1. Лабораторная работа №2. Генерация сложных сигналов

1.1. Задание для работы

Составить программу для генерации сложного сигнала, состоящего из четырех частей, каждая длиной 64 отсчета.

Записать в текстовый файл отсчеты сложного сигнала (256 отсчетов).

1.2. Теоретические сведения

Операторы сравнения предназначены для сравнения между собой двух значений. В таблице 2.2.1 представлены операторы сравнения языка C++.

Таблица 2.2.1. Операторы сравнения

Символ операции	Значение	Использование
<	меньше	**a < b**
<=	меньше либо равно	**a <= b**
>	больше	**a > b**
>=	больше либо равно	**a >= b**
==	равно	**a == b**
! =	не равно	**a != b**

В языке C++ существует четыре основных логических операции:

- логическая операция И;
- логическая операция ИЛИ;
- логическая операция НЕ или логическое отрицание;
- логическая операция ИСКЛЮЧАЮЩЕЕ ИЛИ.

Логические операции образуют сложное (составное) условие из нескольких простых (два или более) условий. Пример условий показан в таблице 2.2.2.

Таблица 2.2.2. Логические операции языка С++

Операции	Обозначение	Условие	Краткое описание
И	`&&`	`a == 3 && b > 4`	Составное условие истинно, если истинны оба простых условия
ИЛИ	`II`	`a == 3 || b > 4`	Составное условие истинно, если истинно, хотя бы одно из простых условий
НЕ	`!`	`!( a == 3)`	Условие истинно, если а не равно 3
ИСКЛЮЧАЮЩЕЕ ИЛИ		`!(a == 3 || b> 4)`	Составное условие ложно, если оба простых условия истинны или ложны.

Операции сравнения и логические операции в результате дают значение типа bool, то есть true или false. Если же такое выражение встречается в контексте, требующем целого значения, true преобразуется в 1, а false - в 0.

1.2.1. Условный оператор if

Условный оператор в языке С++ имеет следующую структуру:

```
if (логическое выражение) оператор_1;
        [else оператор_2; ]
```

Сначала вычисляется значение логического выражения, если оно не равно нулю (true), выполняется первый оператор, иначе - второй. Вторую ветвь оператора вместе с else можно опустить, тогда, в случае, если выражение ложно, произойдет выход из ветвления.

Условный оператор с двумя ветвями:

```
if (num < 10)
{ // если введенное число меньше 10
    cout << "Это число меньше 10." << endl;
}
else { // иначе
    cout << "Это число больше либо равно 10" << endl;
        }
```

Условный оператор с одной ветвью:

```
if (num != 10) // если введенное число не равно 10
 cout << "Это число не равно 10." << endl;
```

Распространенные ошибки:

1. Использование в выражении при проверке равенства вместо оператора (==) простое присваивание (=).
2. Неверная запись проверки на принадлежность диапазону. Условие 0

< x < 1 необходимо записать следующим образом:

```
if (0 < x && x < 1)
```

1.2.2. Операторы цикла

Операторы цикла используются для организации многократно повторяющихся вычислений. В языке C++ существует три типа циклических конструкций:

- цикл с предусловием (while);
- цикл с постусловием (do while);
- цикл с параметром (с заданным количеством повторений (for)).

Выполнение цикла с предусловием начинается с условия, если оно истинно, выполняется оператор цикла. Если при первой проверке условие ложно, то цикл не выполнится ни разу. В общем виде цикл с предусловием приведен ниже:

```
while (условие)
{
    блок инструкций
}
Пример:
int i = 1;    // инициализация счетчика цикла
while (i <= 10)     // выполняем цикл пока i ^ 10
{
    cout << i * i << endl;
    ++i;      // увеличение i на 1
       }
```

Тело цикла с постусловием всегда выполняется хотя бы один раз. Сначала выполняется простой или составной оператор в теле цикла, а затем проверяется условие. Если условие истинно, тело цикла выполняется еще раз. Цикл завершается, когда условие станет ложным или в теле цикла будет выполнен оператор передачи управления.

