReadDone?
6.Поясните работу процедуры WINAPI WriteDone?
7.Поясните работу основной программы.

Лабораторная работа №8

Рисование графических фигур на экране монитора

1.Библиотека классов Microsoft Foundation Classes (MFC)

Библиотека классов Microsoft Foundation Classes (MFC) содержит набор средств объектно-ориентированного программирования, предназначенных для разработки 32-разрядных приложений. MFC - это мощный инструментарий объектно-ориентированного программирования.

Библиотека MFC дает возможность использования различных классов объектов. Наиболее важные структуры данных и API-функции инкапсулированы в группы классов, пригодных для многократного использования.

По сравнению с традиционными библиотеками функций, которые использовались в программах на языке С, библиотека MFC имеет много достоинств.

Основные преимущества использования классов С++ следующие:

- устраняются конфликты, возникшие из-за совпадения имен функций и переменных;
- код и данные инкапсулируются в классах;
- осуществляется наследование функциональных возможностей;
- размер кода значительно сокращается за счет хорошо организованной библиотеки классов;
- создаваемые программистом классы являются естественным расширением языка.

Благодаря MFC код, требуемый для отображения окна приложения, можно сократить примерно в три раза. При этом появляется возможность уделять больше времени и внимания разработке процедур взаимодействия приложения с Windows, для решения которых и создается приложение.

В основу MFC положены следующие принципы:

- возможность комбинирования обычных вызовов функций с методами классов;
- баланс между производительностью и эффективностью базовых классов;
- преемственность в переходе от использования API-функций к библиотеке классов;
- простоту переноса библиотеки классов на разные платформы ОС.

- наиболее полное использование возможностей C++ без чрезмерного усложнения программного кода.

Простота базовых классов делает их весьма несложными в использовании, а по скорости работы их методы практически не уступают библиотечным функциям языка C.

При создании MFC также учтены возможности работы в смешанном режиме. Другими словами, в одной программе могут применяться как библиотека MFC, так и API-функции.

Еще одно важное свойство, MFC заключается в том, что имеется возможность непосредственного применения базовых классов в программах.

Основные достоинства библиотеки MFC:

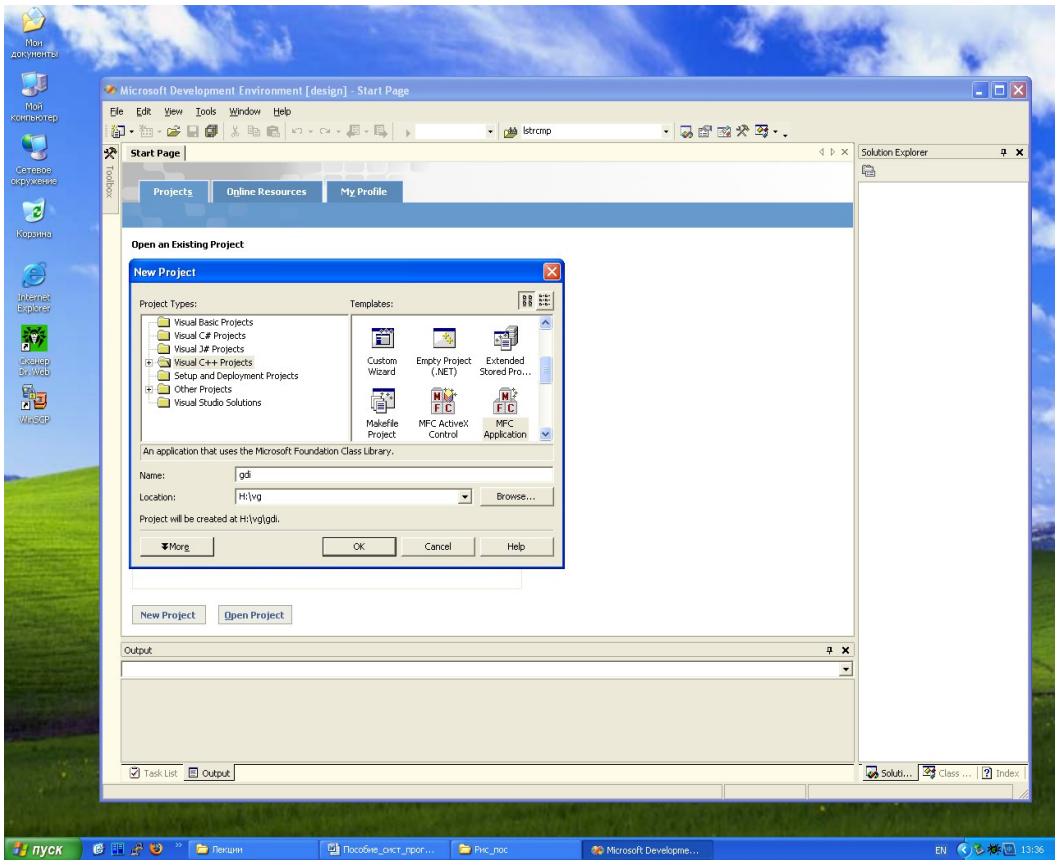
- Расширенная система обработки исключительных ситуаций, благодаря которой приложения менее чувствительны к ошибкам и сбоям. Такие ошибки, как «нехватка памяти» обрабатываются автоматически.
- Улучшенная система диагностики, позволяющая записывать в файл информацию об используемых объектах. Сюда также следует отнести возможность контроля корректности значений переменных-членов.
- Полная поддержка всех API-функций, элементов управления, сообщений, GDI, графических примитивов, меню и диалоговых окон.
- Возможность определить тип объекта во время выполнения программы. Это позволяет осуществлять динамическое манипулирование свойствами экземпляров классов.
- Отпала необходимость в использовании многочисленных громоздких структур switch/case, которые часто являются источниками ошибок. Все сообщения связываются с их обработчиками внутри классов. Этот способ привязки сообщений к обработчикам событий применим ко всем сообщениям.
- Небольшой размер и быстрота выполнения программного кода. Как уже говорилось выше, по этим показателям MFC не уступает стандартным библиотечным функциям языка C.
- Поддержка COM (Component Object Model - модель компонентных объектов).
- Использование тех же соглашений об именовании методов классов, которые применялись при подборе имен для API-функций Windows. Это существенно облегчает идентификацию действий, выполняемых классами.

Библиотека MFC поставляется вместе с исходными текстами классов, что дает возможность настраивать базовые классы в соответствии со своими потребностями.

2.Создание проекта программы рисования фигур

Используя мастер Application Wizard, создадим проект с именем GDI. Для этого необходимо:

- В меню File открыть вкладку New, Project, выбрать иконку MFC Application, в окне Name: набрать имя gdi, в окне Location: указать путь к папке, в которую будет занисан проект, нажать



ОК.

Рис.7. Создание проекта Gdi MFC Application

- На вкладке MFC Application Wizard – выбрать Application Type и установить Single documents (поддержка единичного документа), MFC standard, Use MFC in a static library, Document/View architecture support, Resource language – Английский (США).

