Лабораторная работа №3. Функции, запись бинарных файлов.

1.1 Задание для работы

Для каждого простого сигнала из лабораторной работы №2 написать функцию, которая будет возвращать объект std::vector с отсчетами сигнала.

Создать std::vector и объединить в нем четыре простых сигнала.

Записать данный вектор в бинарный файл формата. adc

Открыть файл в имитационной среде Спектр-2 и построить осциллограмму.

1.2 Теоретические сведения

1.2.1 Функции в языке C++

Функция - это именованная последовательность описаний и операторов, обычно предназначенная для решения определенной задачи. Функция может принимать параметры и возвращать значение.

Любая программа на C++ состоит из функций, одна из которых должна иметь имя main. С функции ша1пначинается выполнение программы. Выполнение функции начинается в момент ее вызова. Как и переменная, функция должна быть объявлена и определена, при этом объявление функции должно находиться в тексте программы раньше ее вызова.

При объявлении функции указывается ее имя, тип возвращаемого значения и список параметров, передаваемых в нее:

```
int sum(int a, int b); //объявление функции
```

Определение функции содержит, кроме объявления, тело функции, представляющее собой последовательность описаний и операторов в фигурных скобках:

```
int sum(int a, int b) { //определение функции return (a+b);
}
```

В объявлении, определении и при вызове одной и той же функции типы и порядок следования параметров должны совпадать. На имена параметров ограничения не накладываются.

Для того, чтобы вызвать функцию, необходимо указать ее имя, после которого в круглых скобках через запятую перечисляются имена передаваемых в функцию аргументов:

```
#include <iostream>
int sum(int a, int b); //объявление функции

int main()
```

```
{
    int a = 2, b = 3, c, d;
    c = sum(a, b);  /*вызов функции и передача в качестве входных
    параметров значений переменных a и b*/
    cin >> d;
    cout << sum(c, d); /*вызов функции и передача в качестве входных параметров значений
переменных c и d*/
    return 0;
       }
```

Все величины, описанные в функции, и ее параметры являются локальными, областью их действия является функция. При выходе из функции значения локальных переменных не сохраняются, так как участок стека освобождается. Чтобы этого избежать используется параметр static.

При совместной работе функции должны обмениваться информацией. Это можно осуществить с помощью:

- глобальных переменных;
- через параметры;
- через возвращаемое функцией значение.

Возврат из функции в вызвавшую ее функцию реализуется оператором return:

```
return [выражение];
```

Функция может содержать несколько операторов return, однако ее работа прекращается при выполнении первого возможного оператора return. Если функция описана как void, выражение не указывается. Оператор return можно опускать для функций типа void, если возврат из неё происходит перед фигурной скобкой, и для функции main.

```
int function_1
{
    return 1;
}

double function_2
{
    return 1; //1 преобразуется к типу double
}
```

ВАЖНО: Нельзя возвращать из функции указатель на локальные переменные, поскольку память, выделенная локальным переменным при входе в функцию, освобождается после возврата из неё.

1.2.2 Параметры функции

В функциях есть два типа параметров: формальные и фактические. Формальные параметры (параметры) - это параметры, перечисленные в заголовке описания функции, их еще называют просто параметрами. Фактические параметры (аргументы) - это параметры записанные в операторе вызова функции.

Существует два способа передачи параметров функции:

- по значению;
- по адресу:
  - по указателю;
  - по ссылке.

При передаче параметров по значению в память заносятся копии значений аргументов, и операторы функции работают с этими копиями. Доступа к исходным значениям параметров у функции нет, а, следовательно, нет возможности их изменить.

При передаче по адресу в память заносятся копии адресов аргументов, функция осуществляет доступ к ячейкам памяти по этим адресам и может изменить исходные значения аргументов.

```
#include <iostream>
void f(int i, int* j, int& k);

int main()
{
    int i = 1, j = 2, k = 3; cout << "i j k\n";
    cout << i << '   ' << j << '   ' << k << '\n';
    f(i, &j, k);
    cout << i << '   ' << j << '   ' << k;
    return 0;
}

void f(int i, int* j, int& k)
{
    i++;
    (*j)++;
    k++;
}
```

В примере выше параметр i передается по значению. Его изменение функцией не повлияет на исходное значение. Параметр j передается по адресу с использованием указателя, при этом для передачи в функцию адреса фактического параметра используется операция взятия адреса, а для получения его значения функции требуется операция разыменования.