Цикл с постусловием в общем виде:

```
do
{
    блок инструкций
} while (условие);
Пример:
    int i = 0;       // инициализация счетчика цикла
    int sum = 0;     // инициализация счетчика суммы
    do {      // выполняем цикл
        i++;
        sum += i;
    } while (i < 1000);     // пока выполняется условие
    Цикл с параметром имеет следующий формат :
    for (инициализация; условие; модификации)
    {
        блок инструкций
    }
}
```

В части инициализации можно записать несколько переменных, используемых в цикле, разделенных запятой. Цикл с параметром выполняется как

цикл с предусловием: сначала проверяется истинность условия, а затем выполняется или не выполняется тело цикла.

Пример:

```
for (int i = 0; i <= 10; i++)     /* инициализация счетчика, условие,
увеличение счетчика на 1*/
        cout << i * i << endl; // тело цикла
```

Распространенные ошибки:

1. Использование в теле цикла не инициализированных переменных.
2. Неверная запись условия выхода из цикла.

1.2.3. Массивы на языке C++

Массив - это структура данных, представленная в виде последовательной группы значений одного типа, объединенных под одним именем. Массивы используются для обработки большого количества однотипных данных. Отдельное значение данных массива называется элементом массива. Элементами массива могут быть данные любого типа. Массивы, имеющие одно измерение, называются одномерными.

Для создания массива используется оператор объявления. При объявлении массива необходимо указать тип значений элементов массива, имя массива и его размерность (количество элементов в массиве). Массивы бывают статическими и динамическими. Размерность статического массива указывается при его описании и вместе с его типом определяют объем памяти, необходимый для размещения массива. Размерность статического массива должна быть целочисленной константой или константным выражением, в котором известны все значения на момент компиляции. Объявление одномерного статического массива на языке C++ выглядит следующим образом:

```
        double a[10]; //описание массива из 10 элементов вещественного типа
```

Размерность массивов лучше задавать с помощью именованных констант. При таком подходе для ее изменения размерности достаточно скорректировать значение константы в одной строке программы.

В отличие от статических массивов, у динамических размерность может быть переменной, то есть объем памяти, выделяемый под массив, определяется на этапе выполнения программы.

Элементы массива в языке C++ нумеруются с нуля, соответственно первый элемент массива будет иметь индекс 0, второй - 1, третий - 2 и т.д.

При инициализации статического массива все его элементы указываются по порядку в фигурных скобках при объявлении массива в программе:

```
        int b[3] = { 1, 5, 4 }; // b[0] = 1, b[1] = 5, b[2] = 4, b[3] = 0
```

Если при инициализации в фигурных скобках будет указано меньше элементов, чем размерность массива, то все оставшиеся элементы будут равны 0.

Для того, чтобы присвоить значение отдельному элементу массива, используют индексы (оператор []):

```
b[3] = 5; // b[0] = 1, b[1] = 5, b[2] = 4, b[3] = 0
```

Если необходимо заполнить массив своими значениями с клавиатуры, из

файла или случайными значениями, используются циклы. Пример заполнения одномерного массива случайными числами:

```
#include <iostream> // стандартный поток ввода/вывода
#include <cstdlib> /* заголовочный файл определяет несколько функций общего назначения, в том числе функции генерации случайных чисел*/
#include <ctime> // содержит функции для работы со временем и датой
using namespace std;
int main()
{
    int randomDigits[3] = {};// пустой массив из трех элементов
    srand(time(NULL)); /* устанавливает в качестве базы для генерации случайных чисел текущее время, таким образом, при каждом запуске генерируется новый набор случайных значений*/
    for (int i = 0; i < 3; i++) // цикл для заполнения массива
    {
        randomDigits[i] = rand(); /* функция rand() генерирует случайные числа*/
        cout << randomDigits[i] << endl;
    }
    return 0;
       }
```

В приведенном выше примере массив из 3 элементов заполняется случайными целыми числами.

1.2.4. Работа с текстовыми файлами в C++

Для работы с файлами используются специальные типы данных, называемые потоками. В программах на языке C++ при работе с текстовыми файлами необходимо подключать библиотеку fstream:

```
#include <fstream>
```

Поток ifstream служит для работы с файлами в режиме чтения, а ofstream в режиме записи. Для работы с файлами в режиме, как записи, так и чтения, служит поток fstream.