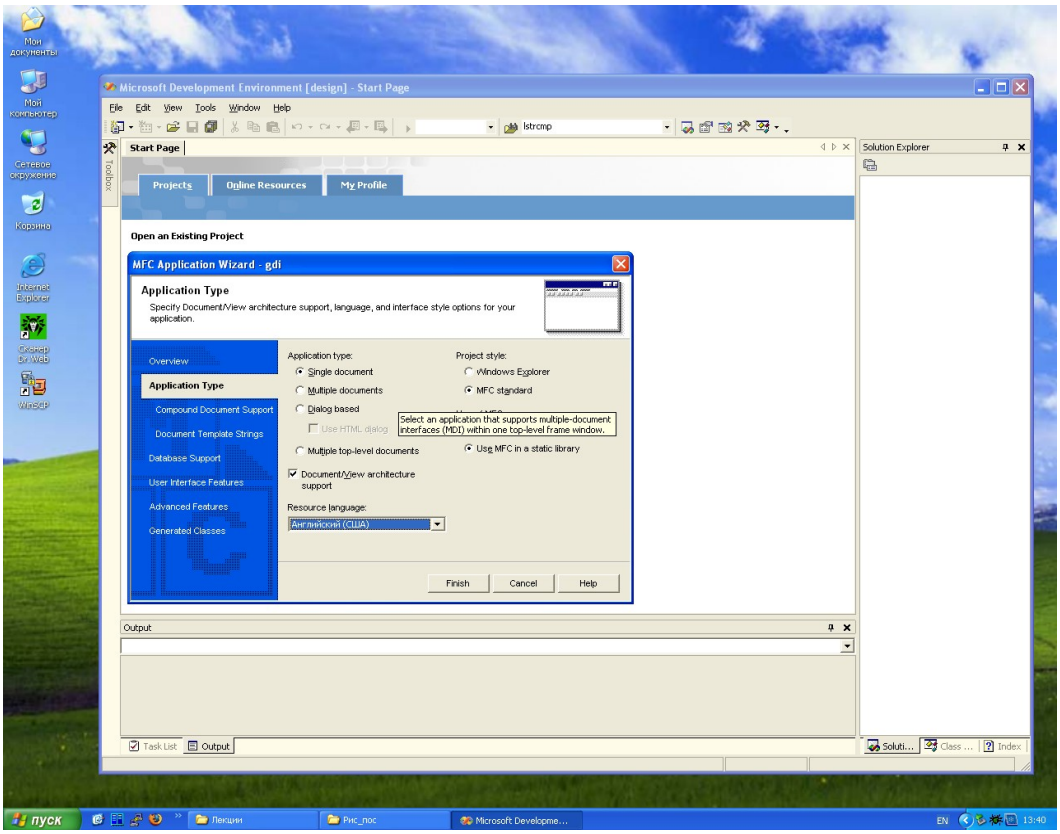


Рис.8.Установка группы опций Application Type для проекта Gdi

- На вкладке MFC Application Wizard – выбрать Compound Document Support, установить None.
- На вкладке MFC Application Wizard - Document Template Strings должны стоять стандартные установки .
- На вкладке MFC Application Wizard – выбрать Database support и установить None.
- На вкладке MFC Application Wizard – выбрать User Interface Features и установить Thick frame, Minimize box, Maximize box, System menu, Toolbars - None.
- На вкладке MFC Application Wizard – выбрать Advance Features, снять все установки, Number of files on recent file list – 4.
- На вкладке MFC Application Wizard – Generated Classes должен появится перечень из 4 классов проекта: CgdiView, CgdiApp, CgdiDoc, CMainFrame.
- В итоге получим папку с файлами проекта Gdi. Если папка с файлами проекта не появилась на экране, необходимо выбрать в меню View, Solution Explorer. Открыв папку Source Files, можно убедиться в наличии файлов проекта.

