Государственный университет телекоммуникаций им. Бонч-Бруевича
кафедра «Цифровой обработки сигналов»

И. И. Гук,
доцент кафедры ЦОС
gook_igor@mail.ru

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

к лабораторной работе № 2

«Программная реализация файловой модели на языке С»

2008 год

Оглавление

1. Создание пустого проекта....3
   1.1. Создать папку для хранения файлов проекта....3
   1.2. Запустить программу wvDev-C++....3
   1.3. Создать проект....4
   1.4. Откомпилировать созданный проект....6
2. Разработка алгоритма для простейшего выражения....8
3. Написание программного кода простейшего алгоритма....9
   3.1. Создать заголовочный файл разрабатываемого проекта....9
   3.2. Модифицировать функцию main()....11
   3.3. Модифицировать заголовочный файл prjMPY.h....12
   3.4. Реализовать программный код инициализации начального состояния алгоритма....12
   3.5. Создание указателя pCnt....14
   3.6. Определение оригинальных имен стандартных типов....15
   3.7. Создание указателей на входной/выходной файлы....15
   3.8. Связывание файловых указателей с реальными файлами, где хранятся входные данные и результат вычислений....16
   3.9. Создание входного/выходного буферов....19
   3.10. Создание указателей на входной/выходной буферы в контекстной структуре....21
   3.11. Реализовать программный код считывания данных из файла во входной буфер....22
   3.12. Реализовать программный код записи данных из выходного буфера в файл....23
   3.13. Реализовать программный код функции обработки считанных данных и записать результат в выходной буфер....24
   3.14. Написать код функции runMPY()....26
   3.15. Отладить код функции код функции....28
   3.16. Закрытие и повторный запуск проекта....31
4. Приложения....34
   Листинг файла prjMPY.h....34
   Листинг файла constantrj.cpp....34
   Листинг функции initMPY()....35
   Листинг функции runMPY()....35
   Листинг функции main()....37

1. Создание пустого проекта

1.1. Создать папку для хранения файлов проекта.

Это можно сделать в любом файловом менеджере. Например, в проводнике *Windows*, или в *TotalComander*. На рисунке внизу показан пример создания папки в *TotalComander*. Необходимо перейти в ту директорию, где планируется создать папку проекта. Рекомендуется, что бы это была директория на *Flash* накопителе студента. Затем нажать клавишу *F*7. В появившемся окне набрать имя папки. Рекомендуется выбрать для папки хранения проекта то же имя, что и для самого проекта. В нашем случае — это *prjMPY*.

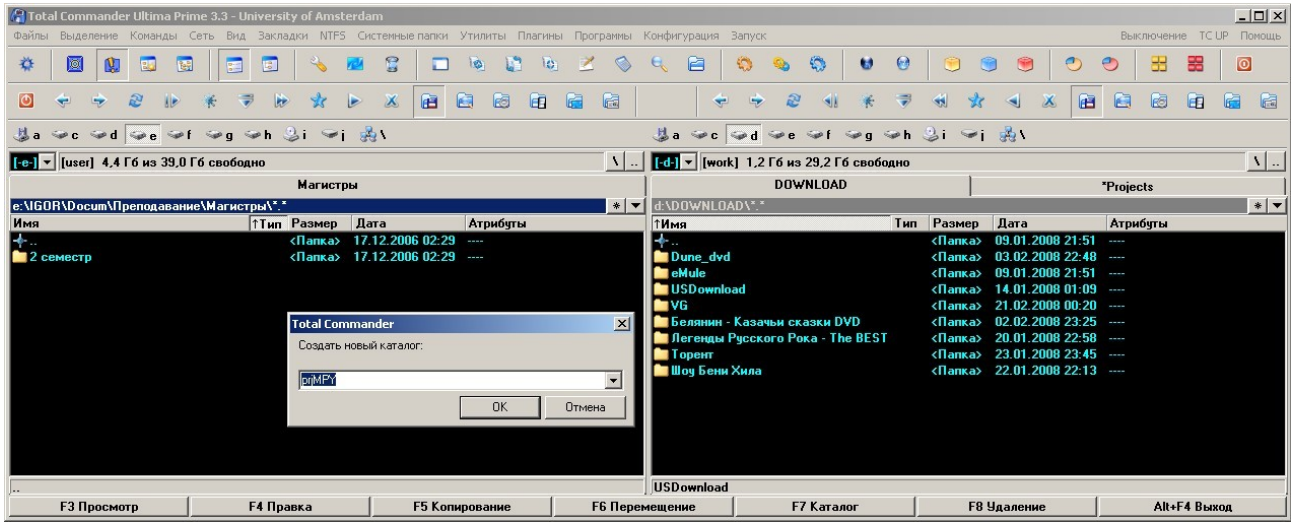

Рис.1. Создание папки для файлов проекта.

1.2. Запустить программу *wvDev-C++*

Для чего нажать мышкой иконку на рабочем столе (или выбрать соответствующий пункт в меню через *«Пуск->Программы»* операционной системы *Windows)*. Появившееся окно программы показано на рис.2.

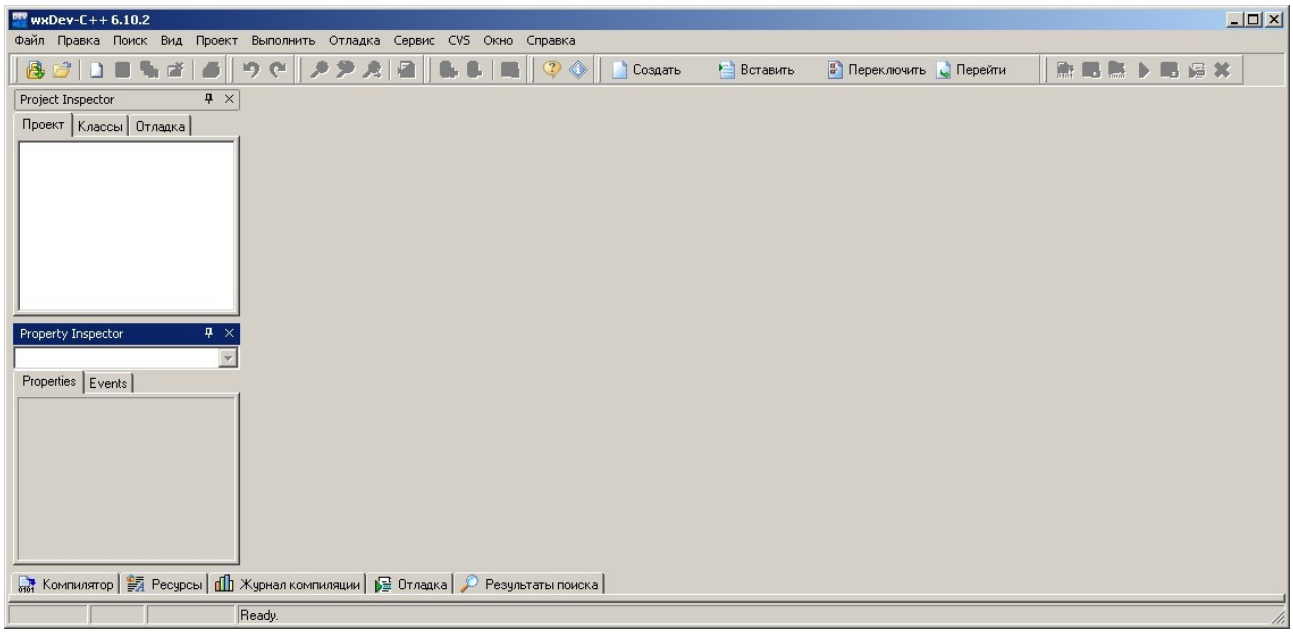

Рис.2. Окно программы wvDev-C++.

1.3. Создать проект

Для этого в главном меню программы *wvDev-C++* необходимо выбрать пункт *«Файл->Создать->Проект»,* как это показано на рис.3.

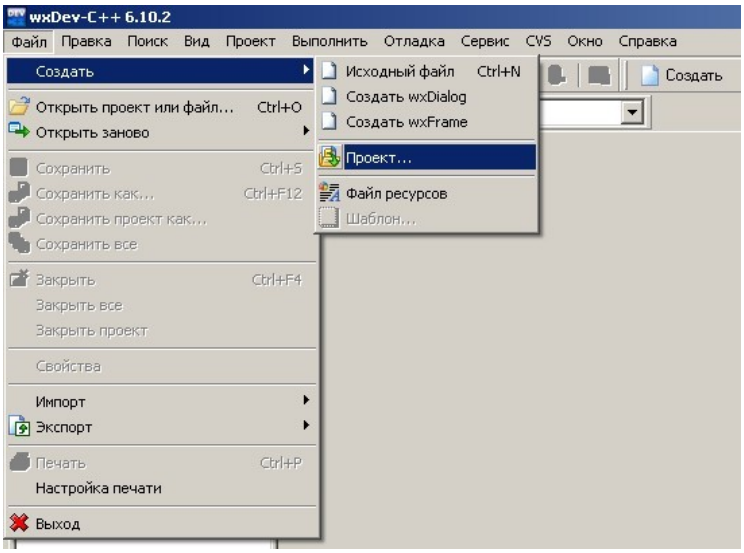

Рис.3. Создание проекта.

В появившемся окне «Новый проект» необходимо выбрать вид проекта — *Console Application,* имя проекта — в соответствии с именем папки для проекта (например, *prjMPY),* а так же тип проекта — *C++ проект.* Все настройки проекта показаны на рис.4.

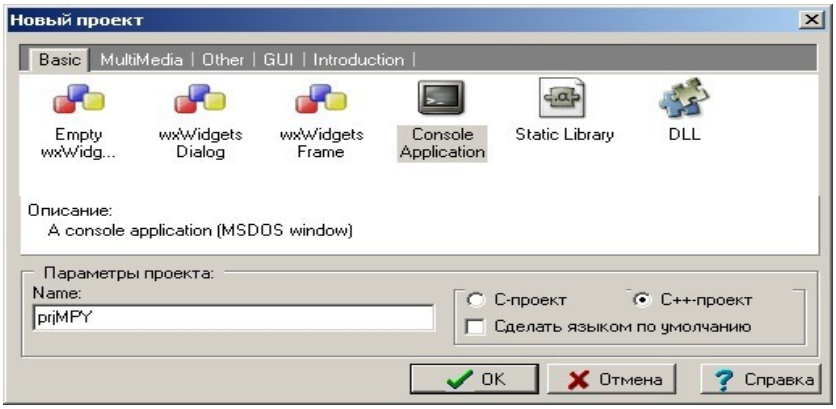

Рис.4. Настройки проекта.

После внесение всех параметров необходимо нажать кнопку *«OK».* Появиться окно *«Create new project»,* показанное на рис.5. Необходимо указать в качестве места размещения проекта созданную в пункте 1 папку для хранения файлов проекта.

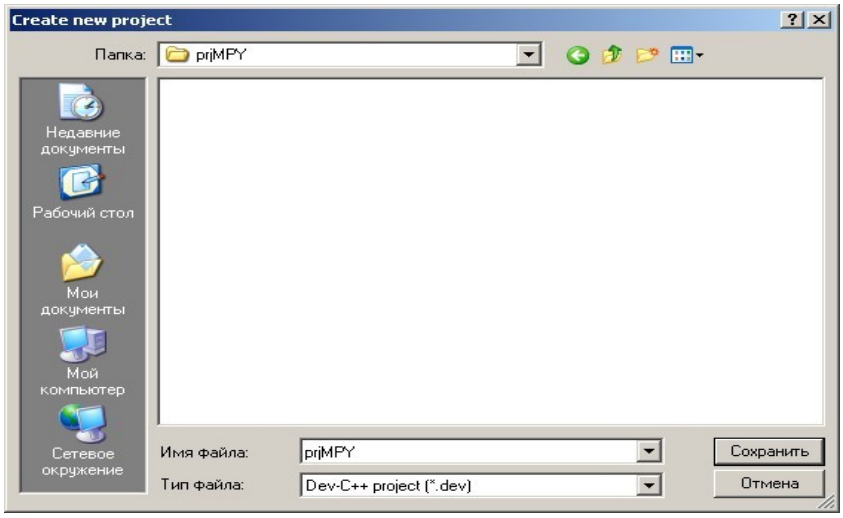

Рис.5. Выбор места хранения проекта.

Окно программы *wvDev-C++* примет вид показанный на рис.6. Проект создан.

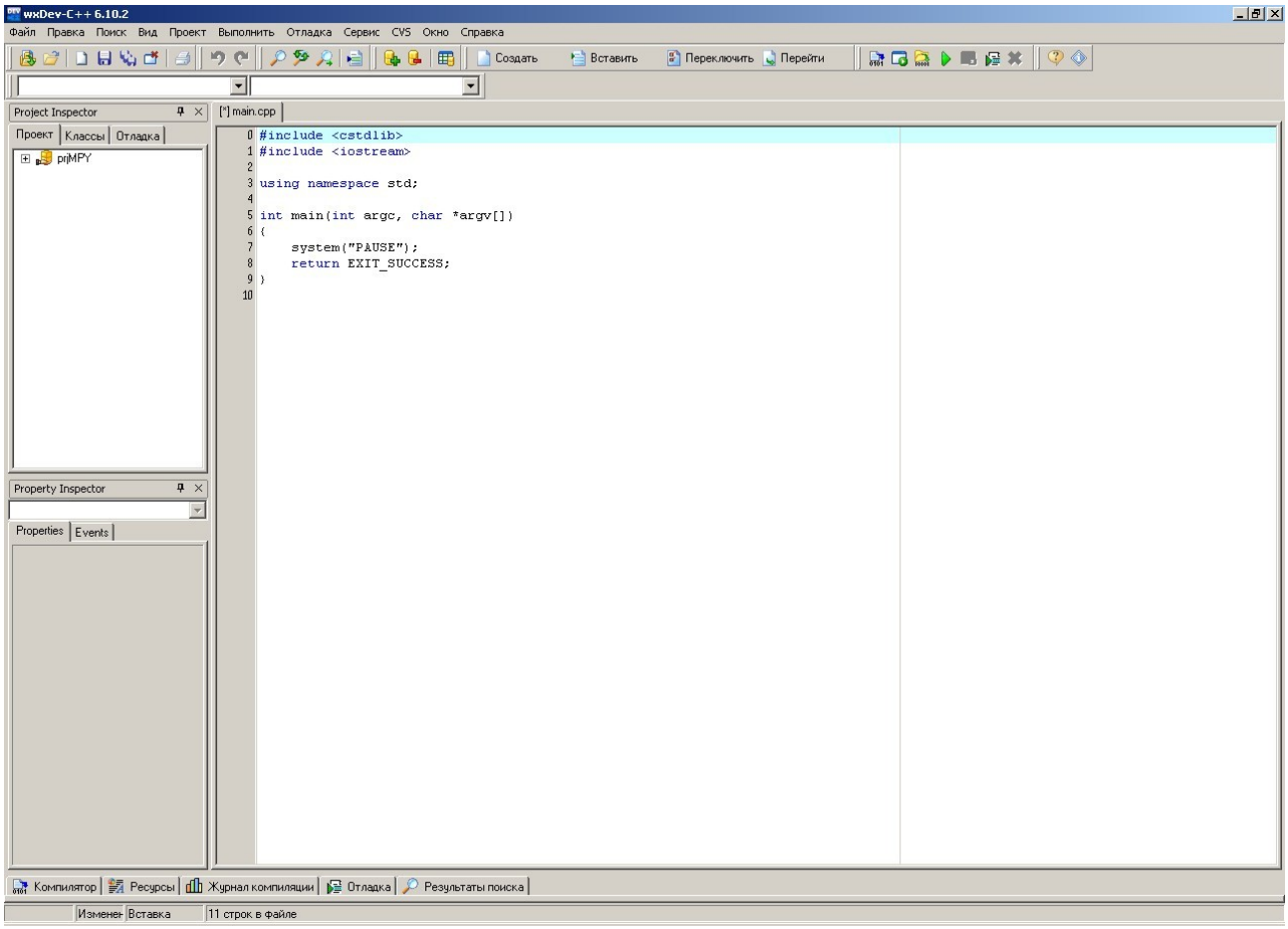

Рис.6. Окно программы wvDev-C++ после создания проекта.