Параметр k передается по адресу с помощью ссылки. При передаче по ссылке в функцию передается адрес указанного при вызове параметра, внутри функции все обращения к параметру неявно разыменовываются. При использовании ссылок, а не указателей читаемость программы лучше, также это избавляет от использования операций получения адреса и разыменования.

1.2.3 Контейнер std::vector

В Стандартной библиотеке C++ есть и улучшенная версия динамических массивов (более безопасная и удобная) — std::vector. std::vector идет в комплекте с дополнительными возможностями, которые делают его одним из самых полезных и универсальных инструментов в языке C++.

Представленный в C++03, std::vector (или просто «вектор») — это тот же динамический массив, но который может сам управлять выделенной себе памятью. Это означает, что вы можете создавать массивы, длина которых задается во время выполнения, без использования операторов new и delete (явного указания выделения и освобождения памяти). std::vector находится в заголовочном файле vector. Объявление std::vector следующее:

```
#include <vector>

// Нет необходимости указывать длину при инициализации
std::vector<int> array;
std::vector<int> array2 = { 10, 8, 6, 4, 2, 1 }; // используется список инициализаторов для инициализации массива
std::vector<int> array3{ 10, 8, 6, 4, 2, 1 }; // используется uniform-инициализация для инициализации массива (начиная с C++11)
```

Обратите внимание, что в неинициализированном, что в инициализированном случаях вам не нужно явно указывать длину массивов. Это связано с тем, что std::vector динамически выделяет память для своего содержимого по запросу.

Доступ к элементам массива может выполняться как через оператор [] (который не выполняет проверку диапазона), так и через функцию at() (которая выполняет проверку диапазона):

```
array[7] = 3; // без проверки диапазона
array.at(8) = 4; // с проверкой диапазона
```

В отличие от стандартных динамических массивов, которые не знают свою длину, std::vector свою длину запоминает. Чтобы её узнать, нужно использовать функцию size():

```
#include <vector>
#include <iostream>

int main()
{
    std::vector<int> array{ 12, 10, 8, 6, 4, 2, 1 };
    std::cout << "Длина: " << array.size() << '\n';

    return 0;
}
```

Изменить длину стандартного динамически выделенного массива довольно проблематично и сложно. Изменить длину std::vector так же просто, как вызвать функцию resize():

```
#include <vector>
#include <iostream>
```

```
int main()
{
    std::vector<int> array{ 0, 1, 2 };
    array.resize(7); // изменяем длину array на 7

    std::cout << "Длина: " << array.size() << '\n';

    for (auto const& element : array)
        std::cout << element << ' ';

    return 0;
}
```

1.3 Пример программы

Исходные данные: ***Простой сигнал: SIN, f = 1000 Гц. 64 отсчета. Частота дискретизации 48 кГц.***

```
#define _USE_MATH_DEFINES

#include <iostream>
#include <math.h>
#include <vector>

int main()
{
    std::cout << "Genereted SIN!\n";

    float data[64];
    double freq_sampling = 48000.0;
    double freq_Hz = 1000.0;
    int sample_in_period = freq_sampling / freq_Hz;

    for (int i = 0; i < 64; ++i)
    {
        data[i] = sin(float(2 * M_PI * i * freq_Hz / freq_sampling));
    }

    std::vector<float> data_for_file;
    data_for_file.resize(64);
    for (int i = 0; i < data_for_file.size(); ++i)
        data_for_file[i] = data[i];
}
```

1.4 Содержание отчета

Отчет к лабораторной работе должен содержать:

1. Текст задания, вариант.
2. Листинг программы.
3. Файл с отсчетами сложного сигнала

1.5 Варианты задания

Варианты заданий используются из лабораторной работы №2

1.6 Приложение А Формат файла .adc

Формат файлов .adc (АЦП) представляет собой бинарный файл содержащий в себе заголовок и цифровые квадратурные отсчеты.

Бинарное представление
Заголовок - 32 байта:
Период дискретизации.. double(QWORD) - 8 байт
Разрядность АЦП..... long (DWORD) - 4 байта
количество отсчетов... long (DWORD) - 4 байта
максимальное значе-... float (DWORD) - 4 байта
ние отсчета (для формата с плавающей точкой,
если неизвестно, то заполнить нулями)
флаги..... long (DWORD) - 4 байта
младший бит:
целочисленные отсчеты - 00000000H
отсчеты с плавающей точкой - 00000001H
первый бит:
отсчеты комплексные - 00000002H
отсчеты некомплексные - 00000000H
резерв заголовка..... - 8 байт
желательно заполнить нулями
отсчеты - (массив переменной длины)
Некомплексные отсчеты:
целочисленные - **short int*** - 2N байт
с плавающей точкой - **float*** - 4N байт
Комплексные отсчеты:
Записаны подряд пары вида (реальная,мнимая) части
целочисленные - **short int*** - 4N байт
если разрядность АЦП выше 16, то целочисленные отсчеты
представлены следующим образом:
- **int*** - 8N байт
с плавающей точкой - **float*** - 8N байт