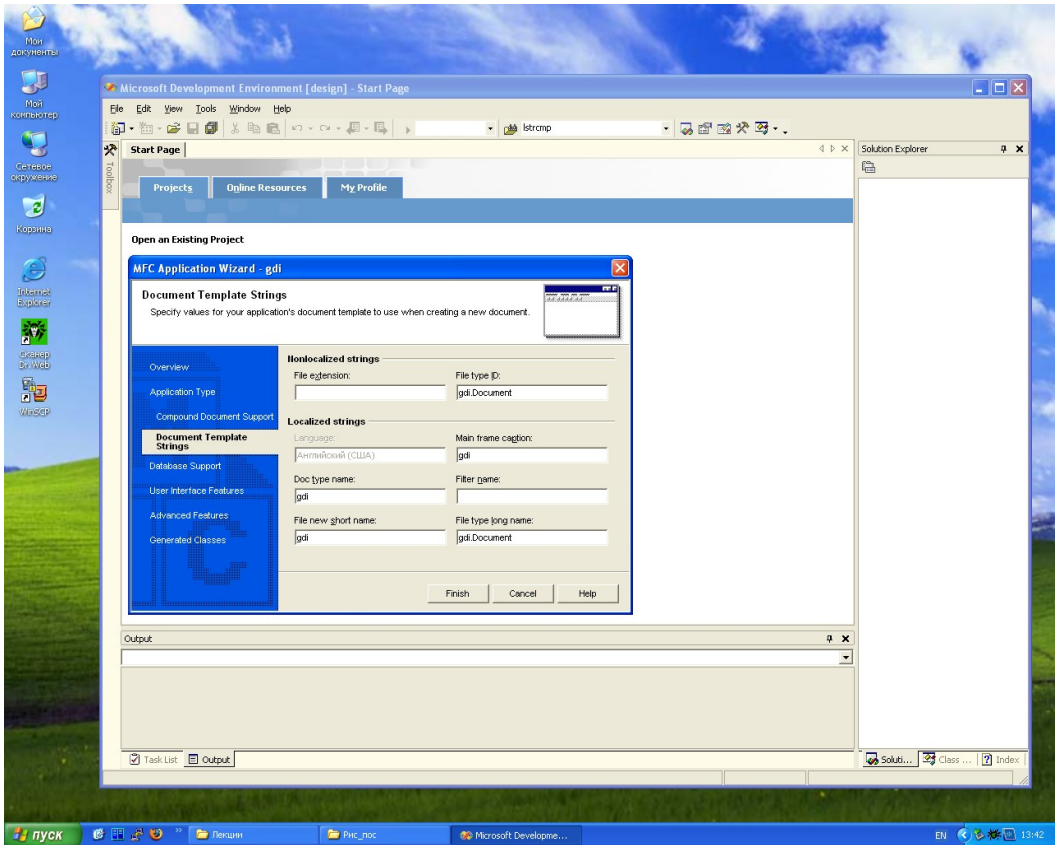


Рис. 9. Установка группы опций Compound Document Support

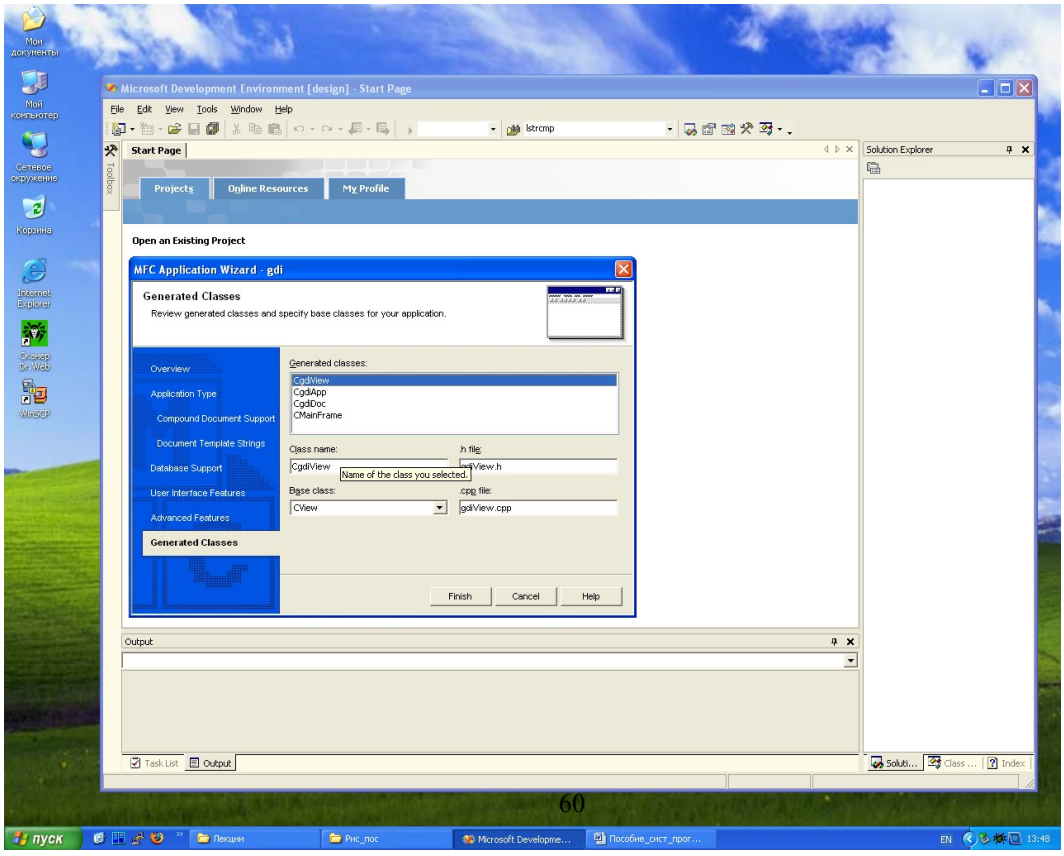


Рис.10. Классы, используемые в проекте Gdi

Для того, чтобы иметь возможность рисовать в рабочей области приложения графические фигуры, нужно предварительно модифицировать часть файла исходного кода gdiView.cpp. Для этого необходимо открыть этот файл, щелкнув по нему мышью, затем в этом файле найти фрагмент , озаглавленный как CgdiView drawing.

Файл GdiView.cpp, который содержит реализацию класса CGdiView, порождается от класса CGdiView и управляет отображением документа на экране монитора. В файл класса CGdiView добавлен фрагмент, выделенный полужирным шрифтом, который позволяет производит окраску графических фигур указанными цветами.

3.Программа рисования фигур

```
// GdiView.cpp : реализация класса CGdiView
#include "stdafx.h"
#include "Gdi. h"
#include "GdiDoc.h"
#include "GdiView.h"
#ifdef _DEBUG
#define new DEBUG_NEW
#endif

//CGdiViewdrawing
      void CgdiView :: OnDraw(CDC* pDC)
      {
        CGdiDoc* pDoc = GetDocument( );
        ASSERT_VALID(pDoc);
      static DWORD dwColor[9]={
        RGB(0,0,0),
        //черный
        RGB(255,0,0),                                    //красный
        RGB(0,255,0),                                    //зеленый
        RGB(0,0,255),                                    //синий
        RGB(255,255,0),                                  //желтый
        RGB(255,0,255),                                  //пурпурный
```