1.4. Откомпилировать созданный проект

Для этого нужно нажать зеленую треугольного кнопку на панели инструментов программы *wvDev-C++*, как это показано на рис.7.

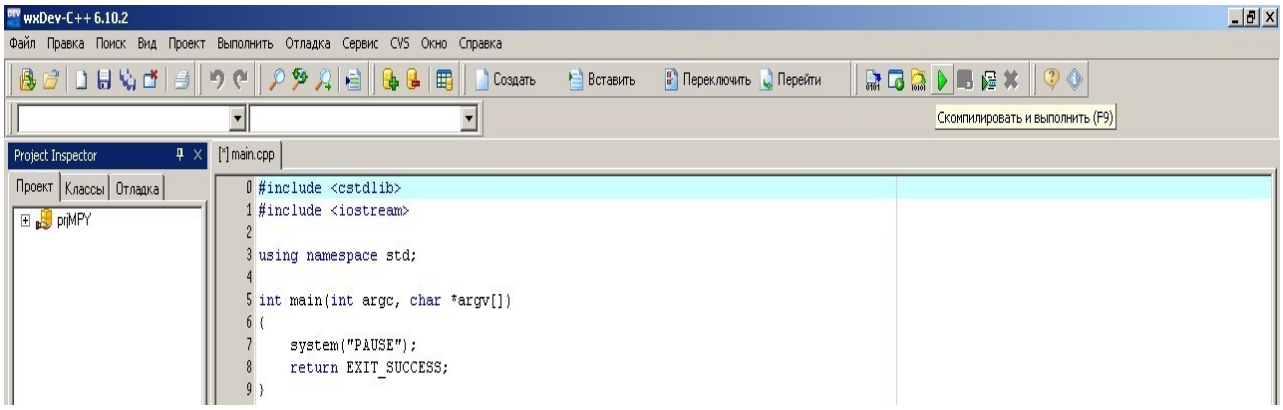

Рис.7. Компиляция созданного проекта.

Появиться окно запроса на сохранения файла с функцией main() (см. рис.8). Подтвердите сохранение файла нажав кнопку *«Сохранить».*

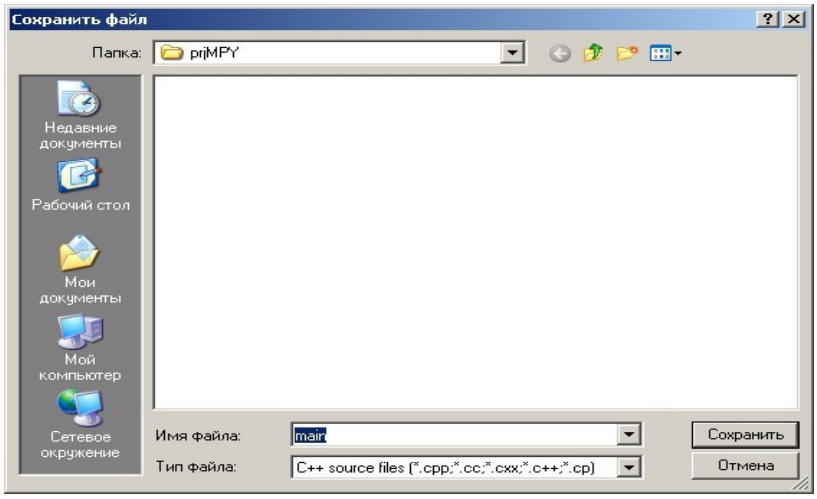

Рис.8. Запрос на сохранение функции main().

Начнется процесс компиляции проекта, что будет отражено появление окна *«Compile Progress»,* показанного на рис.9.

Рис.9. Процесс компиляции проекта.

Если процесс компиляции произошел успешно, появиться консольное окно, показанное на рис.10. В этом окне будет предложение нажать любую кнопку. Последуйте этому совету.

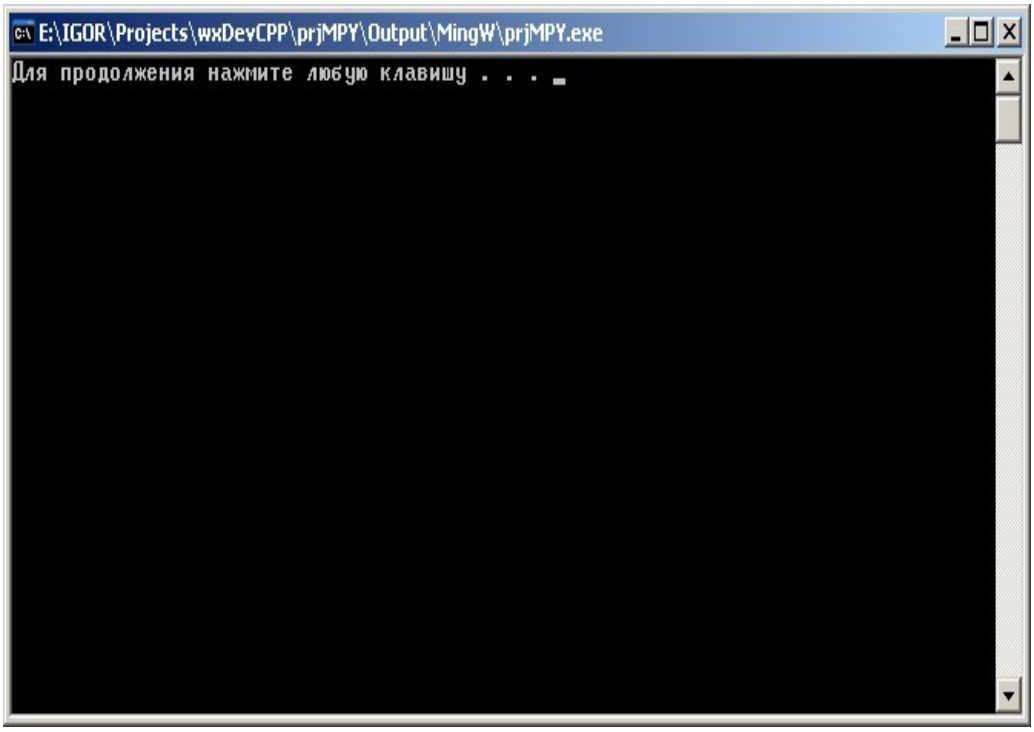

Рис.10. Успешное завершение процесса компиляции.

Первая программа успешно создана, откомпилирована и запущенна на выполнение. Правда, пока еще эта программа ничего не делает.

При помощи файлового менеджера просмотрите папку проекта (см. рис.11). Еще не было написано ни строчки кода, а проект уже содержит достаточно большое количество файлов и папок. Все они (за исключением файла с функцией *main())* необходимы для хранения служебных данных процесса компилирования.

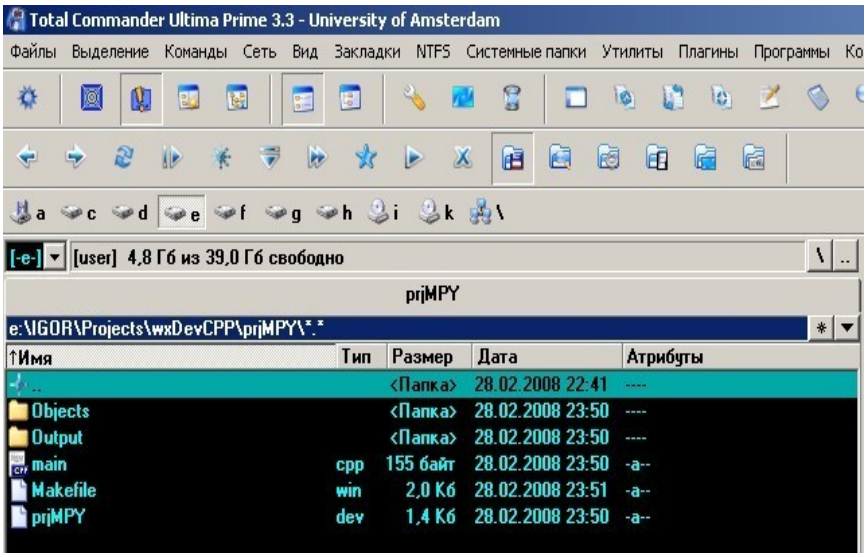

Рис.11. Папка проекта после создания, компиляции и тестового запуска проекта.

С этого момента начинается разработка алгоритма и написание программного кода нашего алгоритма.

2. Разработка алгоритма для простейшего выражения

Необходимо разработать алгоритма для вычисления выражения:

$$y(n) = A \cdot x(n) , \qquad (1)$$

где $y(n)$ — выходной сигнал,

$x(n)$ — входной сигнал,

A — постоянный масштабирующий коэффициент,

n — нормированное время.

Для выполнения процедуры, описываемой выражением (1) необходимо выполнить следующие действия:

1. Произвести инициализацию начального состояния алгоритма.
2. Проверить наличие данных во входном файле: если данные есть, то перейти к следующему пункту, в противном случае — перейти к п.6.
3. Считать данные из файла во входной буфер.
4. Обработать считанные данные и записать результат в выходной буфер.
5. Записать выходной буфер в выходной файл.
6. Завершить выполнение программы.

Графическое представление алгоритма представлено на рис.12.

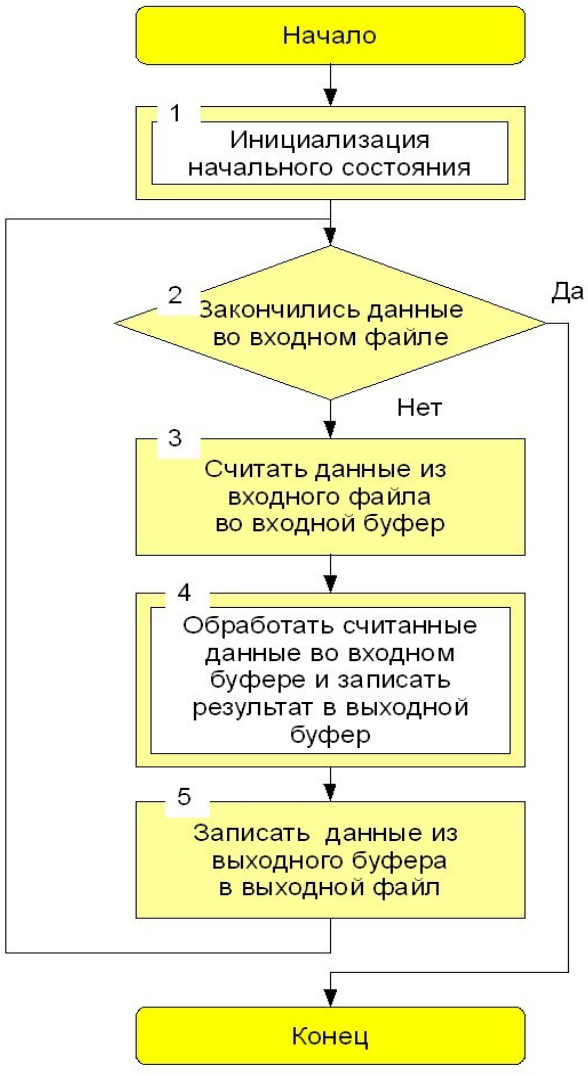

Рис.12. Алгоритм простейшей программы.

Отметим, что п.п.1 и 2 должны быть реализованы в виде подпрограмм (то есть в виде отдельных функций).

3. Написание программного кода простейшего алгоритма

Напомним, что программный код должен соответствовать требованиям файловой модели, которая предполагает:

- Наличие одного заголовочного файла на проект.
- Разбиение алгоритма на функции и размещение каждой функции в отдельном файле.
- Создание всех глобальных переменных в отдельном файле.
- Передача параметров между функциями через контекстную структуру.
- Функция main() выполняет только считывание/запись данных и вызов интерфейсных функций проекта, и не содержит программного кода, соответствующего разрабатываемому алгоритму.
- Размещение входных и выходных данных в соответствующих файлах.
- Запрет на использования статических переменных. При необходимости такие переменные заменяются глобальными.
- Запрет на создание переменных, констант, функций и т.д. в заголовочном файле проекта.

Итак, приступим к написанию кода алгоритма в соответствии с требования файловой модели.

3.1. Создать заголовочный файл разрабатываемого проекта

Для этого в главном меню программы *wvDev-C++* необходимо выбрать пункт *«Файл->Создать->Исходный файл»*, как это показано на рис.7.

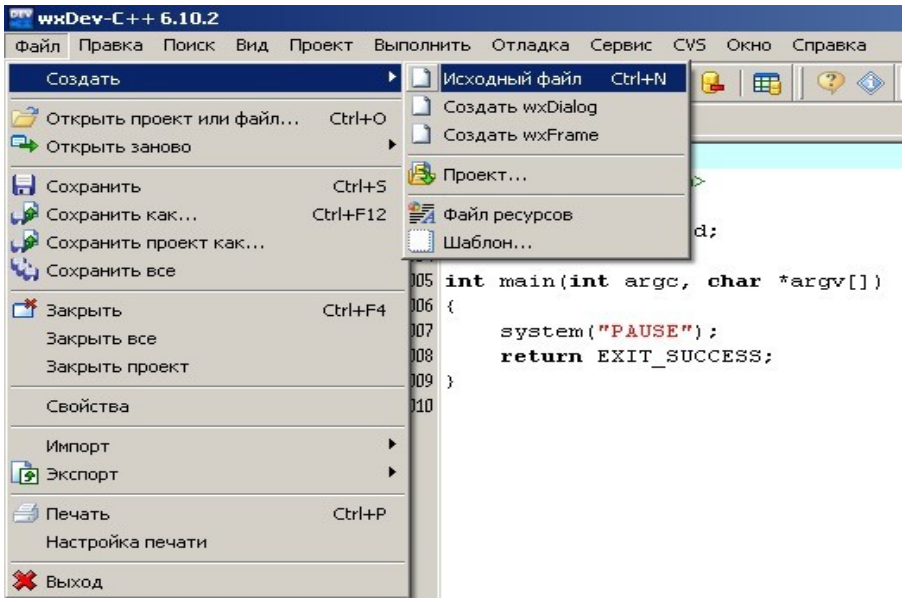

Рис.12. Окно программы wvDev-C++.