Отсчеты с плавающей точкой. Поле "Разрядность АЦП" игнорируется.
Для надежности устройство сохранения файла АЦП задает это значение в 32 бита (это размер, а не эффективная разрядность). Для действительных сигналов - выравнивание 4 байта, для комплексный - 8 байт.
Все программы, умеющие читать файл АЦП, как правило, поле "Разрядность АЦП" для отсчетов с плавающей точкой (**float**) игнорируют.

Поле "Разрядность АЦП" необходимо для определения уровня сигнала. Например, разрядность АЦП 16-ти битных отсчетов может быть 12, 14, 16. Максимальный уровень сигнала в приведенном примере может быть 2^-11, 2^-13 или 2^-15. При условии, что старший бит - знаковый.

Рисунок 1 Описание формата файла АЦП

Используя C++ заголовок файла можно представить следующей структурой:

```
//! Структура заголовка файла АЦП (ADC)
```

```
struct ADCFILEHEADER {
    double           dt; //!< Период дискретизации в секундах
    unsigned long     h; //!< Разрядность АЦП
    unsigned long    fN; //!< Общее количество отсчетов в файле
    float            Am; //!< Амплитуда, соответствующая максимальному значению
целочисленного отсчета
    unsigned long flags; //!< Флаги
    char      reserv[8]; //!< Резерв
};
typedef struct ADCFILEHEADER* LPADCFILEHEADER; //!< Указатель на ADCFILEHEADER

#define ADCF_DATA_INTEGER 0x00000000 //!< Тип данных - целочисленные
#define ADCF_DATA_FLOAT   0x00000001 //!< Тип данных - float
#define ADCF_TYPE_REAL    0x00000000 //!< Тип отсчетов - действительные
#define ADCF_TYPE_COMPLEX 0x00000002 //!< Тип отсчетов – комплексные
```

Размер данной структуры 32 байта, и она всегда находится вначале файла.

Ниже представлена функция, которая производит запись в файл вектора с действительными отсчетами.

```
bool write_adc(std::vector<float> data, double freq_sampling)
{
    ADCFILEHEADER head;
    head.Am = 1.0;
    head.h = 32;
    head.dt = 1.0 / freq_sampling;
    head.fN = data.size();
    head.flags = ADCF_DATA_FLOAT;

    const char* FName = "1.adc";       //Путь к файлу

    float im = 0.0;

    /*РАБОТА С БИНАРНЫМ ФАЙЛОМ*/
    ofstream out(FName, ios::binary);            //Ставим режим "бинарный файл"
    out.write((char*)&head, sizeof(head));        //Записываем в файл структуру заголовка
    for (int i = 0; i < data.size(); ++i)
    {
        out.write((char*)&data[i], sizeof(data[i]));   // Записываем действительные отсчеты
из вектора
    }
    out.close();

    return true;
}
```

1.7 Приложение Б Среда имитационного моделирования Спектр-2

Спектр-2 с достаточно полным набором встроенных устройств позволяет моделировать различные радиотехнические системы, генерировать сигналы разных видов, исследовать прохождение сигналов через собранную пользователем систему. Можно наглядно оценить влияние параметров устройств системы на ее работу. Предусмотрена возможность изменять параметры устройств во время работы системы.

Для запуска необходимо запустить исполняемый файл Specr2.exe

Рисунок 2 Исполняемый файл приложения

После запуска системы, в окне приветствия необходимо выбрать пункт «Собрать радиотехническую систему» либо «Файл» - «Собрать систему»

Рисунок 3 Окно приветствия

В окне «Параметры системы» указать частоту дискретизации 48000 и размер пачки БПФ 256