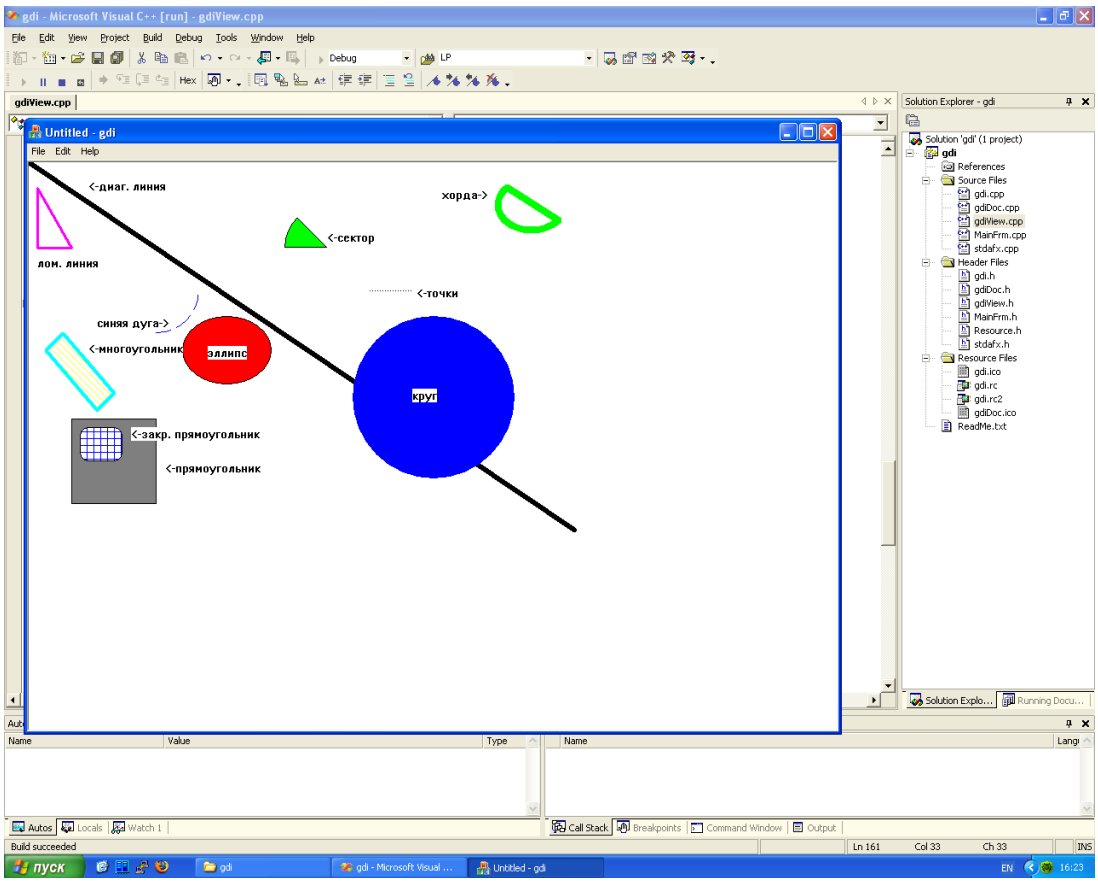
```
  RGB(0,255,255),                                   //голубой
  RGB(127,127,127),                                 //серый
  RGB(255,255,255)};                                //белый

int xcoord;
POINT polylpts[4],polygpts[5];
CBrush newbrush;
CBrush* oldbrush;
CPen newpen;
CPen* oldpen;
// рисование толстой диагональной линии
newpen.CreatePen(PS_SOLID,6,dwColor[0]);
oldpen=pDC->SelectObject(&newpen);
pDC->MoveTo(0,0);
pDC->LineTo(640,430);
pDC->TextOut(70,20,"<-diagonalline",15);
// удаление пера
pDC->SelectObject (oldpen);
newpen.DeleteObject();

// рисование синей дуги
newpen.CreatePen(PS_DASH,1,dwColor[3]);
oldpen=pDC->SelectObject(&newpen);
pDC->Arc(100,100,200,200,150,175,175,150);
pDC->TextOut(80,180,"small arc->",11);
// удаление пера
pDC->SelectObject(oldpen);
newpen.DeleteObject( );

// рисование сегмента с толстым зеленым контуром
newpen.CreatePen(PS_SOLID,8,dwColor[2]);
oldpen=pDC->SelectObject(&newpen);
pDC->Chord(550,20,630,80,555,25,625,70);
pDC->TextOut(485,30,"chord->",7);
// удаление пера
pDC->SelectObject(oldpen);
newpen.DeleteObject ( );

// рисование эллипса и заливка его красным цветом
CreatePen(PS_SOLID,1,dwColor[1]);
oldpen=pDC->SelectObject(&newpen);
```

```
newbrush.CreateSolidBrush(dwColor[1]);
oldbrush=pDC->SelectObject(&newbrush);
pDC->Ellipse(180,180,285,260);
pDC->TextOut(210,215, "ellipse", 7);
// удаление кисти
pDC->SelectObject(oldbrush);
newbrush.DeleteObject();
// удаление пера
pDC->SelectObject(oldpen);
newpen. DeleteObject (  );

// рисование  круга и заливка его синим цветом
newpen.CreatePen(PS_SOLID,1,dwColor[3]);
oldpen=pDC->SelectObject(&newpen);
newbrush.CreateSolidBrush (dwColor[3]);
oldbrush=pDC->SelectObject(&newbrush);
pDC->Ellipse(380,180,570,370);
pDC->TextOut(450,265,"circle",6);
// удаление кисти
pDC->SelectObject(oldbrush);
newbrush.DeleteObject() ;
// удаление пера
pDC->SelectObject(oldpen) ;
newpen.DeleteObject();

 // рисование сектора и заливка его зеленым цветом
newpen.CreatePen(PS_SOLID,1,dwColor[0]);
oldpen=pDC->SelectObject(&newpen);
newbrush.CreateSolidBrush(dwColor[2]);
oldbrush=pDC->SelectObject(&newbrush);
pDC->Pie(300,50,400,150,300,50,300,100);
pDC->TextOut(350,80,"<-pie wedge",11);
// удаление кисти
pDC->SelectObject(oldbrush) ;
newbrush.DeleteObject();
//удаление пера
pDC->SelectObject(oldpen) ;
newpen.DeleteObject();

// рисование прямоугольника и заливка его серым цветом
newbrush.CreateSolidBrush(dwColor[7]);
```

```
oldbrush=pDC->SelectObject(&newbrush);
pDC->Rectangle(50,300,150,400);
pDC->TextOut(160,350,"<-rectangle",11);
// удаление кисти
pDC->SelectObject(oldbrush);
newbrush.DeleteObject( );

// рисование закругленного прямоугольника
// и заливка его синим цветом
newbrush.CreateHatchBrush(HS_CROSS,dwColor[3]);
oldbrush=pDC->SelectObject(&newbrush);
pDC->RoundRect(60,310,110,350,20,20);
pDC->TextOut (120,310,"<-rounded rectangle" ,19);
// удаление кисти
pDC->SelectObject(oldbrush);
newbrush.DeleteObject( );

// рисование нескольких точеск
for(xcoord=400;xcoord<450;xcoord+=3)
pDC->SetPixel(xcoord,150,0L);
pDC->TextOut(455,145,"<-pixels",8);

// рисование толстой ломаной линии пурпурного цвета
newpen.CreatePen(PS_SOLID,3,dwColor[5]);
oldpen=pDC->SelectObject(&newpen);
polylpts[0].x=10;
polylpts[0].y=30;
polylpts[1].x=10;
polylpts[1].y=100;
polylpts[2].x=50;
polylpts[2].y=100;
polylpts[3].x=10;
polylpts[3].y=30;
pDC->Polyline(polylpts,4);
pDC->TextOut(10,110,"polyline",8);
// удаление пера
pDC->SelectObject(oldpen);
newpen.DeleteObject( );

// рисование многоугольника с голубым контуром
// и заливка его диагональной желтой штриховкой
```

```
        newpen.CreatePen(PS_SOLID,4,dwColor[6]);
        oldpen=pDC->SelectObject(&newpen);
        newbrush.CreateHatchBrush(HS_FDIAGONAL,dwColor[4]);
        oldbrush=pDC->SelectObject(&newbrush);
        polygpts[0].x=40;
        polygpts[0].y=200;
        polygpts[1].x=100;
        polygpts[1].y=270;
        polygpts[2].x=80;
        polygpts[2].y=290;
        polygpts[3].x=20;
        polygpts[3].y=220;
        polygpts[4].x=40;
        polygpts[4].y=200;
        pDC->Polygon(polygpts,5);
        pDC->TextOut(70,210,"<-polygon",9);
        // удаление кисти
        pDC->SelectObject(oldbrush);
        newbrush.DeleteObject ();
        // удаление пера
        pDC-> SelectObject(oldpen);
        newpen.DeleteObject( );
}
```

После ввода кода рисования фигур необходимо выполнить компиляцию программы, для этого в меню Build следует выбрать Build Solution. В результате компиляции в папке Debug проекта будет создан файл Cview.cpp, выполнение которого обеспечивает рисование указанных в файле фигур.



4.Пояснения к программе рисования фигур

Рисование линий и геометрических фигур с заданным цветом выполняется программируемыми «перьями», а раскраска фигур осуществляется «кистями», которые реализуются в классах CBrush и CPen.