Появиться окошко (см.рис.13) с запросом на подключение данного файла к текущему проекту. Необходимо подтвердить это, нажав кнопку *«Yes»*.

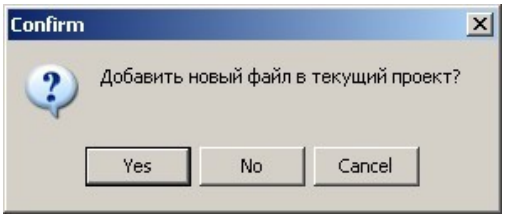

Рис.13. Подключение создаваемого файла к текущему проекту.

Программа *wvDev-C++* примет вид, показанный на рис.14.

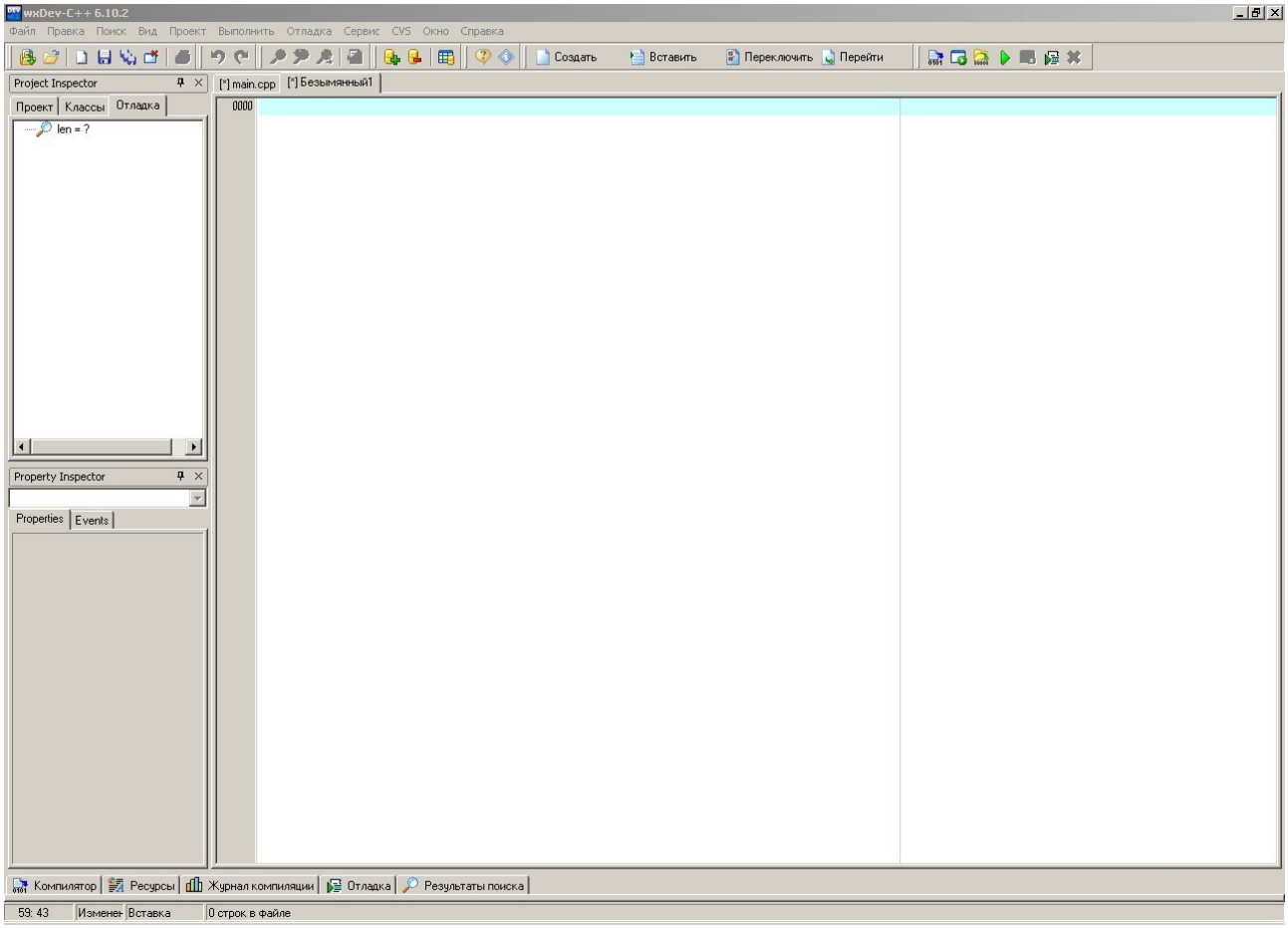

Рис.14. Окно программы wvDev-C++ при создании нового файла.

После этого необходимо сохранить созданный файл в папке проекта. Для чего необходимо в главном меню выбрать пункт «*Файл->Сохранить как...»*(см.*рис.15).*

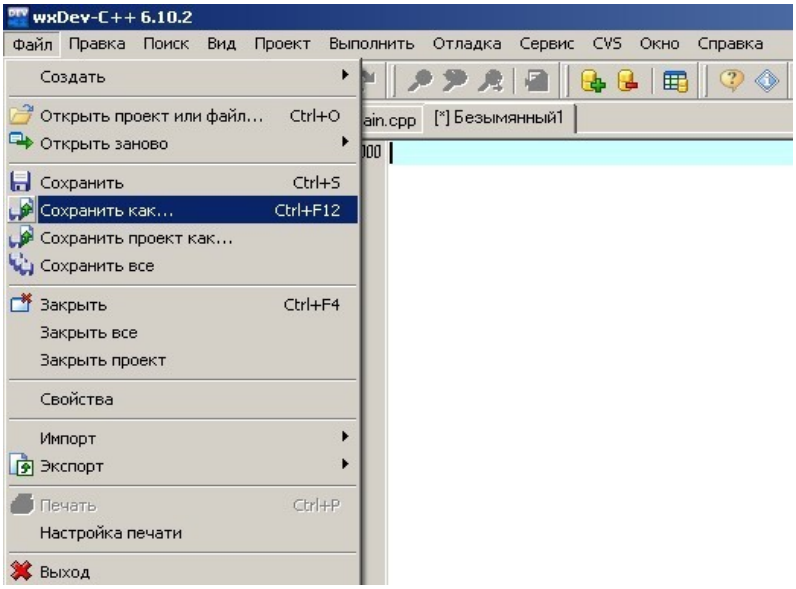

Рис.15. Сохранение создаваемого файла.

Появиться окно, где необходимо выбрать место хранения, тип и имя для созданного файла. Для хранения необходимо указать папку проекта (обычно она указана по умолчанию). Имя файла рекомендуется выбрать в соответствии с именем проекта (например, *prjMPY).* Тип файла — *Headers files.* Все настройки сохранения файла показаны на рис.16.

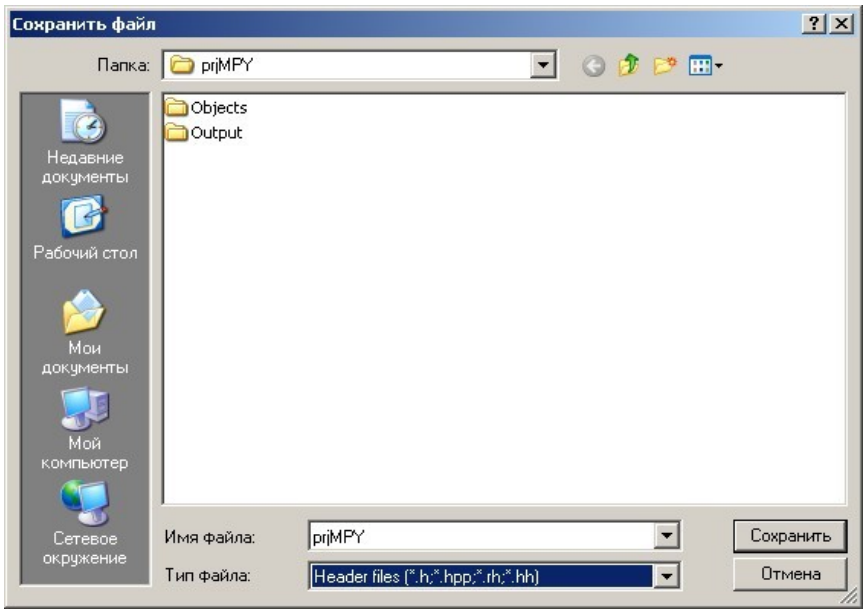

Рис.16. Указание места, имени и типа сохраняемого файла.

На этом создание нового файла завершено. В этом можно убедиться, просмотрев файлы в папке проекта в файловом менеджере. Кроме этого, перейдя на закладку *«Проект»* в окне *«Project Inspector» (см.рис.17)* можно увидеть все файлы текущего проекта.

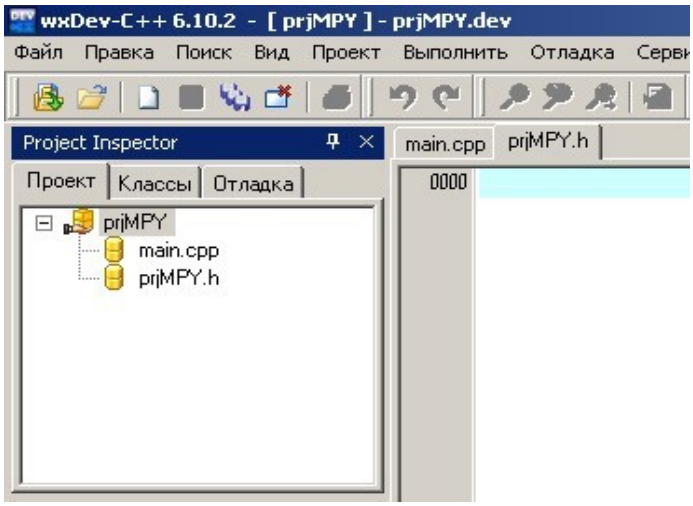

Рис.17. Закладка «Проект» в окне «Project Inspector» после создания файла prjMPY.h.

3.2. Модифицировать функцию *main()*

Необходимо в окне *«Project Inspector»* выбрать закладку *«Проект»* и развернуть проект как это показано на рис.17 (для этого может понадобиться щелкнуть мышкой по значку «+» напротив имени проекта). Затем щелкнуть мышкой по имени файла *main()*. В текстовом редакторе откроется закладка с окном редактирования файла *main()*. В этом окне необходимо удалить существующий текст и набрать заново следующее:

```
#include "prjMPY.h"

int main(void) {

    // Выход из программы
    return 0;
}
```

Первая строчка подключает к файлу с функций *main()* файл *prjMPY.h.* Дальше описывается функция *main()*. Пока еще эта функция ничего не делает. Постепенно текст будет изменяться в соответствии с нашим алгоритмом.

3.3. Модифицировать заголовочный файл *prjMPY.h*

Переходим на закладку с окном редактирования файла *prjMPY.h* в текстовом редакторе и набираем текст:

```
// Подключение внешних библиотек
#include <stdio.h>
```

Первая строчка отмеченная двумя наклонными черточками является комментарием и в процессе компиляции не участвует. Вторая строчка подключает к файлу стандартную библиотеку ввода/вывода, которая понадобится для организации считывания данных из входного файла и записи в выходной файл результата работы программы.
В дальнейшем и этот файл будет постепенно дополняться текстом в соответствии с реализуемым алгоритмом.

3.4. Реализовать программный код инициализации начального состояния алгоритма

Данный пункт соответствует бл.1 на рис.12. Для его реализации в функции *main()* необходимо набрать код вызова функции инициализации. Файл с функцией *main()* будет выглядеть следующим образом:

```
#include "prjMPY.h"

int main(void) {

    // Инициализация начального состояния
    initMPY(pCntx);

    // Выход из программы
    return 0;
}
```

Если попробовать провести компиляцию проекта, появиться сообщение об ошибке. Эта ситуация показана на рис.18. Ошибка отмечена коричневым цветом, а внизу появилось окно с описание типа ошибки.

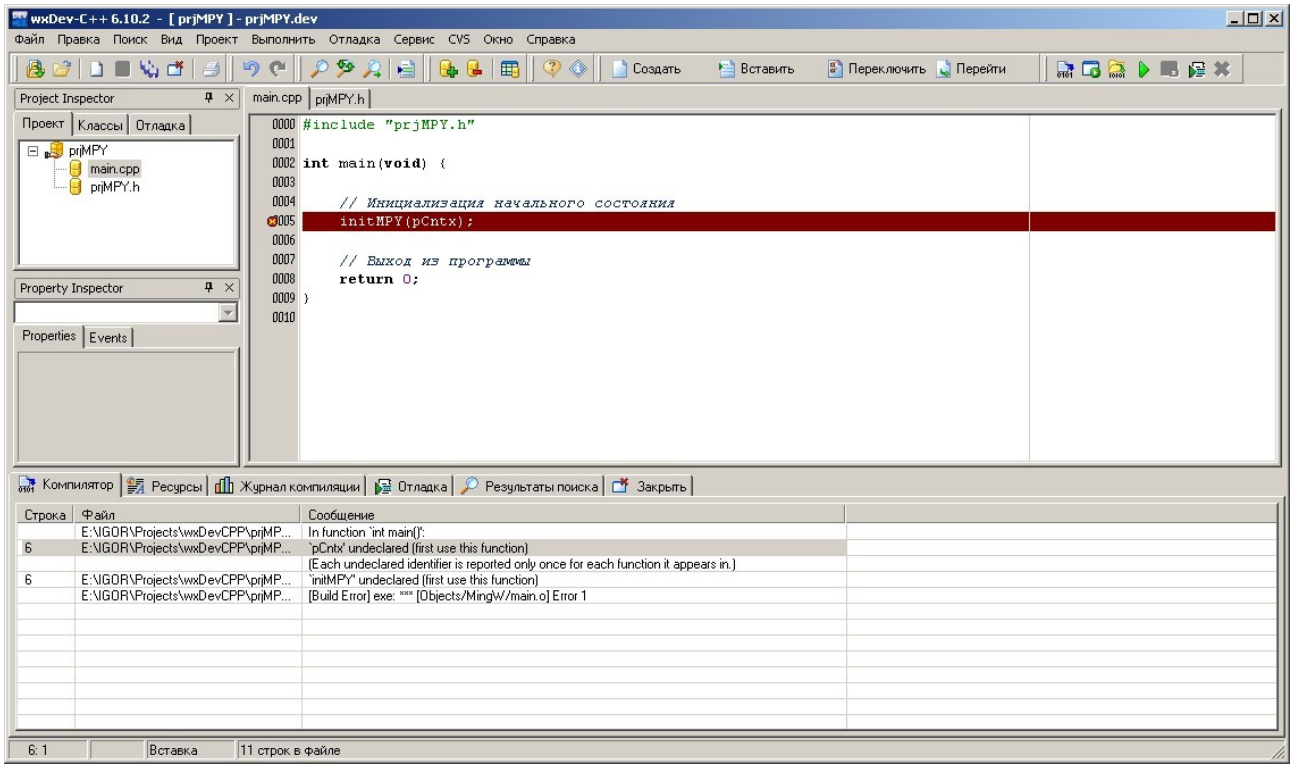

Рис.18. Сообщение об ошибки компилирования проекта.