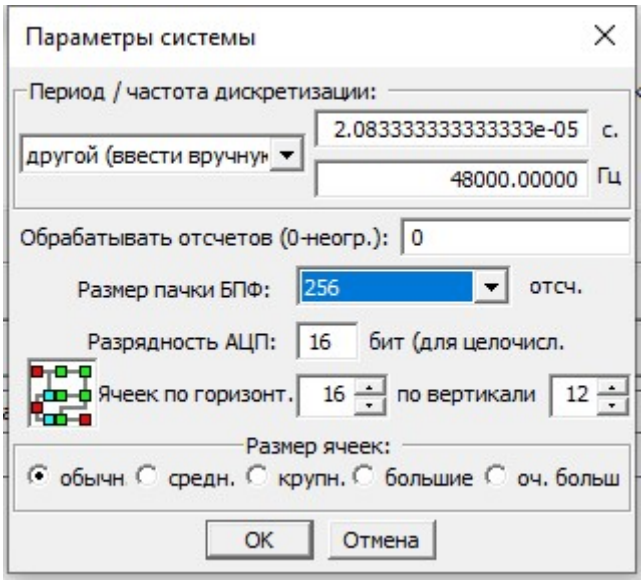

Рисунок 4 Параметры системы

В загрузившемся окне «Редактор схем» справа в дереве устройств найти устройство «Входной файл АЦП» либо указать название в поиске

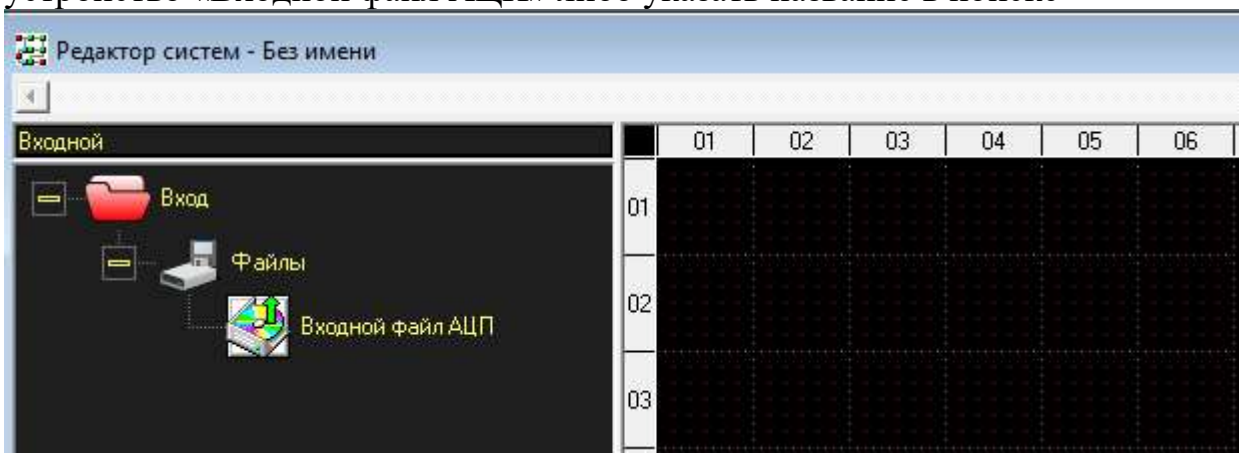

Рисунок 5 Поиск устройств в Дереве устройств

Затем вытащить данной устройство левой кнопкой мыши на поле. После этого откроется диалог «Входной файл АЦП», где выбрать нужный файл .adc. и нажать «Открыть».