Структура dwColor создается массив, в котором хранятся девять RGB-значений цветов для используемых кистей и перьев:

```
static DWORD dwColor[9]=
{
RGB(0,0,0),              //черный
RGB(255,0,0),            //красный
RGB(0,255,0),            //зеленый
RGBCO,0,255),            //синий
RGB(255,255,0),          //желтый
```

```
RGB(255,0,255),          //пурпурный
RGB(0,255,255),          //голубой
RGB(127,127,127),        //серый
RGB(255,255,255)         //белый
};
```

Кисти и перья создаются с помощью классов CBrush и CPen, которые можно передавать функциям класса CDC - базового класса для контекста устройства отображения. Кисти могут иметь сплошную и штриховую заливку, а также заливку в виде растрового узора. Перья рисуют сплошной, штриховой или пунктирной линиями. Синтаксис программы, отвечающей за создание кистей перьев, приведен ниже:

```
CBrush newbrush;
CBrush* oldbrush;
CPen  newpen;
CPen* oldpen;
```

Поскольку для рисования различных графических примитивов применяются аналогичные алгоритмы, рассмотрим лишь два наиболее типичных фрагмента. В первом случае на экран выводится толстая черная диагональная линия.

```
// рисование толстой черной диагональной линии
newpen.CreatePen(PS_SOLID,6,dwColor[0]);
oldpen-pDC->SelectObject(&newpen);
pDC->MoveTo(0,0);
pDC->LineTo(640,430);
pDC->TextOut(70,20,"<-diagonal 1ine",15);
// удаление пера
pDC->SelectObject(oldpen);
newpen.DeleteObjectO;
```

Перо создается функцией CreatePen(), которая задает рисование черных сплошных линий толщиной в шесть логических единиц. Сразу после этого функция SelectObject() обновляет перо, связывает его с контекстом устройства отображения (экраном монотора) и возвращает указатель на предыдущий объект пера. Функции LineTo() формируют диагональную линию, которая рисуется выбранными пером. Наконец, функция TextOut() выводит рядом с нарисованной фигурой надпись.

Работа с кистями организована аналогичным образом. В следующем коде создается кисть с заливкой в виде горизонтальных и вертикальных штрихов (HS_CROSS) синего цвета:

```
// рисование черного закругленного прямоугольника
// и заливка его синим цветом
newbrush.CreateHatchBrush(HS_CROSS,dwColor[3]);
```

```
oldbrush=pDC->SelectObject(&newbrush);
pDC->RoundRect(60,310,110,350,20,20);
pDC->TextOut (120,310,"<-rounded rectangle",19);
// удаление кисти
pDC->SelectObject(oldbrush);
newbrush.DeleteObject C);
```

Функция RoundRect () рисует прямоугольник с закругленными краями и заданными координатами, после чего выводится надпись. Все остальные фигуры рисуются аналогичным образом.

5.Содержание работы

1.Изучить работу Мастера MFC приложений.
2.Создать проект MFC приложения для рисования фигур.
3.Вставить фрагмент рисования фигур в файл CGdiView.cpp после строки // GdiView drawing, отредактировать и выполнить программу
4.Подготовить отчет.
5.Ответить на контрольные вопросы.

6.Порядок выполнения программ в среде Visual Studio C++ .NET 2003

1. Запустить Microsoft Visual Studio C++ .NET 2003/
2. Создать MFC проект под именем Gdi.
3.Набрать текст программы рисования фигур в файл CGdiView.cpp.
4.Создать файл Gdi.exe. Для этого необходимо выделить файл CGdiView.cpp. В меню Build выбрать Build Solution. Исправить ошибки компиляции.
5.Запустить программу, результаты работы программы проанализировать на экране дисплея.

7.Содержание отчета

В отчет должны быть включены следующие разделы:
1.Общие сведения о библиотек MFC. Особенности MFC. Основные классы MFC.
2.Процесс построения проекта с использованием библиотеки MFC.
3.Описание программы рисования фигур.
4.Копия экрана с нарисованными фигурами.

8.Контрольные вопросы

1.Какие принципы положены в основу библиотеки MFC?
2.Особенности MFC библиотеки?

3.Основной класс библиотеки MFC?
4.Какие файлы составляют проект в MFC приложении?
5.Как формируются цвета в программе рисования фигур?
6. Какие основные инструменты используются при рисовании фигур?
7.Как в программе реализуются инструменты для рисования фигур и заливка их различными цветами.
8.Поясните работу программы рисования фигур.

Лабораторная работа №9

Программирование приложения для расчета и рисования графика ряда Фурье

1.Создание проекта Fourier

Используя мастер Application Wizard, создадим проект с именем **Fourier**. Для этого необходимо:

- В меню File открыть вкладку New, выбрать MFC Application, в окне набрать имя Fourier, нажать ОК.
- На вкладке MFC Application Wizard – выбрать Application Type и установить Single documents (поддержка единичного документа), MFC standard, Use MFC in shared DLL, Document/View architecture support,

Resource language – Английский (США).

- На вкладке MFC Application Wizard – выбрать Compound Document Support, установить None.
- На вкладке MFC Application Wizard - Document Template Strings должны стоять стандартные установки .
- На вкладке MFC Application Wizard – выбрать Database support и установить None.
- На вкладке MFC Application Wizard – выбрать User Interface Features и установить Thick frame, Minimize box, Maximize box, System menu, Toolbars - None.
- На вкладке MFC Application Wizard – выбрать Advance Features, снять все установки.
- На вкладке MFC Application Wizard – Generated Classes должен появится перечень из 4 классов проекта: CFourierView, CFourierApp, CMainFrame, CFourierDoc. Далее следует нажать Finish.

В итоге получим папку с файлами проекта Fourier. Если папка с файлами проекта не появилась на экране, необходимо выбрать в меню

View – Solution Explorer. Открыв папку Source Files, можно убедиться в наличии файлов проекта.

2.Добавление ресурса диалогового окна

Для создания в проекте ресурса диалогового окна необходимо воспользоваться панелью Resource View. Для этого в меню View необходимо выбрать Resource View. В правой части окна появится панель Resource View – Fourier. На этой панели необходимо открыть папку Dialog, правой кнопкой мыши щелкнуть по имеющемуся там ресурсу, выбрать Add Resource и на отрывшемся окне выбрать Insert Dialog. На экране появится редактор диалоговых окон с набором инструментов для редактирования, который размещается на экране слева. Созданное редактором диалоговое окно можно модифицировать, используя набор инструментов.

Для добавления окна ввода числа гармоник Фурье и текста заголовка графика следует использовать два элемента Edit Control. Для ввода пояснительного текста на окне диалога используются три элемента Static Text. Для установки окон ввода исходных данных или пояснительного текста необходимо левой кнопкой выбрать нужный элемент (для пояснительного текста Static Text, для ввода данных Edit Control), а затем, после появления знака +, установить размеры окон.

Пояснительный текст в окно вводится, если в меню View выбрать Properties Windows. С права на экране появляется окно Properties. В строке Caption для выбранного элемента ввести необходимый текст вместо Static. В элементы Static Text последовательно ввести – Fourier Series Data Entry, Figure Caption (Edit1), Number of harmonics (Edit2).