Для исправления ситуации необходимо создать файл для хранения функции инициализации и

описать в нем функцию. Создание файла происходит аналогично тому, как это делалось при создании заголовочного файла. За исключением того, что имя файла должно соответствовать имени функции — *initMPY,* а при указании типа сохраняемого файла необходимо выбрать — C++ source files, как это показано на рис.19.

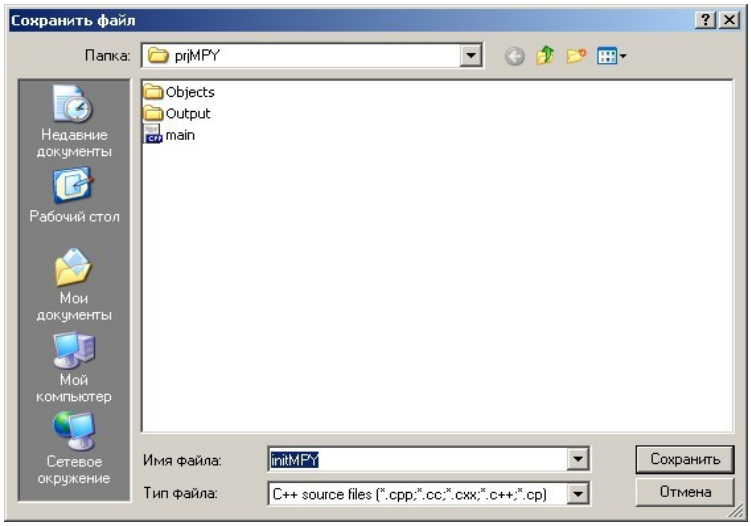

Рис.19. Указание места, имени и типа при сохранении файла для функции инициализации.

В проекте появиться новый файл. Необходимо в окне текстового редактора перейти на закладку для редактирования файла *initMPY.cpp* и набрать текст:

```
// Функция инициализации начального состояния
// разрабатываемого проекта

#include "prjMPY.h"

void initMPY(CONTEXMPY* pCntx) {

}
```

Компилятор на первом этапе преобразования текстового файла в бинарный исполняемый код каждый файл с исходным кодом компилирует независимо друг от друга. Необходимо указать компилятору все функции, существующие в проекте. Это можно сделать объявив функцию в заголовочном файле. Поэтому необходимо дополнить заголовочный файл следующим текстом:

```
// Подключение внешних библиотек
#include <stdio.h>

// Объявление функций проекта
void initMPY(CONTEXMPY* pCntx);
```

Обратите внимание, что в качестве параметра функции инициализации используется неизвестный тип — CONTEXMPY*. Суффикс «*» говорит о том, что это указатель на тип CONTEXMPY. Но сам тип CONTEXMPY не является стандартным типом для языка Си и его необходимо определить в заголовочном файле следующим образом:

```
// Подключение внешних библиотек
#include <stdio.h>


// Определение типа контекстной структуры
typedef struct {
;
} CONTEXMPY;

// Объявление функций проекта
void initMPY(CONTEXMPY* pCntx);
```

Если попытаться провести компиляцию проекта снова появиться сообщение об ошибке. Но это уже ошибка другого типа — компилятор сообщает, что ему неизвестна переменная с именем *pCnt,* которая передается функции инициализации. Эту переменную необходимо создать.

3.5. Создание указателя *pCnt*

Для этого вначале создается файл для хранения всех переменных, констант, массивов проекта с именем *constant.cpp*. Процесс уже описывался выше. На рис.20 показано окно сохранения со всеми необходимыми настройками.

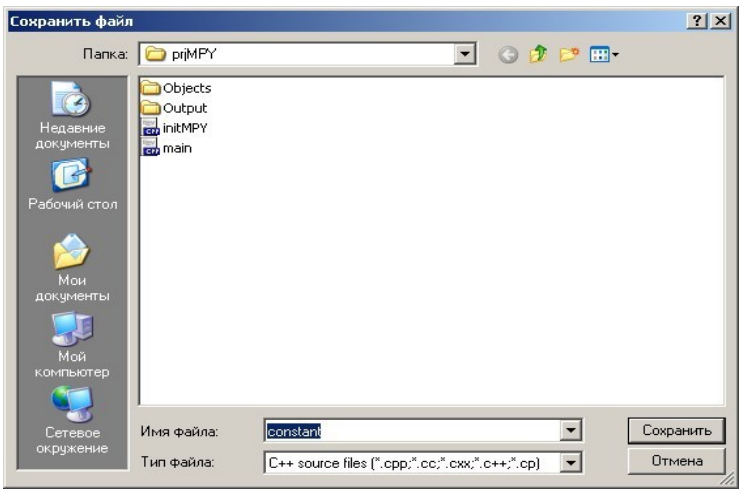

Рис.20. Сохранение файла constant.cpp.

Затем необходимо перейти на закладку файла *constant.cpp* в текстовом редакторе и набрать следующий текст:

```
#include "prjMPY.h"

// Создание указателя на контекстную структуру
CONTEXMPY* pCntx;
```

И в заключении, необходимо объявить данную переменную в заголовочном файле, для того чтобы она была видна во всех файлах проекта:

```
// Подключение внешних библиотек
#include <stdio.h>

// Определение типа контекстной структуры
typedef struct {
;
} CONTEXMPY;

// Объявление функций проекта
void initMPY(CONTEXMPY* pCntx);

// Объявление переменных, массивов и т.д.
extern CONTEXMPY* pCntx;
```

Для объявления переменных в заголовочном файле служит слово extern перед типом и именем переменной. Отметим, что объявление переменной не ведет к созданию переменной. Но, указатель никуда *pCnt* не указывает. Поэтому необходимо создать переменную *cntx* типа CONTEXMPY в файле *constant.cpp*:

```
#include "prjMPY.h"

// Создание указателя на контекстную структуру
CONTEXMPY* pCntx;

// Создание контекстной структуры
CONTEXMPY cntx;
```

Объявить ее в файле *prjMPY.h*:

```
// Подключение внешних библиотек
#include <stdio.h>

// Определение типа контекстной структуры
typedef struct {
;
} CONTEXMPY;

// Объявление функций проекта
void initMPY(CONTEXMPY* pCntx);

// Объявление переменных, массивов и т.д.
extern CONTEXMPY* pCntx;
extern CONTEXMPY cntx;
```

И провести инициализацию указателя в функции *main()*:

```
#include "prjMPY.h"

int main(void) {

    // Инициализация указателя на контекстную структуру
    pCntx = &cntx;

    // Инициализация начального состояния
    initMPY(pCntx);

    // Выход из программы
    return 0;
}
```

Обратите внимание на знак «&». Этот оператор производит вычисление («взятие») адреса переменной *cntx*.
Теперь можно провести компиляцию проекта и убедиться в отсутствии ошибок.

3.6. Определение оригинальных имен стандартных типов.

Это обусловленное тем, что одни и те же стандартные типы могут иметь различное значение для различных типов процессоров. Например, тип *int* может быть в одном случае 32-и разрядным целым числом, а в другом — только 16-ти разрядным.
Определение имен делается при помощи оператора *typedef* в заголовочном файле:

```
// Подключение внешних библиотек
#include <stdio.h>

// Определение оригинальных имен стандартных типов
typedef int   word32;
typedef short word16;
typedef char  word8;

// Определение типа контекстной структуры
typedef struct {
;
} CONTEXMPY;

// Объявление функций проекта
void initMPY(CONTEXMPY* pCntx);

// Объявление переменных, массивов и т.д.
extern CONTEXMPY* pCntx;
extern CONTEXMPY cntx;
```

В дальнейшем в коде программы будут использоваться только оригинальные имена типов.

3.7. Создание указателей на входной/выходной файлы.

В файлах хранятся входные данные и результат работы разрабатываемой программы.
Создание указателей производиться в файле *main()*:

```c
#include "prjMPY.h"

int main(void) {
    // Объявление переменных
    FILE *pInpFile;  // Указатель на входной файл
    FILE *pOutFile;  // Указатель на выходной файл

    // Инициализация указателя на контекстную структуру
    pCntx = &cntx;

    // Инициализация начального состояния
    initMPY(pCntx);

    // Выход из программы
    return 0;
}
```

Это все. Файловые указатели созданы. Теперь их необходимо привязать к реальным файлам.

3.8. Связывание файловых указателей с реальными файлами, где хранятся входные данные и результат вычислений.

Данная операция производиться при помощи функции *fopen()* в файле main():

```c
#include "prjMPY.h"

int main(void) {
    // Объявление переменных
    FILE *pInpFile;  // Указатель на входной файл
    FILE *pOutFile;  // Указатель на выходной файл

    // Вывод сообщения о старте программы
    printf("START PROGRAMS!\n");

    // Открыть входной файл файлов для чтения
    pInpFile = fopen("inp.dat", "rb");

    // Открыть выходной файл файлов для записи
    pOutFile = fopen("out.dat", "wb");

    // Инициализация указателя на контекстную структуру
    pCntx = &cntx;

    // Инициализация начального состояния
    initMPY(pCntx);

    // Выход из программы
    return 0;
}
```

Первый параметр функции *fopen()* — это имя файла, второй — тип файла и условия его подключения. Параметр «*rb*» говорит о том, что файл бинарный (буква «*b*») и открывается только для чтения (буква «*r*»). Это в свою очередь означает, что такой файл должен существовать на момент его открытия. Его можно создать, например, в программе EDSW в виде синусоидального сигнала (см.рис.21) и сохранить в папке проекта с именем *inp.dat*. Параметр «*wb*» означает, что файл бинарный (буква «*b*») и открывается для записи (буква «*w*»). Файл может не существовать на момент его открытия. В этом случае он создается и имеет нулевую длину. Если же такой файл на момент открытия существует, он уничтожается и создается снова (опять же — нулевой длинны!).

Обратите внимание, что появилась еще одна функция — *printf()*. При помощи этой функции можно выводить различные сообщения о ходе выполнения программы. В данном случае выводиться текстовое сообщение о начале выполнения программы. Выводимый текст необходимо набирать в кавычках и только в латинской раскладке. При помощи вспомогательных символов, таких как «\n», можно производить форматирования выводимого сообщения. В приведенном примере производиться перевод курсора на новую строку после вывода сообщения «START PROGRAMS!».

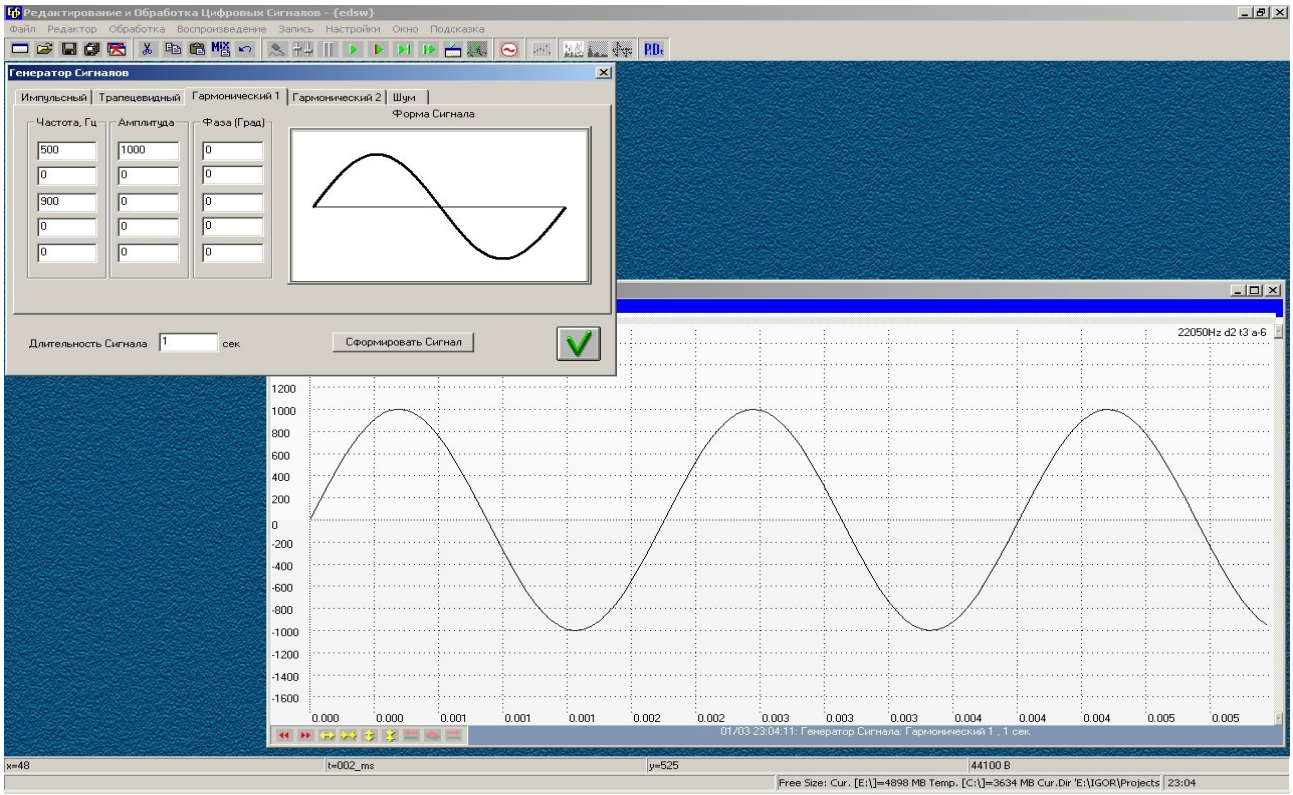

Рис.21. Создание синусоидального сигнала в программе EDSW.

После связывания файлового указателя непосредственно с файлом проводиться процедура проверки успешного выполнения операции. Для этого используется оператор *if*. Если операция связывания прошла успешно (в этом случае файловому указателю в результате выполнения функции *fopen()* будет присвоено значение отличное от NULL), код в фигурных скобках после оператора *if* пропускается. В противном случае (файловому указателю присвоено значение NULL) будет выведено сообщение об ошибке (для чего используется функция *printf()* в фигурных скобках после операторе *if*) и прекращено выполнение программы при помощи оператора *return*. Код функции *main()* к этому моменту имеет вид:

```
#include "prjMPY.h"

int main(void) {
     // Объявление переменных
     FILE *pInpFile;   // Указатель на входной файл
     FILE *pOutFile;   // Указатель на выходной файл

     // Вывод сообщения о старте программы
     printf("START PROGRAMS!\n");

     // Открыть входной файл файлов для чтения
     pInpFile = fopen("inp.dat", "rb");

     // Проверить корректность открытия входного файла
     if(pInpFile == NULL) {
          printf("ERROR, INPUT FILE!\n");
          return 2;
     }
     printf("OPEN INPUT FILE SUCSESFUL!\n");

     // Открыть выходной файл файлов для записи
     pOutFile = fopen("out.dat", "wb");

     // Проверить корректность открытия выходного файла
     if(pOutFile == NULL) {
          printf("ERROR, OUT FILE\n");
          return 1;
     }
     printf("GREAT OUTPUT FILE SUCSESFUL!\n");

     // Инициализация указателя на контекстную структуру
     pCntx = &cntx;

     // Инициализация начального состояния
     initMPY(pCntx);

     // Выход из программы
     return 0;
}
```

Обратите внимание, что функция *printf()* используется не только для вывода сообщений об ошибке, но и для иллюстрации процесса выполнения программы в целом. Эта функция бывает очень удобна в процессе отладки программного кода, ее можно вставлять в код функции и отслеживать корректность ее работы. Но в рамках файловой модели ее можно использовать только в функции *main()*.