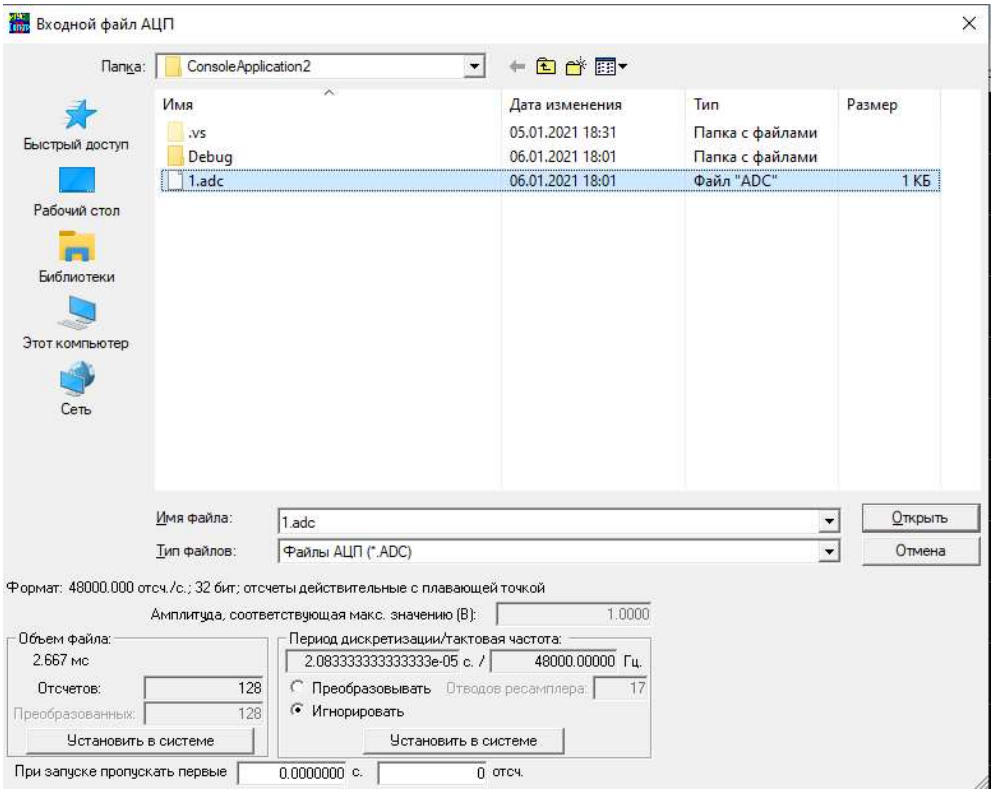

Рисунок 6 Открытие файла АЦП

После этого нажать на свободную ячейку в «Редакторе схем» правой кнопкой мыши и в контекстном меню выбрать «Осциллограмма». Нажать «ОК». После этого левой кнопкой мыши соединить 2 элемента на поле (кубика).

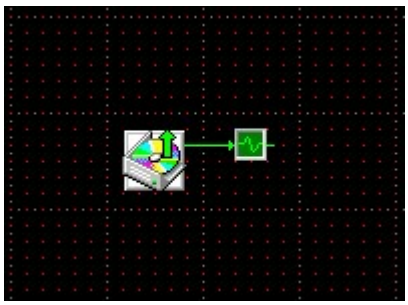

Рисунок 7 Соединение 2х элементов: Входной файл АЦП и Осциллограмма

В верхнем тулбаре нажать кнопку «Осциллограммы», на рисунке цифра 1, откроется окно. После этого нажать кнопку «Обработка очередной пачки», на рисунке цифра 2, или нажать F7.

Рисунок 8 Верхний тулбар программы

В окне «Осциллограф» построится данные из файла.

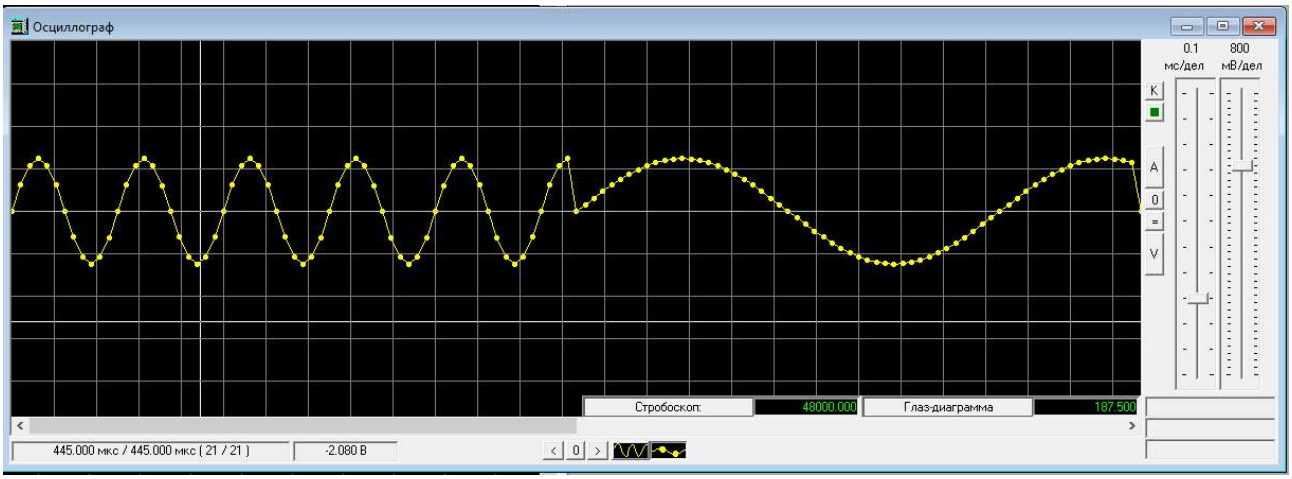

Рисунок 9 Окно «Осциллограф»

Дополнительную информацию можно получить из Справки среды имитационного моделирования нажав F1.