Для изменения свойств диалогового окна необходимо изменить название окна. Для этого необходимо на диалоговое окно на панели Properties строку Caption, ввести текст Data Entry и нажать Enter.

Осталось изменить свойства ID диалогового окна. С этой целью в окне Properties в строке ID ввести IDD_FourierDlg и нажать Enter.

3.Создание основного меню и изменение ресурса меню

Для вывода на экран меню проекта Fourier необходимо воспользоваться панелью Resource View, открыть папку Menu и левой кнопкой дважды щелкнуть по строке IDR_MAINFRAME. Тогда на экране появится меню проекта.

Модификация меню, которое в проекте Fourier предлагается по умолчанию, производится следующим образом. Для удаления

ненужного элемента меню (например, Edit) необходимо выделить элемент Edit, нажать клавишу Delete.

Для создания нового элемента в меню File необходимо щелкнуть правой кнопкой по пустому полю, выбрать Insert New. Затем в окне Properties выбрать Caption и набрать имя нового элемента Data Entry, который позволит открыть диалоговое окно Data Entry.

Область заголовка проекта Fourier также можно изменить, для чего на панели Resource-View нужно открыть элемент String Table и дважды щелкнуть левой кнопкой по ресурсу. Затем в строке Caption изменить строку, идентифицируемую значением IDR_MAINFRAME на Fourier\nFourier Series Application\nFourier\nFourier\n\n\\

4.**Создание программы проекта Fourier**

Ниже представлена программа, предназначенная для рисования графика ряда Фурье. Речь идет о программе метода OnDraw(), который находится в файле Fourier View.cpp. Изменения, внесенные в код шаблона, выделены полужирным шрифтом.

```
//CFourierView drawing
void CFourierView::OnDraw(CDC* pDC)
{
CFourierDoc* pDoc = GetDocument();
ASSERT_VALID(pDoc);
// код рисования графика ряда Фурье
int anq;
double y, yp;
pDC->SetMapMode(MM_ISOTROPIC);
pDC->SetWindowExt(500, 500);
pDC->SetViewportExt(m_cxCllent, - m_cyClient);
pDC->SetViewportOrg(m_cxClient / 20, m_cyClient / 2);
anq =0;
yp=0;
// вывод осей координат
pDC -> MoveTo(0, 240);
pDC -> LineTo(0, - 240);
pDC -> MoveTo(0, 0);
pDC -> LineTo(400, 0);
pDC -> MoveTo (0, 0);
// вывод надписи
pDC->TextOut(420, 10, pDoc -> m_text);
// вывод графика ряда Фурье на интервале одного периода
for (int i = 0: i <= 400; i++)
```

```
{
for (int j = 1; j <= pDoc -> m_terms; j++)
{
y = (250.0 / ((2.0 * j) - 1.0)) *
sin(((j * 2.0 ) - 1.0) *
    (anq * 2.0 * 3.14159265359 /400.0));
    yp += y;
}
pDC -> LineTo(i,  (int)yp);
yp-=yp;
 anq++;
}
}
```

С помощью этой программы создается свободно масштабируемая поверхность рисования, позволяющая избежать проблем с масштабированием.

Как видно из представленного выше программы, с помощью функции SetMapMode() устанавливается режим отображения MM_ISOTROPIC, использующий произвольную единицу рисования: pDC -> SetMapMode(MM_ISOTROPIC);

Следующая строка программы задает размеры окна равными по вертикали и горизонтали 500 единицам: pDC -> SetWindowExt(500, 500);

Это означает, что какими бы ни были действительные размеры окна, координатные оси всегда будут иметь одну и ту же длину: 500 единиц. В следующей строи размеры области просмотра задаются равными текущим размерам рабочей области окна:

pDC -> SetViewportExt(m_cxC1ient, -m_cyC1ient);

Такой прием является гарантией того, что все содержимое окна будет видимым,

В приведенной программе точка начала области просмотра устанавливается пересечении середины рабочей области по вертикали и одной ее двадцатой по горизонтали от левого края:

pDC -> SetViewportOrg(m_cxClient/20, m_cyClient/2):

Итак, оси координат X и У построены. Рисование осей координат выполняется с помощью следующего фрагмента.

```
// рисование осей координат X и Y
pDC -> MoveTo(0, 240);
pDC -> LineTo(0, -240);
pDC -> MoveTo(0, 0);
```

```
pDC -> LineTo(400, 0);
pDC -> MoveTo(0, 0);
```

В показанном ниже фрагменте программы, предназначенном для рисования ряда Фурье, используются два цикла for. В переменной i хранится значение угла, применяемое функцией синуса, а в переменной j - номер текущей гармоники Фурье. Каждая точка строится на экране как результат суммирования всех гармоник Фурье для заданного угла. Таким образом, если вам нужно, чтобы приложение нарисовало 1000 гармоник, необходимо будет произвести приблизительно 400000 отдельных вычислений.

```
// рисование ряда Фурье
for (int i= 0; i <=400; i++)
{
for (int j =1; j <= pDoc -> myterms; j++)
{
y = (250.0 / ((2.0 * j) - 1.0)) *   \
sin(((j * 2.0 ) - 1.0) * \
(ang * 2.0 * 3.14159265359 / 400.0));
      yp+=y ;
}
pDC -> LineTo(i,  (int)yp);
yp+=yp;
ang++;
```

Функция LineTo() используется для объединения всех вычисленных точек графика в одну сплошную линию, которая и рисуется на экране. Программа построения графика завершается строкой, которая выводит надпись справа от оси *X:*

```
// draw a label
pDC->TextOut(420,  10, pDoc -> mytext);
```

5.Программа для масштабирования графика

Фрагмент программы, обеспечивающей масштабирования графика уже приведен в предыдущем разделе. Он начинается с функции SetMapMode() и заканчивается функцией SetViewportOrg(). Функции SetViewportExt() и SetViewportOrg() используют переменные-члены m_cxClient и m_cyClient. В программе эти значения не определены и должны передаваться из диалогового окна. Это данные о размере рабочей области.

Чтобы получить эту информацию, нужно добавить в проект функцию-метод void OnSize( ) и параметры, необходимые для ее

работы. С этой целью нужно открыть панель Class View. Для этого в меню View необходимо выбрать Class View. Тогда в правой части экрана появится данное окно. Необходимо выбрать CFourierView, щелкнуть правой кнопкой и выбрать вкладку Add, затем Function. В поле Return type ввести void, в поле Function Name ввести OnSize. В поле Parameter name ввести последовательно три строки unsigned int nType, int cx, int cy, используя кнопку Add. В результате в окне Parameter list должны появиться эти три строки. Установить Access – protected. Нажать кнопку Finish. Данный метод добавляется в файл CFourierView.