И последнее замечание по работе с файловыми указателями. Если в программе открываются файлы (т.е. если происходит связывание файловых указателей с файлами), тогда перед выходом из программы необходимо все открытые файлы закрыть. Данная операция производиться функцией *fclose()*:

```
#include "prjMPY.h"

int main(void) {
    // Объявление переменных
    FILE *pInpFile;  // Указатель на входной файл
    FILE *pOutFile;  // Указатель на выходной файл

    // Вывод сообщения о старте программы
    printf("START PROGRAMS!\n");

    // Открыть входной файл файлов для чтения
    pInpFile = fopen("inp.dat", "rb");

    // Проверить корректность открытия входного файла
    if(pInpFile == NULL) {
        printf("ERROR, INPUT FILE!\n");
        return 2;
    }
    printf("OPEN INPUT FILE SUCSESFUL!\n");

    // Открыть выходной файл файлов для записи
    pOutFile = fopen("out.dat", "wb");

    // Проверить корректность открытия выходного файла
    if(pOutFile == NULL) {
        printf("ERROR, OUT FILE\n");
        return 1;
    }
    printf("GREAT OUTPUT FILE SUCSESFUL!\n");

    // Инициализация указателя на контекстную структуру
    pCntx = &cntx;

    // Инициализация начального состояния
    initMPY(pCntx);

    // Закрыть входной и выходной файлы
    fclose(pInpFile);
    fclose(pOutFile);

    // Выход из программы
    return 0;
}
```

3.9. Создание входного/выходного буферов.

В этих буферах будут храниться исходные данные для работы алгоритма, а так же результат вычислений. Буферы (или массивы) создаются, как и все переменные, в файле в файле *constant.cpp*:

```
#include "prjMPY.h"

// Создание указателя на контекстную структуру
CONTEXMPY* pCntx;

// Создание контекстной структуры
CONTEXMPY cntx;

// Массив для входных данных
word16 inpBuff[SAZEBUFF];

// Массив для выходных данных
word16 outBuff[SAZEBUFF];
```

При создании массивов указывается тип элементов массива (в данном случае *word*16 — оригинальное имя стандартного типа *short*), имя массива (*inpBuff* и *outBuff*) и в квадратных скобках — размер массива. Размер массива может указываться числом, или, как в этом случае, при помощи так называемого макроса. При компиляции проекта будет произведена замена слова SAZEBUFF числом, которое определяется директивой *#define* в файле *prjMPY.h*:

```
// Подключение внешних библиотек
#include <stdio.h>

// Определение макросов
#define SAZEBUFF   120   // Размер входного и выходного буферов

// Определение оригинальных имен стандартных типов
typedef int   word32;
typedef short word16;
typedef char  word8;

// Определение типа контекстной структуры
typedef struct {
;
} CONTEXMPY;

// Объявление функций проекта
void initMPY(CONTEXMPY* pCntx);

// Объявление переменных, массивов и т.д.
extern CONTEXMPY* pCntx;        // Указатель на контекстную структуру
extern CONTEXMPY cntx;          // Контекстная структура
```

И в заключении необходимо объявить буферы в файле *prjMPY.h*:

```
// Подключение внешних библиотек
#include <stdio.h>

// Определение макросов
#define SAZEBUFF   120   // Размер входного и выходного буферов

// Определение оригинальных имен стандартных типов
typedef int   word32;
typedef short word16;
typedef char  word8;

// Определение типа контекстной структуры
typedef struct {
;
} CONTEXMPY;

// Объявление функций проекта
void initMPY(CONTEXMPY* pCntx);

// Объявление переменных, массивов и т.д.
extern CONTEXMPY* pCntx;        // Указатель на контекстную структуру
extern CONTEXMPY cntx;          // Контекстная структура
extern word16 inpBuff[];        // Массив для входных данных;
extern word16 outBuff[];        // Массив для выходных данных;
```

Обратите внимание, что при объявлении массивов в квадратных скобках ничего не указывается.

3.10. Создание указателей на входной/выходной буферы в контекстной структуре.

Все параметры между функциями алгоритма должны передаваться через контекстную структуру (это требование файловой модели). Поэтому, в контекстной структуре вводятся два поля для указателей на входной/выходной буферы, а так же поле для хранения их размера (так как в данном случае для входного и выходного массивов он один и тот же):

```
// Подключение внешних библиотек
#include <stdio.h>

// Определение макросов
#define SAZEBUFF   120   // Размер входного и выходного буферов

// Определение оригинальных имен стандартных типов
typedef int   word32;
typedef short word16;
typedef char  word8;

// Определение типа контекстной структуры
typedef struct {
    word16 *pInpBuff;   // Указатель на входной буфер
    word16 *pOutBuff;   // Указатель на выходной буфер
    word16 lenBuff;     // Размером входного и выходного буферов
} CONTEXMPY;

// Объявление функций проекта
void initMPY(CONTEXMPY* pCntx);

// Объявление переменных, массивов и т.д.
extern CONTEXMPY* pCntx;        // Указатель на контекстную структуру
extern CONTEXMPY cntx;          // Контекстная структура
extern word16 inpBuff[];        // Массив для входных данных;
extern word16 outBuff[];        // Массив для выходных данных;
```

Затем необходимо провести инициализацию (присвоение значений) параметров контекстной структуре. Это производиться в функции инициализации:

```
// Функция инициализации начального состояния
// разрабатываемого проекта

#include "prjMPY.h"

void initMPY(CONTEXMPY* pCntx) {
    pCntx->pInpBuff = inpBuff;     // Указатель на входной буфер
    pCntx->pOutBuff = outBuff;     // Указатель на выходной буфер
    pCntx->lenBuff  = SAZEBUFF;    // Размером входного и выходного буферов

}
```

Небольшие пояснения. Первое. Обращение к структуре cntx производиться через указатель на эту структуру *pCntx*. Что бы обратиться к конкретному полю структуры через указатель на нее используется оператор «->» с указанием имени поля. Второе. Для определения адреса массива нет необходимости использовать оператор «&», как это делалось при инициализации указателя на контекстную структуру (см. п.3.5). Имя массива и есть адрес массива.

3.11. Реализовать программный код считывания данных из файла во входной буфер

Этот пункт соответствует бл.2 и 3 на рис.12. Для реализации используется оператор организации циклических вычислений *while()* и функция считывания данных из файла *fread():*

```
#include "prjMPY.h"

int main(void) {
    // Объявление переменных
    FILE *pInpFile;  // Указатель на входной файл
    FILE *pOutFile;  // Указатель на выходной файл
    word32 len;      // Переменная цикла

    // Вывод сообщения о старте программы
    printf("START PROGRAMS!\n");

    // Открыть входной файл файлов для чтения
    pInpFile = fopen("inp.dat", "rb");

    // Проверить корректность открытия входного файла
    if(pInpFile == NULL) {
        printf("ERROR, INPUT FILE!\n");
        return 2;
    }
    printf("OPEN INPUT FILE SUCSESFUL!\n");

    // Открыть выходной файл файлов для записи
    pOutFile = fopen("out.dat", "wb");

    // Проверить корректность открытия выходного файла
    if(pOutFile == NULL) {
        printf("ERROR, OUT FILE\n");
        return 1;
    }
    printf("GREAT OUTPUT FILE SUCSESFUL!\n");

    // Инициализация указателя на контекстную структуру
    pCntx = &cntx;

    // Инициализация начального состояния
    initMPY(pCntx);

    // Инициализировать переменную цикла
    len = SAZEBUFF;

    // Цикл обработки входного буфера
    while(len == SAZEBUFF) {
        // Считать данные из файла
        len = fread(pCntx->pInpBuff, sizeof(word16), SAZEBUFF, pInpFile);

    }

    // Закрыть входной и выходной файлы
    fclose(pInpFile);
    fclose(pOutFile);

    // Выход из программы
    return 0;
}
```

Первый параметр функции *fread()* указывает адрес входного буфера для хранения считанных из файла данных, второй — определяет размер считываемых из файла отсчетов в байтах, третий — количество считываемых из файла отсчетов, четвертый — это указатель на файл с данными.

Цикл будет выполняться до тех пор, пока истинно условие в круглых скобках оператора *while()*. Оператор «==» означает выполнение проверки на равенство двух операндов. В результате формируется значение TRUE (Истинно), или FALSE (Ложно).

Обратите внимание, что создается дополнительная переменная для организации цикла — *len.* Этой переменной присваивается начальное значение равное размеру буфера. Затем, в цикле, происходит присвоение данной переменной значения, которое возвращает функция *printf().*

Данная функция возвращает фактическое количество считанных из входного файла отсчетов. Когда происходит считывание данных в конце файла, их количество становиться меньше размера входного буфера. Переменной *len* присваивается значение не равное размеру входного массива. При очередном сравнении размера буфера, задаваемого макросом SAZEBUFF, в операторе *while()* формируется значение *FALSE* и осуществляется выход из цикла.

3.12. Реализовать программный код записи данных из выходного буфера в файл

Пункт соответствует бл.5 на рис.*12*. В цикл *while()* добавляется функция записи данных из буфера в файла *fwrite():*

```
#include "prjMPY.h"

int main(void) {
    // Обьявление переменных
    FILE *pInpFile;  // Указатель на входной файл
    FILE *pOutFile;  // Указатель на выходной файл
    word32 len;      // Переменная цикла

    // Вывод сообщения о старте программы
    printf("START PROGRAMS!\n");

    // Открыть входной файл файлов для чтения
    pInpFile = fopen("inp.dat", "rb");

    // Проверить корректность открытия входного файла
    if(pInpFile == NULL) {
        printf("ERROR, INPUT FILE!\n");
        return 2;
    }
    printf("OPEN INPUT FILE SUCSESFUL!\n");

    // Открыть выходной файл файлов для записи
    pOutFile = fopen("out.dat", "wb");

    // Проверить корректность открытия выходного файла
    if(pOutFile == NULL) {
        printf("ERROR, OUT FILE\n");
        return 1;
    }
    printf("GREAT OUTPUT FILE SUCSESFUL!\n");

    // Инициализация указателя на контекстную структуру
    pCntx = &cntx;

    // Инициализация начального состояния
    initMPY(pCntx);

    // Инициализировать переменную цикла
    len = SAZEBUFF;

    // Цикл обработки входного буфера
    while(len == SAZEBUFF) {
        // Считать данные из файла
        len = fread(pCntx->pInpBuff, sizeof(word16), SAZEBUFF, pInpFile);

        // Записать данные в файл
        fwrite(pCntx->pOutBuff, sizeof(word16), len, pOutFile);
    }

    // Закрыть входной и выходной файлы
    fclose(pInpFile);
    fclose(pOutFile);

    // Выход из программы
    return 0;
}
```

Параметры функции *fwrite()* аналогичны параметрам функции *fread()*, первый — указывает адрес выходного буфера, второй — определяет размер отсчетов в байтах, третий — количество записываемых в файл отсчетов, четвертый — это указатель на файл с результатом.

3.13. Реализовать программный код функции обработки считанных данных и записать результат в выходной буфер

Это бл.4 на рис.12. Для реализации необходимо в функцию main() вставить вызов функции обработки входного буфера *runMPY()* в цикле *while():*

```
#include "prjMPY.h"

int main(void) {
    // Объявление переменных
    FILE *pInpFile;  // Указатель на входной файл
    FILE *pOutFile;  // Указатель на выходной файл
    word32 len;      // Переменная цикла

    // Вывод сообщения о старте программы
    printf("START PROGRAMS!\n");

    // Открыть входной файл файлов для чтения
    pInpFile = fopen("inp.dat", "rb");

    // Проверить корректность открытия входного файла
    if(pInpFile == NULL) {
        printf("ERROR, INPUT FILE!\n");
        return 2;
    }
    printf("OPEN INPUT FILE SUCSESFUL!\n");

    // Открыть выходной файл файлов для записи
    pOutFile = fopen("out.dat", "wb");

    // Проверить корректность открытия выходного файла
    if(pOutFile == NULL) {
        printf("ERROR, OUT FILE\n");
        return 1;
    }
    printf("GREAT OUTPUT FILE SUCSESFUL!\n");

    // Инициализация указателя на контекстную структуру
    pCntx = &cntx;

    // Инициализация начального состояния
    initMPY(pCntx);

    // Инициализировать переменную цикла
    len = SAZEBUFF;

    // Цикл обработки входного буфера
    while(len == SAZEBUFF) {
        // Считать данные из файла
        len = fread(pCntx->pInpBuff, sizeof(word16), SAZEBUFF, pInpFile);

        // Вызвать функцию обработки входного буфера
        runMPY(pCntx);

        // Записать данные в файл
        fwrite(pCntx->pOutBuff, sizeof(word16), len, pOutFile);
    }

    // Закрыть входной и выходной файлы
    fclose(pInpFile);
    fclose(pOutFile);

    // Выход из программы
    return 0;
}
```

Затем необходимо создать файл для хранения кода функции *runMPY()* и описать в нем саму функцию обработки входного буфера. Все это проводиться аналогично процедуре, описанной в п.3.4 для функции инициализации *initMPY()*. В созданном файле *runMPY.cpp* необходимо написать следующий код:

```
// Функция обработки входного буфера и
// запись результата в выходной буфер

#include "prjMPY.h"

void runMPY(CONTEXMPY* pCntx) {

}
```

И не забыть объявить функцию *runMPY()* в заголовочном файле:

```
// Подключение внешних библиотек
#include <stdio.h>

// Определение макросов
#define SAZEBUFF   120   // Размер входного и выходного буферов

// Определение оригинальных имен стандартных типов
typedef int   word32;
typedef short word16;
typedef char  word8;

// Определение типа контекстной структуры
typedef struct {
    word16 *pInpBuff;   // Указатель на входной буфер
    word16 *pOutBuff;   // Указатель на выходной буфер
    word16 lenBuff;     // Размером входного и выходного буферов
} CONTEXMPY;

// Объявление функций проекта
void initMPY(CONTEXMPY* pCntx);
void runMPY(CONTEXMPY* pCntx);

// Объявление переменных, массивов и т.д.
extern CONTEXMPY* pCntx;        // Указатель на контекстную структуру
extern CONTEXMPY cntx;          // Контекстная структура
extern word16 inpBuff[];        // Массив для входных данных;
extern word16 outBuff[];        // Массив для выходных данных;
```

Практически все готово. Осталось написать код функции *runMPY()*.

3.14. Написать код функции *runMPY()*

Именно эта функция реализует вычисления согласно выражения (1). Функция *runMPY()* должна выполнить поэлементное умножение отсчетов входного буфера и записать результат в выходной буфер.

При этом возникает потребность в масштабирующем коэффициенте. Введем его как еще одно поле в определении контекстной структуры:

```
// Подключение внешних библиотек
#include <stdio.h>

// Определение макросов
#define SAZEBUFF    120  // Размер входного и выходного буферов
#define MASHTAB     5    // Маштабирующий коэффициент

// Определение оригинальных имен стандартных типов
typedef int   word32;
typedef short word16;
typedef char  word8;

// Определение типа контекстной структуры
typedef struct {
    word16 *pInpBuff;   // Указатель на входной буфер
    word16 *pOutBuff;   // Указатель на выходной буфер
    word16 lenBuff;     // Размером входного и выходного буферов
    word32 constA;      // Маштабирующий коэффициент
} CONTEXMPY;

// Объявление функций проекта
void initMPY(CONTEXMPY* pCntx);
void runMPY(CONTEXMPY* pCntx);

// Объявление переменных, массивов и т.д.
extern CONTEXMPY* pCntx;        // Указатель на контекстную структуру
extern CONTEXMPY cntx;          // Контекстная структура
extern word16 inpBuff[];        // Массив для входных данных;
extern word16 outBuff[];        // Массив для выходных данных;
```

Обратите внимание, что дополнительно вводиться еще один макрос для обозначения величины масштабирующего коэффициента — *MASHTAB*.

Кроме этого, необходимо выполнить инициализацию нового поля контекстной структуры в функции *initMPY():*

```
// Функция инициализации начального состояния
// разрабатываемого проекта

#include "prjMPY.h"

void initMPY(CONTEXMPY* pCntx) {
    pCntx->pInpBuff = inpBuff;     // Указатель на входной буфер
    pCntx->pOutBuff = outBuff;     // Указатель на выходной буфер
    pCntx->lenBuff  = SAZEBUFF;    // Размером входного и выходного буферов
    pCntx->constA   = MASHTAB;     // Маштабирующий коэффициент
}
```

Процедура поэлементного умножения отсчетов входного буфера и записи результата в выходной массив может быть выполнена при помощи цикла *for()*. Код функции обработки входного буфера имеет вид:

```
// Функция обработки входного буфера и
// запись результата в выходной буфер

#include "prjMPY.h"

void runMPY(CONTEXMPY* pCntx) {
    // Объявление локальных переменных
    word16 *pInpBuff;   // Указатель на входной буфер
    word16 *pOutBuff;   // Указатель на выходной буфер
    word16 lenBuff;     // Размером входного и выходного буферов
    word32 constA;      // Маштабирующий коэффициент
    word32 count;       // Переменная цикла
    word16 coeff;       // Переменная для хранение промежуточных результатов

    // Инициализация локальных переменных
    pInpBuff = pCntx->pInpBuff;
    pOutBuff = pCntx->pOutBuff;
    lenBuff  = pCntx->lenBuff;
    constA   = pCntx->constA;

    // Цикл обработки и записи
    for(count = 0; count < lenBuff; count++){

        // Чтение текущего отсчета
        coeff = *pInpBuff++;

        // Чтение соответствующего коэффициента фильтра
        coeff *= constA;

        // Запсь текущего отсчета
        *pOutBuff++ = coeff;
    }
}
```

Вначале создаются локальные переменные. Первые четыре соответствуют четырем полям контекстной структуры. Чтобы аналогия была более заметна, они имеют те же имена, что и соответствующие им поля, однако, это не является обязательным. Назначение двух других переменных достаточно очевидно из комментариев — одна переменная является счетчиком количества раз выполнения цикла, вторая — предназначенная для временного хранения результатов вычислений.

Затем производиться инициализация первых четырех локальных переменных соответствующими значениями из контекстной структуры.

Цикл *for()* имеет три параметра разделенных «;». Первый — это начальное значение счетчика циклов, второй — определяет максимальное значение счетчика, а третий показывает способ инкрементации счетчика (в данном случае счетчик увеличивается на единицу на каждом шаге цикла).

В самом цикле первое выражение определяет порядок считывания данных из входного буфера. Префикс «*» перед указателем говорит о том, что необходимо считать данные из ячейки памяти, адрес которой и определяется значением указателя *pInpBuff*. Суффикс «++» означает необходимость увеличения значения адреса на 1 после завершения операции считывания, т.е. значение *pInpBuff* должно стать таким, что бы указывать на следующий отсчет во входном буфере. Таким образом, на каждом шаге цикла данные последовательно считываются из ячеек входного буфера и их значение присваивается переменной *coeff*.

Второе выражение цикла производит масштабирование переменной *coeff*. Это сокращенная форма записи. Полный вариант выглядит следующим образом:

$$coeff = coeff * constA \ .$$

Третье выражение производит операцию обратную считыванию данных. Она производит запись значения переменной *coeff* в буфер, адрес которого определяется значением указателя *pOutBuff*. Причем, после выполнения операции записи происходит инкрементация указателя на 1 (аналогично как и для указателя *pInpBuff*).

3.15. Отладить код функции код функции

К настоящему моменту программный код полностью соответствует разработанному в разделе 2 алгоритму. Процесс компиляции проекта и запуска его на выполнения осуществляется путем нажатия зеленого треугольника на панели инструментов компилятора *wvDev-C++,* что соответствует команде «*скомпилировать и выполнить*» (отметим, что данная команда может также быть вызвана выбором в главном меню пункта «*Выполнить->Скомпилировать и выполнить»,* а так же нажатием функциональной клавиши *F*9). При таком способе никак не отражается процесс выполнения программного кода. Точнее исполнение кода происходит настолько быстро, что это трудно заметить. Необходимо ввести в код функции *main()* дополнительные операторы, которые остановят на время выполнение программы. Окончательный вариант функции *main()* будет иметь вид:

```
#include "prjMPY.h"

int main(void) {
    // Обьявление переменных
    FILE *pInpFile;   // Указатель на входной файл
    FILE *pOutFile;   // Указатель на выходной файл
    word32 len;       // Переменная цикла

    // Вывод сообщения о старте программы
    printf("START PROGRAMS!\n");

    // Открыть входной файл файлов для чтения
    pInpFile = fopen("inp.dat", "rb");

    // Проверить корректность открытия входного файла
    if(pInpFile == NULL) {
        // Вывод сообщение об ошибке
        printf("ERROR, INPUT FILE!\n");

        // Ожидаем нажатия на любую клавишу.
        printf("\nPress ENTER to stop... \n");
        getchar();

        // Выход из программы
        return 2;
    }
    printf("OPEN INPUT FILE SUCSESFUL!\n");

    // Открыть выходной файл файлов для записи
    pOutFile = fopen("out.dat", "wb");

    // Проверить корректность открытия выходного файла
    if(pOutFile == NULL) {
        // Вывод сообщение об ошибке
        printf("ERROR, OUT FILE\n");

        // Ожидаем нажатия на любую клавишу.
        printf("\nPress ENTER to stop... \n");
        getchar();

        // Выход из программы
        return 1;
    }
    printf("GREAT OUTPUT FILE SUCSESFUL!\n");

    // Инициализация указателя на контекстную структуру
    pCntx = &cntx;

    // Инициализация начального состояния
    initMPY(pCntx);

    // Инициализировать переменную цикла
    len = SAZEBUFF;

    // Цикл обработки входного буфера
    while(len == SAZEBUFF) {
        // Считать данные из файла
        len = fread(pCntx->pInpBuff, sizeof(word16), SAZEBUFF, pInpFile);

        // Вызвать функцию обработки входного буфера
        runMPY(pCntx);

        // Записать данные в файл
        fwrite(pCntx->pOutBuff, sizeof(word16), len, pOutFile);
    }

    // Вывод сообщения об окончании выполнения программы
    printf("STOP PROGRAMS!\n");

    // Ожидаем нажатия на любую клавишу.
    printf("\nPress ENTER to stop... \n");
    getchar();

    // Закрыть входной и выходной файлы
    fclose(pInpFile);
    fclose(pOutFile);

    // Выход из программы
    return 0;
}
```

Остановка процесса выполнения программного кода реализована при помощи оператора *getchar()*. Данный оператор ожидает нажатия любой клавиши и выводит ее значение в окно консоли. При чем происходит зацикливание выполнения оператора, программа как бы «зависает» на нем. При нажатии ENTER выполнение данного оператора прекратится и выполнение программы продолжится.

Обратите внимание, что данный оператор вводиться не только в конце программы, но и в операторах *if()*, где анализируется успешность операции открытия файлов. В код функции *main()* дополнительно вставляются операторы *printf()*. Они выводят дополнительные информационные сообщения о ходе выполнения программы.

Откомпилируйте и запустите программу на выполнение, как это делалось раньше, например, нажав клавишу *F*9. Появиться консольное окно с выводом сообщения об ошибки открытия файла (см. рис.22).

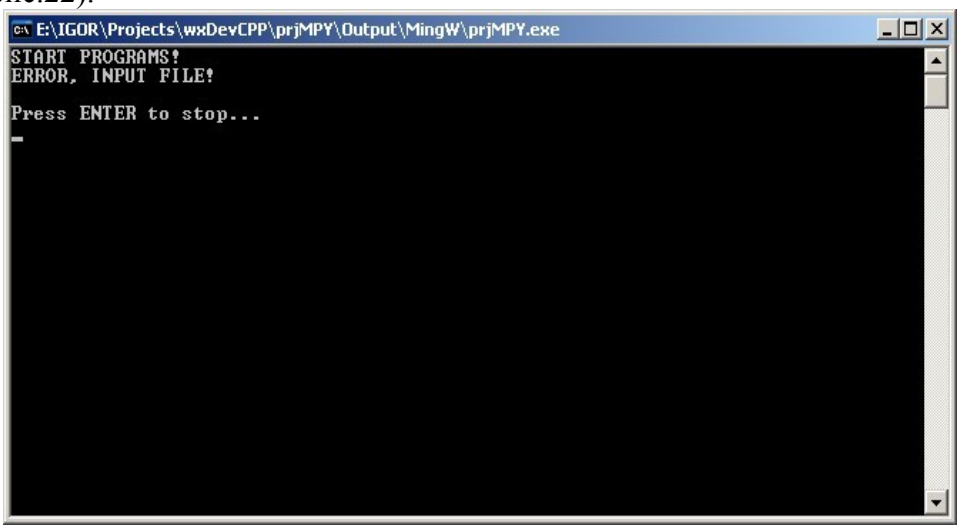

Рис.22. Сообщение об ошибке открытия файла.

Если вы внимательно выполняли рекомендации, то после п.3.8. в папке проекта должен был находиться файл *inp.dat* с отсчетами синусоидального сигнала. Если это не так, создайте такой сигнал и сохраните его в папке проекта и повторите процесс компиляции и запуска. Ошибка не исчезнет.

Исправить данную ситуацию можно запустив программу в режиме отладки. Для этого надо нажать кнопку «*Отладка*» внизу окна программы компилятора *wvDev-C++* (см.рис.23).

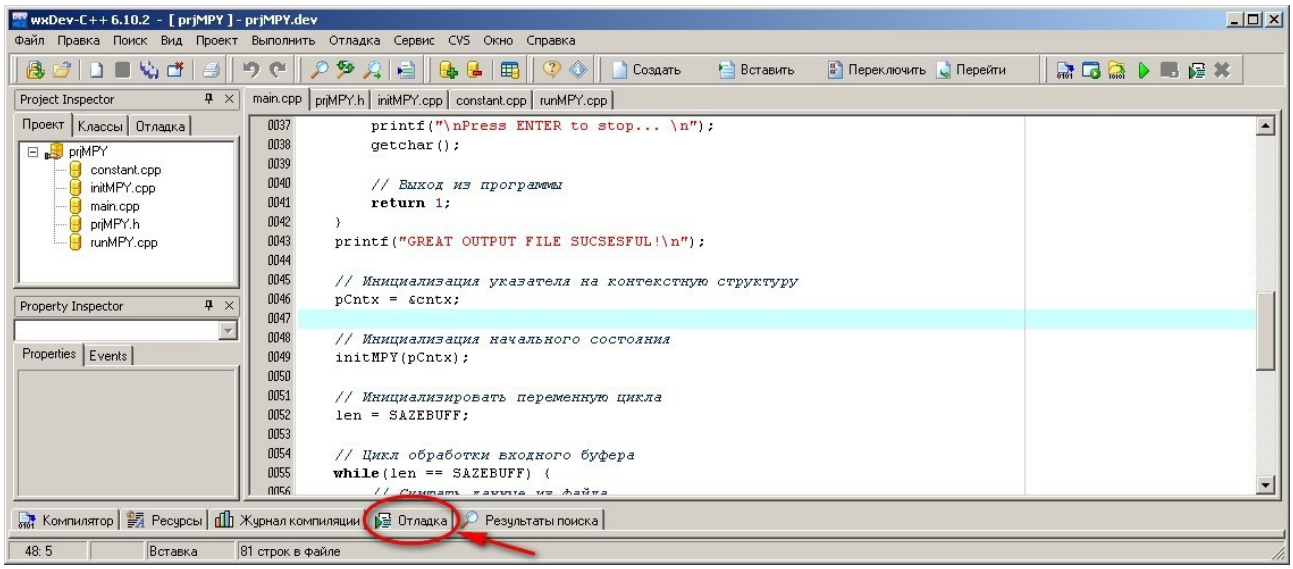

Рис.23. Запуск программы в режиме отладки.

Если запуск в режиме отладки производиться первый раз, появиться окно запроса, показанное на рис.24. Необходимо подтвердить согласие, нажав кнопку «*Yes*».

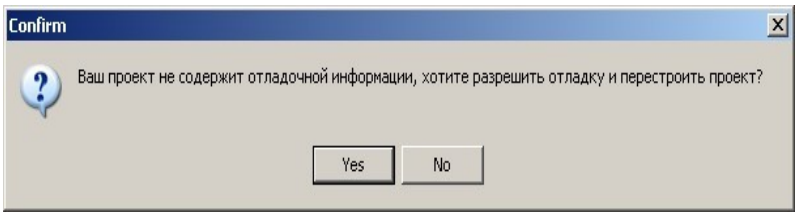

Рис.24. Запрос на запуск режима отладки.

В компиляторе *wvDev-C++* появиться дополнительное информационное окно. Запуск программы в режиме отладки осуществляется нажатием кнопки «*Отладка*» в этом окне (см.рис.25).