Теперь нужно ввести две переменные метода. Для этого правой кнопкой мыши следует щелкнуть по имени класса CFourierView. На экране появится меню, в котором следует выбрать Add, а затем Add Variable. В открывшееся окно добавляется переменная m_cyClient, а затем m_cxClient, нажать кнопку Finish. В файле CFourierView.cpp должны появится строки

```
void CFourierView :: OnSize(unsigned int nType, int cx, int cy)
{
CView :: OnSize(nType, cx, cy);
m_cxClient=cx;
m_cyClient=cy;
}
```

Теперь необходимо обратиться к заголовочному файлу CFourierView.h и убедиться, что он содержит объявление метода и переменных.

```
DECLARE_MESSAGE_MAP()
void OnSize(unsigned int nType, int cx, int cy);
void OnFourier(void);
        int m_cyClient;
        int m_cxClient;
```

6.Связь диалогового окна с проектом

Ресурс диалогового окна дает возможность пользователю вводить в приложения определенные данные, в нашем случае это количество гармоник и заголовок графика. Для этого необходимо обеспечить связь диалогового окна с проектом. Обычно это достигается посредством добавления нового класса.

Необходимо найти окно Class View -Fourier, правой кнопкой мыши щелкнуть по строке Fourier, в появившемся окне выбрать Add, а затем

Add Class. В появившемся окне левой кнопкой мыши дважды щелкнуть по MFC Class. Появляется окно MFC Class – Fourier.

В окно Class Name следует ввести имя класса CFourierDlg, базовый класс – CDialog. Значение ID для данного класса должно быть IDD_FourierDlg, такое же какое назначено для диалогового окна, нажать кнопку Finish. В результате создается класс CFourieDlg. Поддержка диалогового окна обеспечивается файлами CFourierDlg.cpp и CFourierDlg.h.

Затем в созданный класс необходимо добавить переменные диалога. Для этого следует в окне Class View - Fourier, щелкнуть правой кнопкой по имени класса CFourierDlg, выбрать Add, затем Variable, ввести CString m_text, установить protected, нажать Finish. Затем щелкнуть правой кнопкой по имени класса CFourierDlg выбрать Add, затем Variab, ввести int m_terms, установить protected, нажать кнопку Finish.

Теперь следует добавить функцию – член OnFourier в класс CFourierView, который содержится в файле FourierView.cpp. Для этого следует в окне Class View-Fourier правой кнопкой мыши щелкнуть по имени файла CFourierView и выбрать вкладку Add, затем Add Function. В появившемся окне в поле Return type ввести void, в поле Function name ввести OnFourier, установить Access – protected, нажать Finish.

После того как это сделано, убедитесь, что в файлы FourierDlg.cpp и FourierDlg.h внесены все необходимые изменения.

Файлы FourierDlg.cpp и FourierDlg.h

Представленный ниже фрагмент программы взят из файла исходного кода FourierDlg.cpp. Полужирным шрифтом в нем выделены строки, которые следует добавить в текст вашего модуля, если их там еще нет:

```
// FourierDlg.cpp : Implementation file
#include "stdafx.h"
#include "Fourier,h"
#include "FourierDlg.h"
IMPLEMENT_DYNAMIC(CFourierDlg, CDialog)
FourierDlg::CFourierDlg(CWnd* pParent /*=NULL*/)
 : CDialog(CFourierDlg::IDD, pParent)
{
m_terms = 4;        // присвоение значений полей по умолчанию
m_text = _T("Title");   // надпись оси в диалоговом окне
void CFourierDlg::DoDataExchange(CDataExchange* pDX)
{
CDialog: :DoDataExchange(pDX);
```

DDX_Text(pDX, IDC_EDIT2, m_terms); // организация обмена данными
DDX_Text(pDX, IDC_EDIT1, m_text);
DDV_MinMaxUInt(pDX, m_terms, 1, 100000);
}

Обратите внимание на тот факт, что, как видно из первых выделенных строк, в 1 полях ввода по умолчанию устанавливаются значения 4 и "Title".

Следующий фрагмент кода, который нам предстоит редактировать, взят из заголовочного файла FourierDlg.h. Строки, которые следует добавить в текст нашего модуля, если их там еще нет, выделяются, как и прежде, полужирным шрифтом:

```
// CFourierDlg dialog
class CFourierDlg : public CDialog
{
DECLARE_DYNAMIC(CFourierDlg)

// Dialog Data
enum { IDD = IDD_FOURIERDLG };

DECLARE_MESSAGE_MAP( )
public:
    int m_terms;
  CString m_text;
};
```

Смысл задействованных в этом коде переменных и значений очевиден, если исходить из того обстоятельства, что они используются в интерфейсе диалогового окна.

В библиотеке MFC существует класс CDialog, предназначенный для разработки регулярных и модальных диалоговых окон. Для создания простейших диалоговых окон, в частности окна About, можно использовать готовый класс из библиотеки MFC. Однако для более сложных диалоговых окон, в которых производится ввод данных, потребуется создать собственный класс, порождаемый от уже существующего. Диалоговое окно в представленном выше примере порождено от класса CDialog. Модальное диалоговое окно необходимо закрыть до того, как введенные в нем данные будут переданы приложению.

Функциональные возможности модального диалогового окна, порождаемого от существующего класса, можно расширить посредством добавления новых членов класса, а именно переменных и

методов. Члены-переменные можно также использовать для хранения данных, вводимых пользователем, а также данных, предназначенных для вывода на экран. Классы диалоговых окон, порождаемые от класса CDialog, могут иметь собственную схему сообщений. Однако если переопределяются только функции OnInitDialog(), OnOK() и OnCancel(, таковую можно не создавать.

В нашем простом примере класс CFourierDlg() порожден от класса CDialog. Владельцем этого модального диалогового окна является родительское окно.

Диалоговое окно передает введенные данные в приложение, когда пользователь щелкает на кнопке ОК. После щелчка на кнопке ОК или Cancel диалоговое окно закрывается и удаляется с экрана. При этом члены-функции обращаются к своим членам-переменным для получения информации, введенной пользователем.

Файлы FourierDoc.cpp и FourierDoc.h

В MFC-проекте поддержка документа обеспечивается специальными файлами. В нашем проекте это файл FourierDoc.cpp. Обычно в таких файлах хранятся строчные данные, с которыми могут работать различные ресурсы. Мы используем файл FourierDoc. cpp для передачи информации из диалогового окна в сам проект.

Представленный ниже отредактированный фрагмент кода был взят из файла исходного кода FourierDoc.cpp. Полужирным шрифтом в нем выделены строки, добавленные вручную:

```
// FourierDoc.cpp : implementation of the CFourierDoc class
//
#include "stdafx.h"
#include "Fourier.h"
#include "FourierDoc.h"

// CFourierDoc construction/destruction
CFourierDoc : : CFourierDoc( )
 {
m_terms =4;
m_text = "Title";
}
```

Число 4 и слово "Title" - это значения по умолчанию, используемые в созданном вами диалоговом окне ввода данных.