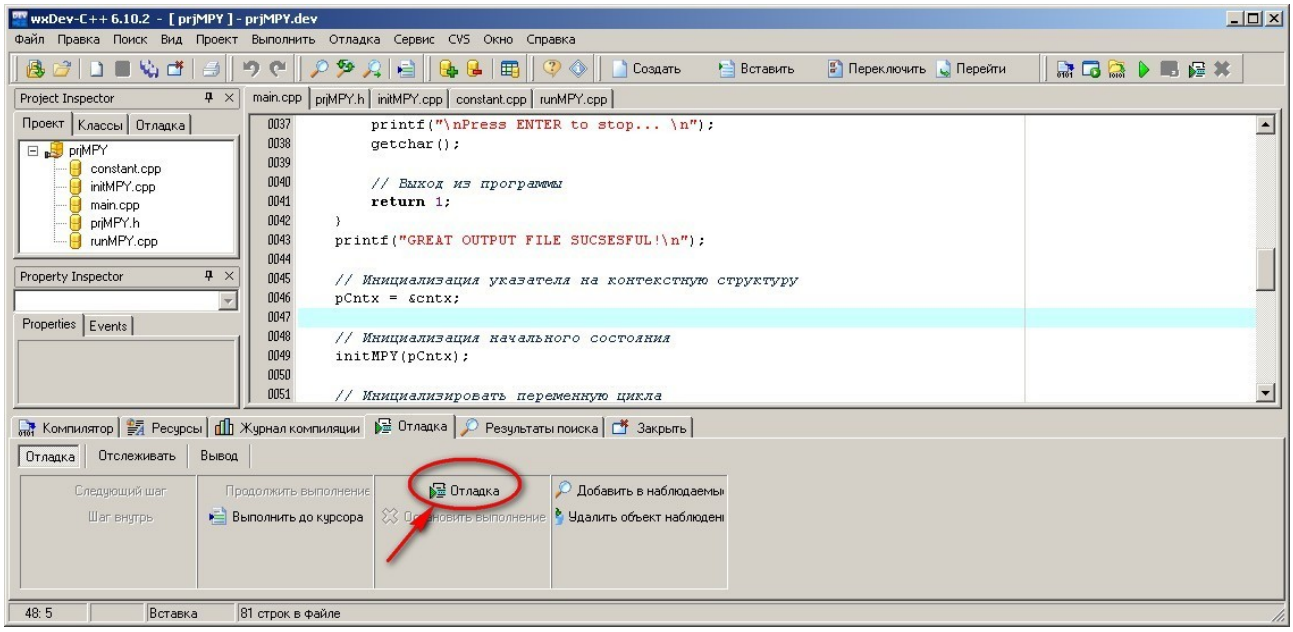

Рис.25. Компилятор wvDev-C++ в режиме отладки.

Появиться консольное окно (см.рис.26), где видны контрольные сообщения о ходе выполнения программы (выводимые функциями *printf()*).

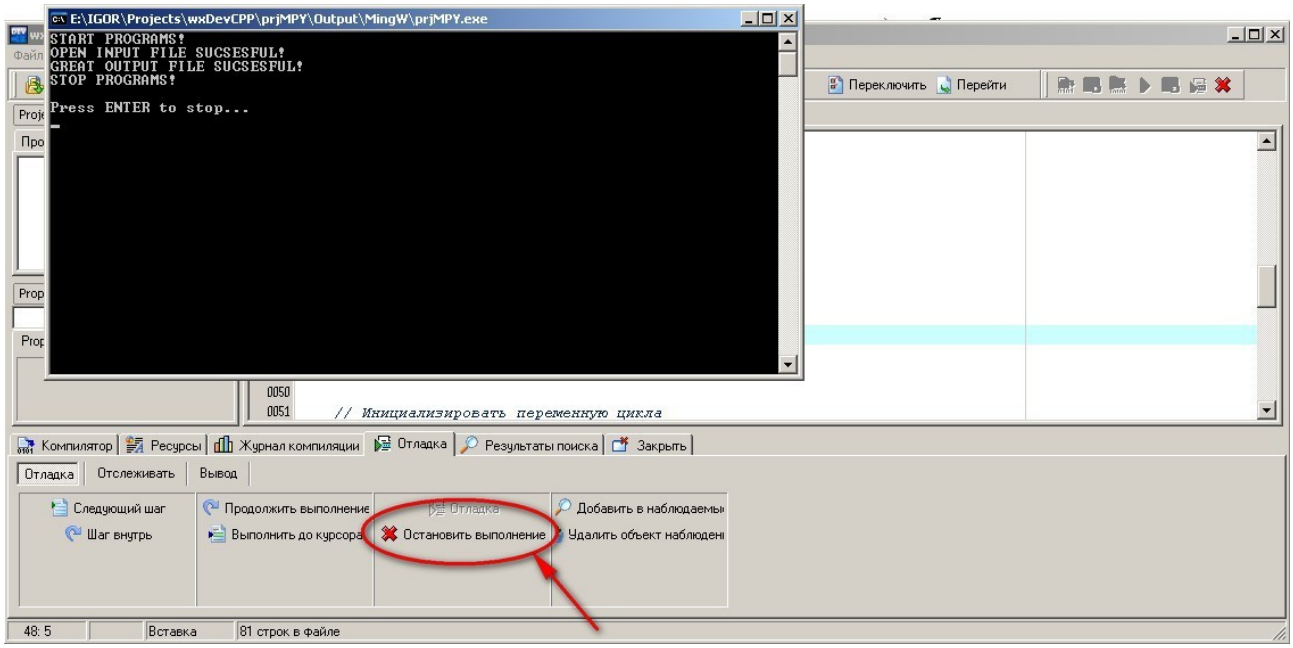

Рис.26. Выход из режима отладки.

Выход из режима отладки может быть осуществлен нажатием клавиши ENTER, если активно консольное окно, или кнопки «*Остановить выполнение*» в окне компилятора (см.рис.26). Второй способ наиболее универсален.

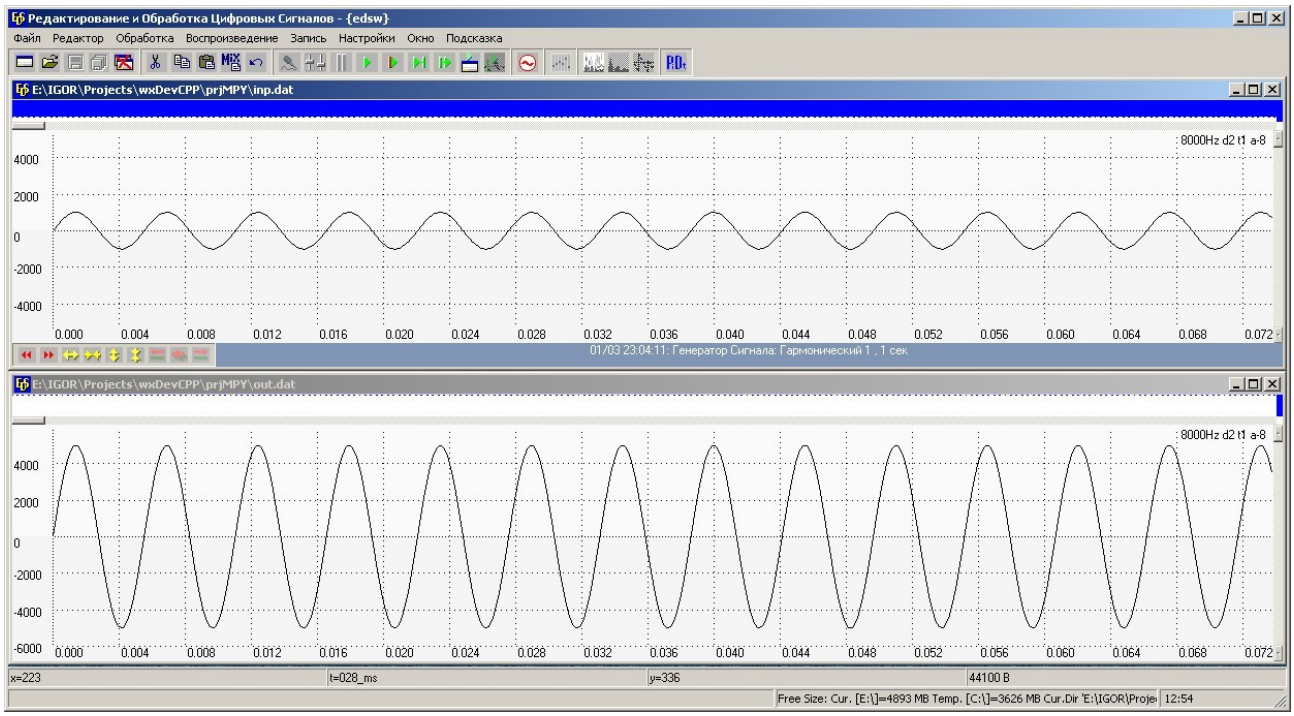

Рис.27. Анализ результатов выполнения программы в EDSW.

Если все выполнено правильно, компиляция и запуск программы на выполнение должно пройти без ошибок и в папке проекта появиться файл *out.dat* с результатом работы программы. Открыв входной и выходной файлы (*int.dat* и *out.dat*, соответственно) в программе EDSW можно убедиться в корректной работе разработанного кода (см.рис.27) — масштабирование входного сигнала в 5 раз.

3.16. Закрытие и повторный запуск проекта

После того, как закончен сеанс работы с проектом, его можно закрыть нажав значок ⊠ в правом верхнем углу окна программы *wxDev-C++*. Появиться запрос на сохранение изменений в проекте, как это показано на рис.28. Нажмите кнопку «Yes» для подтверждения.

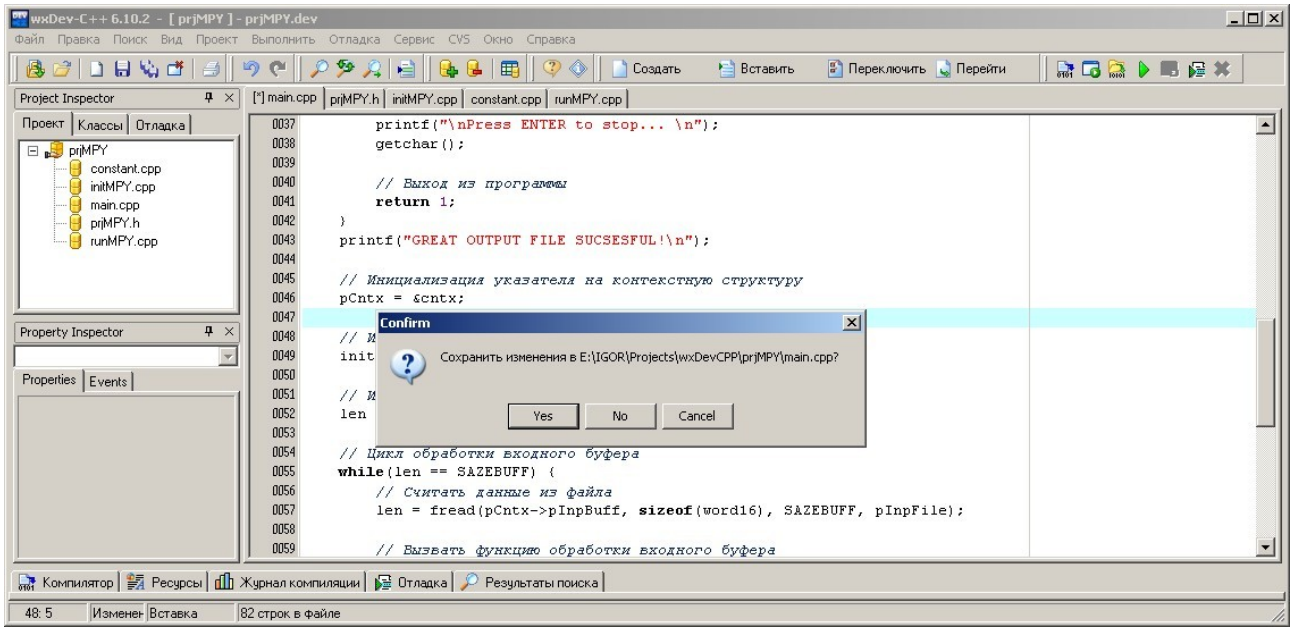

Рис.28. Закрытие программы wxDev-C++.

В дальнейшем, когда появиться необходимость продолжить работу с проектом, необходимо запустить программу *wxDev-C++* и открыть проект. Для этого в главном меню выбрать пункт «*Файл->Открыть проект или файл*», как это показано на рис.29.

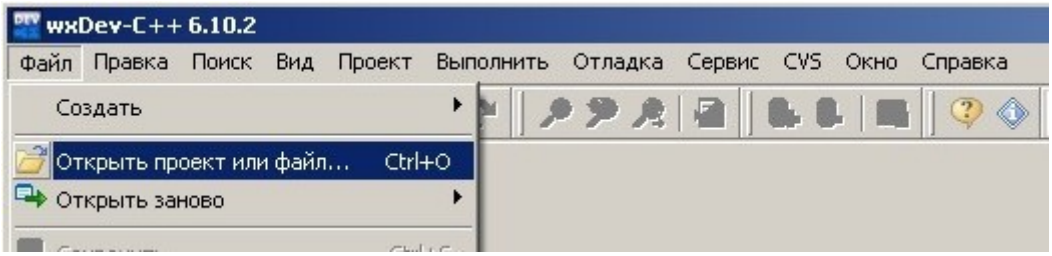

Рис.29. Запуск проекта.

В появившемся окне указать тип открываемого файла «*DevC++ project*»(см.рис.30).

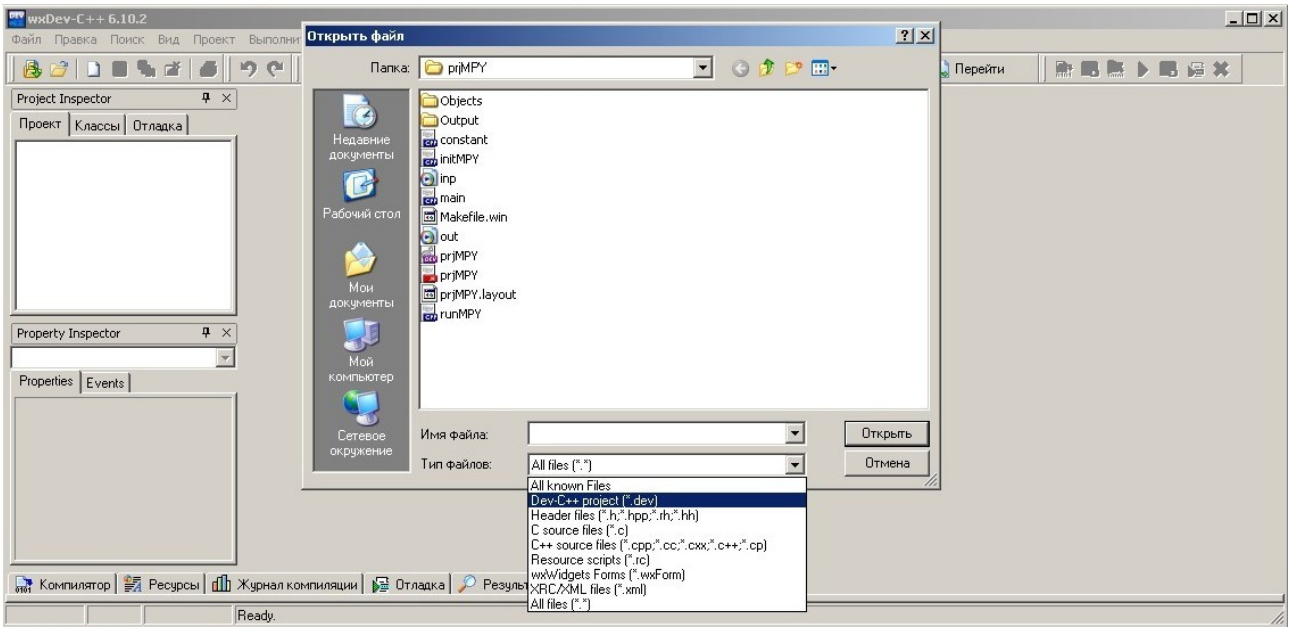

Рис.30. Выбор типа файла проекта для запуска.