Следующий фрагмент кода был взят из заголовочного файла CFourierDoc.h и также отредактирован. Здесь полужирным шрифтом выделены строки, добавленные нами вручную:

```
// FourierDoc.h : interface of the CFourierDoc class
```

```
#pragma once
class CFourierDoc : public Cdocument
{
protected: // create from serialization only
CFourierDoc( );
DECLARE_DYNCREATE(CFourierDoc)
int m_terms;
CString m_text;
```

Далее необходимо выяснить, какие изменения были внесены в файл исходного кода FourierView.cpp и заголовочный файл FourierView.h.

Файлы FourierView.cpp и FourierView.h

Представленный ниже отредактированный фрагмент программы взят из файла исходного кода CFourierView.cpp. В нем полужирным шрифтом выделены строки добавленные нами вручную. Обращаем внимание на то, что для краткости изложения часть кода в листинге опущена. Если выделенные проекты отсутствуют, то их следует набрать вручную.

```
// FourierView.cpp : implementation of the CFourierView class
#include "stdafx.h"
#include "Fourier. h"
#include "Fouri erDoc. h"
#include "FourierView.h"
#include " FourierDlg.h"
        CFourierDlg dlg;
#include "math.h"
……….
IMPLEMENT_DYNCREATE(CFourierView, CView)
BEGIN_MESSAGE_MAP(CFourierView, Cview)
ON_WM_SIZE( )
ON_COMMAND(IDD_FourierDlg, OnFourier)
END_MESSAGE_MAP ( )

//CFourierView message handlers
void CFourierView::OnSize(unsigned int nType, int ex, int cy)
{
CView::OnSize(nType, ex, cy);
m_cxClient =cx;
m_cyClient =cy;

void CFourierView: :OnFourier(void)
```

```
{
CFourierDlg dig (this);
int result = dlg.DoModal( );
if (result = IDOK) {
CFourierDoc* pDoc = GetDocument( );
ASSERT_VALID(pDoc);
pDoc -> m_terms = dlg.m_terms;
pDoc -> m_text   = dlg.m_text;
Invalidate( );
}
}
```

Пользователь может ввести название графика и целое число, обозначающее требуемое количество гармоник Фурье. После щелчка на кнопке ОК диалоговое окно будет закрыто, а рабочая область обновится, как это следует из представленного далее фрагмента кода. Заметим, что в этот момент данные, введенные пользователем, передаются приложению.

Как указывалось ранее, класс CFourierDlg порожден от класса CDialog. Данные передаются приложению, когда пользователь щелкнет на кнопке ОК после их ввода в диалоговом окне.

Из диалогового окна информация поступает в код проекта и с помощью членов-переменных передается в локальные переменные m_terms и m_text.

```
pDoc -> m_terms = dlg.m_terms;
pDoc -> m_text   = dlg.m_text;
```

Представленный ниже отредактированный фрагмент кода взят из заголовочного файла FourierView.h. В нем полужирным шрифтом выделены строки, добавленные нами вручную:

```
// FourierView.h : interface of the CFourierView class
// Generated message map functions protected:
DECLARE_MESSAGE_MAP( )
void OnSize(unsigned int nType, int cx, int cy);
void OnFourier(void);
int m_cxClient;
int m_cyClient;
};
```

Макрос DECLARE_MESSAGE_MAP( ) часто используется для констатации того факта, что в классе переопределяется карта обработки сообщений. Этот метод более эффективен, чем использование виртуальных функций.

В файле FourierView.cpp имеется макрос ON_COMMAND(), который в данном случае применяется для вызова метода OnFourier():

On_COMMAND(IDD_FOURIERDLG, OnFourier).

Эта строка связывает пункт меню File проекта с диалоговым окном.

7.Содержание работы

1. Изучить работу Мастера MFC приложений.
2. Создать проект MFC приложения для расчета ряда Фурье с заданным количеством гармоник.
3. Создать и отредактировать меню диалога для ввода исходных данных программы.
4. Создать меню проекта отредактировать для проведения расчетов.
5. Связать диалоговое окно с проектом для проведения расчетов.
6. Вставить фрагмент программы расчета ряда Фурье в файл CFourierView.cpp, отредактировать и выполнить программу расчета ряда Фурье с различным числом гармоник.
7. Подготовить отчет.
8. Ответить на контрольные вопросы.

8.Порядок выполнения программ в среде Visual Studio C++ .NET 2003

1. Запустить Microsoft Visual Studio C++ .NET 2003/
2. Создать MFC проект под именем Fourier.
3. Из методических указаний к лабораторной работе №8 скопировать текст программы расчета ряда Фурье с заданным числом гармоник в файл CFourierView.cpp.
4. Затем, следуя описанию в методичке, создать диалоговое окно для работы с программой.
5. Создать меню проекта и провести редактирование.
6. Связать меню проекта с диалоговым окном.
7. Проверить наличие в файлах CFourierView.cpp, CFourierView.h, CFourierDlg.cpp, CFourierDlg.h, CFourierDoc.cpp, CFourierDoc.h элементов, которые в указанных файлах выделены полужирным текстом.
8. Создать файл Fourier.exe. Для этого необходимо выделить файл CFourierView.cpp. В меню Build выбрать Build Fourier.exe. Исправить ошибки компиляции.
9. Запустить программу, ввести исходные данные в диалоговом окне и выполнить 3 варианта расчетом с различными исходными данными.

9.Содержание отчета

В отчет должны быть включены следующие разделы:
1. Процесс построения приложений с использованием библиотеки MFC.
2. Создание шаблонного проекта Fourier.
3. Создание ресурса диалогового окна.
4. Изменение ресурса меню проекта.
5. Описание программы расчета ряда Фурье с заданным числом гармоник.
6. Результаты расчета ряда Фурье, графики.

10.Контрольные вопросы

1.Какие файлы составляют проект в MFC приложении?
2.Как создается диалоговое окно в MFC приложении и выполняется его редактирование?
3.Как выдать на экран меню проекта в стандартном исполнении и провести его редактирование?
4.Поясните работу программы расчета ряда Фурье.
5.Поясните программу выдачи результатов расчета на график.

Литература

1. Харт, Джонсон,М. Системное программирование в среде Win32, 2-е изд. : Пер. с англ.: -М. : Издательский дом «Вильямс», 2001. -464 с.
2. К. Паппас, У.Мюррей. Эффективная работа: Visual C++ .NET.: СПб.: Питер, 2002.- 816 с.

ОГЛАВЛЕНИЕ

Гальченко Валерий Григорьевич
Лабораторный практикум
Системное программирование в среде Win32
Программирование Windows приложений

Научный редактор доцент, к.ф.-м.н Г.Е.Шевелев
Редактор
Технический редактор
Подписано к печати
Формат 60x84/16. Бумага ксероксная.
Плоская печать. Усл.печ.л. Уч.-изд.л.
Тираж экз. Заказ Цена свободная.
ИПФ ТПУ. Лицензия ЛТ №1 от 18.07.94

Типография ТПУ. 634034, Томск, пр. Ленина, 30