Выбрать соответствующий файл (он должен остаться только один — см.рис.31) и запустить проект.

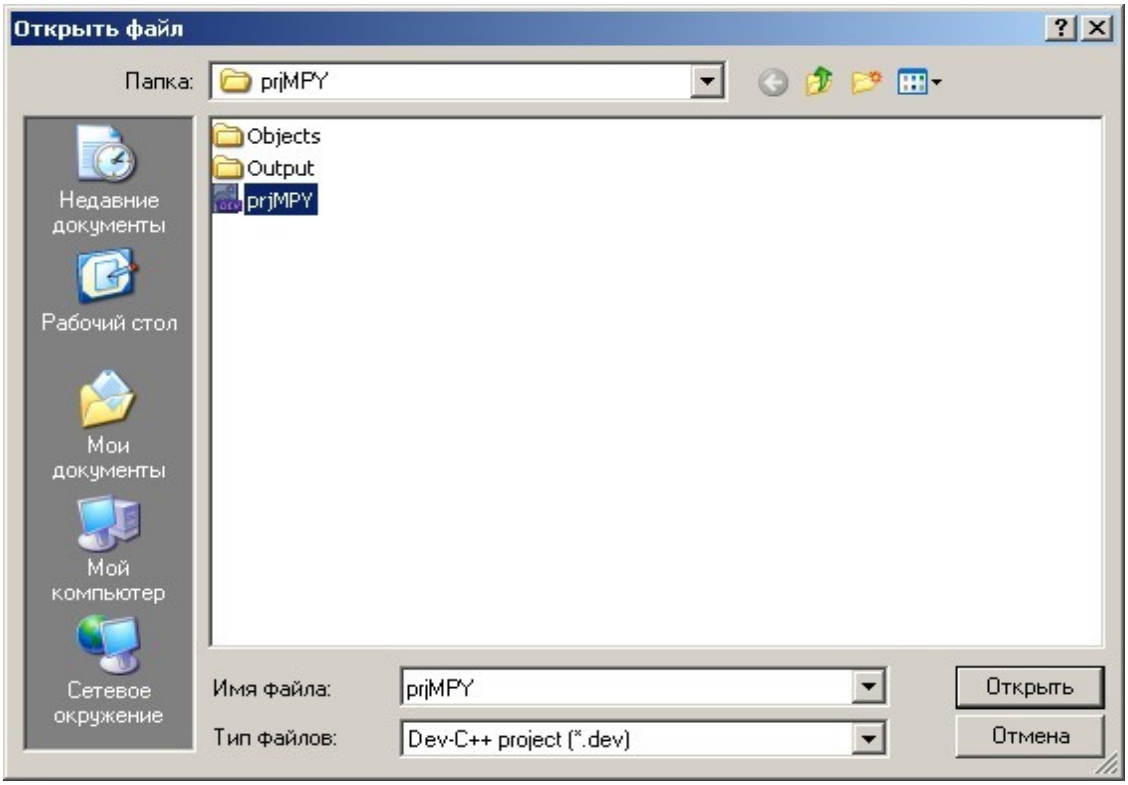

Рис.31. Выбор файла проекта для запуска.

После этого в программе *wxDev-C++* запуститься проект и можно приступить к дальнейшей работе с ним (см.рис.32).

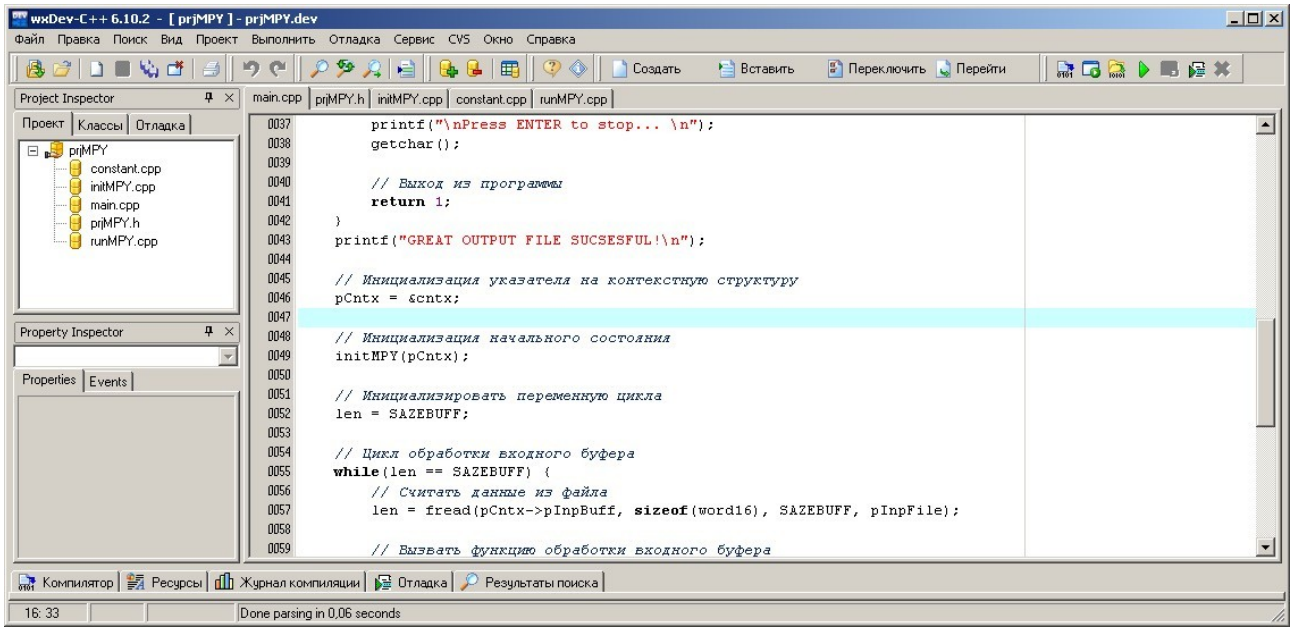

Рис.32. Исходное состояние программы DevC++ после запуска проекта.

Замечание. Иногда случаться, что после запуска проекта в «*Project Inspector*» на закладке «*Проект*» не все файлы проекта отображаются. В таком случае их необходимо подключить заново. Это можно сделать выбрав в главном меню пункт «*Проект->Добавить к проекту*»(см.рис.33).

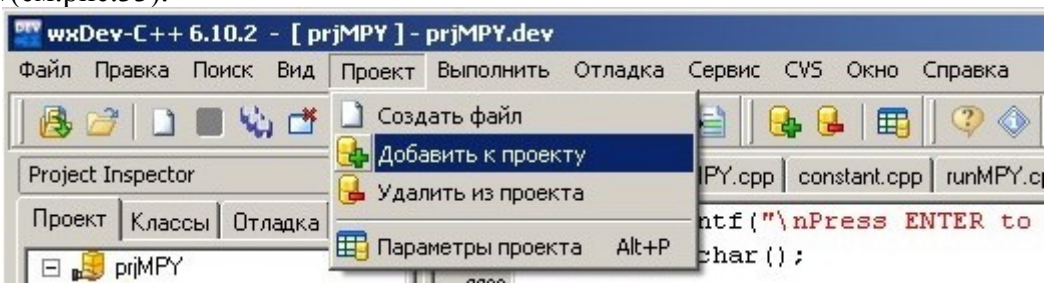

Рис.33. Добавление файлов к проекта.

В появившемся окне указать тип файла «*C++ source files*» и выбрать те файлы, что не отображаются в проекте (см.рис.34).

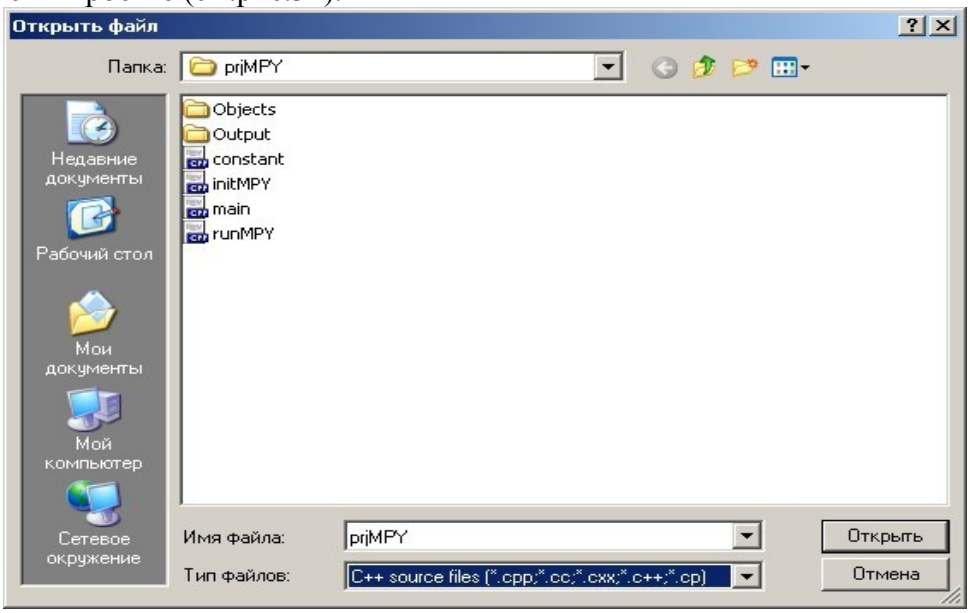

Рис.34. Выбор добавляемых к проектов файлов.

4. Приложения

Приложение 1.

Листинг файла *prjMPY.h*

```
// Подключение внешних библиотек
#include <stdio.h>

// Определение макросов
#define SAZEBUFF    120  // Размер входного и выходного буферов
#define MASHTAB     5    // Маштабирующий коэффициент

// Определение оригинальных имен стандартных типов
typedef int   word32;
typedef short word16;
typedef char  word8;

// Определение типа контекстной структуры
typedef struct {
    word16 *pInpBuff;   // Указатель на входной буфер
    word16 *pOutBuff;   // Указатель на выходной буфер
    word16 lenBuff;     // Размером входного и выходного буферов
    word32 constA;      // Маштабирующий коэффициент
} CONTEXMPY;

// Объявление функций проекта
void initMPY(CONTEXMPY* pCntx);
void runMPY(CONTEXMPY* pCntx);

// Объявление переменных, массивов и т.д.
extern CONTEXMPY* pCntx;        // Указатель на контекстную структуру
extern CONTEXMPY cntx;          // Контекстная структура
extern word16 inpBuff[];        // Массив для входных данных;
extern word16 outBuff[];        // Массив для выходных данных;
```

Приложение 2.

Листинг файла *constantrj.cpp*

```
#include "prjMPY.h"

// Создание указателя на контекстную структуру
CONTEXMPY* pCntx;

// Создание контекстной структуры
CONTEXMPY cntx;

// Массив для входных данных
word16 inpBuff[SAZEBUFF];

// Массив для выходных данных
word16 outBuff[SAZEBUFF];
```

Приложение 3.

Листинг функции *initMPY()*

```
// Функция инициализации начального состояния
// разрабатываемого проекта

#include "prjMPY.h"

void initMPY(CONTEXMPY* pCntx) {
    pCntx->pInpBuff = inpBuff;      // Указатель на входной буфер
    pCntx->pOutBuff = outBuff;      // Указатель на выходной буфер
    pCntx->lenBuff  = SAZEBUFF;     // Размером входного и выходного буферов
    pCntx->constA   = MASHTAB;      // Маштабирующий коэффициент
}
```

Приложение 4.

Листинг функции *runMPY()*

```
// Функция обработки входного буфера и
// запись результата в выходной буфер

#include "prjMPY.h"

void runMPY(CONTEXMPY* pCntx) {
    // Объявление локальных переменных
    word16 *pInpBuff;   // Указатель на входной буфер
    word16 *pOutBuff;   // Указатель на выходной буфер
    word16 lenBuff;     // Размером входного и выходного буферов
    word32 constA;      // Маштабирующий коэффициент
    word32 count;       // Переменная цикла
    word16 coeff;       // Переменная для хранение промежуточных результатов

    // Инициализация локальных переменных
    pInpBuff = pCntx->pInpBuff;
    pOutBuff = pCntx->pOutBuff;
    lenBuff  = pCntx->lenBuff;
    constA   = pCntx->constA;

    // Цикл обработки и записи
    for(count = 0; count < lenBuff; count++){

        // Чтение текущего отсчета
        coeff = *pInpBuff++;

        // Чтение соответствующего коэффициента фильтра
        coeff *= constA;

        // Запсь текущего отсчета
        *pOutBuff++ = coeff;
    }
}
```

Приложение 5.

Листинг функции *main()*

```c
#include "prjMPY.h"

int main(void) {
    // Обьявление переменных
    FILE *pInpFile;  // Указатель на входной файл
    FILE *pOutFile;  // Указатель на выходной файл
    word32 len;      // Переменная цикла

    // Вывод сообщения о старте программы
    printf("START PROGRAMS!\n");

    // Открыть входной файл файлов для чтения
    pInpFile = fopen("inp.dat", "rb");

    // Проверить корректность открытия входного файла
    if(pInpFile == NULL) {
        // Вывод сообщение об ошибке
        printf("ERROR, INPUT FILE!\n");

        // Ожидаем нажатия на любую клавишу.
        printf("\nPress ENTER to stop... \n");
        getchar();

        // Выход из программы
        return 2;
    }
    printf("OPEN INPUT FILE SUCSESFUL!\n");

    // Открыть выходной файл файлов для записи
    pOutFile = fopen("out.dat", "wb");

    // Проверить корректность открытия выходного файла
    if(pOutFile == NULL) {
        // Вывод сообщение об ошибке
        printf("ERROR, OUT FILE\n");

        // Ожидаем нажатия на любую клавишу.
        printf("\nPress ENTER to stop... \n");
        getchar();

        // Выход из программы
        return 1;
    }
    printf("GREAT OUTPUT FILE SUCSESFUL!\n");

    // Инициализация указателя на контекстную структуру
    pCntx = &cntx;

    // Инициализация начального состояния
    initMPY(pCntx);

    // Инициализировать переменную цикла
    len = SAZEBUFF;

    // Цикл обработки входного буфера
    while(len == SAZEBUFF) {
        // Считать данные из файла
        len = fread(pCntx->pInpBuff, sizeof(word16), SAZEBUFF, pInpFile);

        // Вызвать функцию обработки входного буфера
        runMPY(pCntx);

        // Записать данные в файл
        fwrite(pCntx->pOutBuff, sizeof(word16), len, pOutFile);
    }

    // Вывод сообщения об окончании выполнения программы
    printf("STOP PROGRAMS!\n");

    // Ожидаем нажатия на любую клавишу.
    printf("\nPress ENTER to stop... \n");
    getchar();

    // Закрыть входной и выходной файлы
    fclose(pInpFile);
    fclose(pOutFile);

    // Выход из программы
    return 0;
}
```