Для того чтобы записывать данные в текстовый файл, необходимо:

1. Описать переменную типа ofstream.
2. Открыть файл с помощью функции open.
3. Вывести информацию в файл.
4. Обязательно закрыть файл.

Как было сказано ранее, для того чтобы начать работать с текстовым файлом, необходимо описать переменную типа ofstream. Например, так:

```
ofstream outFile;
```

Будет создана переменная outFile для записи информации в файл. На следующем этапе файл необходимо открыть для записи. В общем случае оператор открытия потока будет иметь вид:

```
outFile.open("file.txt", mode);
```

Здесь outFile — переменная, описанная как ofstream, file.txt — полное имя файла на диске, mode — режим работы с открываемым файлом. Стоит обратить внимание на то, что при указании полного имени файла нужно ставить двойной слеш. Для обращения, например к файлу “accounts.txt”, находящемуся в папке sites на диске D, в программе необходимо указать: “D:\\sites\\accounts.txt”.

Функция open() требует в качестве аргумента строки в стиле C. Это может

быть литеральная строка или же строка, сохраненная в символьном массиве.

Файл может быть открыт в одном из следующих режимов:

- ios:: in — открыть файл в режиме чтения данных; режим является режимом по умолчанию для потоков ifstream;
- ios::out — открыть файл в режиме записи данных (при этом информация о существующем файле уничтожается); режим является режимом по умолчанию для потоков ofstream;
- ios::app — открыть файл в режиме записи данных в конец файла;
- ios::ate — передвинуться в конец уже открытого файла;
- ios::trunc — очистить файл, это же происходит в режиме ios::out;
- ios::nocreate — не выполнять операцию открытия файла, если он не существует;
- ios::noreplace — не открывать существующий файл.

Параметр mode может отсутствовать, в этом случае файл открывается в режиме по умолчанию для данного потока.

```
ifstream file;
file.open("Test.txt", ios::in); // открыть файл в режиме для чтения
ifstream file;
file.open("Test.txt"); // открыть файл в режиме для чтения (по умолчанию)
```

После удачного открытия файла (в любом режиме) в переменной outFile будет храниться true, в противном случае false. Это позволит проверить корректность операции открытия файла.

Для считывания данных из текстового файла, необходимо:

1. Описать переменную типа ifstream.
2. Открыть файл с помощью функции open.
3. Считать информацию из файла, при считывании каждой порции данных необходимо проверять, достигнут ли конец файла.
4. Закрыть файл.

Для того чтобы прочитать информацию из текстового файла, необходимо описать переменную типа ifstream. После этого нужно открыть файл для чтения с помощью оператора open. Если переменную назвать fromFile, то первые два оператора будут такими:

```
ifstream fromFile;
fromFile.open("D:\\sites\\accounts.txt");
```

После открытия файла в режиме чтения из него можно считывать информацию точно так же, как и с клавиатуры, указав имя потока, из которого будет происходить чтение данных.

Например, для чтения данных из потока fromFile в переменную a, оператор ввода будет выглядеть так:

```
fromFile >> a;
```

Два числа в текстовом редакторе считаются разделенными, если между ними есть хотя бы один из символов: пробел, табуляция, символ конца строки. Хорошо, когда программисту заранее известно, сколько и какие значения

хранятся в текстовом файле. Однако часто известен лишь тип значений, хранящихся в файле, при этом их количество может быть различным. Для решения данной проблемы необходимо считывать значения из файла поочередно, а перед каждым считыванием проверять, достигнут ли конец файла с помощью функции fromFile.eof(). Здесь fromFile — имя потока, eof() - функция, возвращающая логическое значение (true или false), в зависимости от того достигнут ли конец файла.

Следовательно, цикл для чтения содержимого всего файла можно записать так:

```
    //организуем для чтения значений из файла цикл, выполнение //цикла прервется, когда
достигнем конец файла,
//в этом случае fromFile.eof() вернет истину
    while(!fromFile.eof())
    {
        //чтение очередного значения из потока fromFile в переменную a fromFile >> a;
        //далее идет обработка значения переменной a
    }
```

Для чтения отдельных символов можно использовать функцию get() и функцию getline() - для чтения целых строк.

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>

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));
    }
}
```

1.4. Содержание отчета

Отчет к лабораторной работе должен содержать:

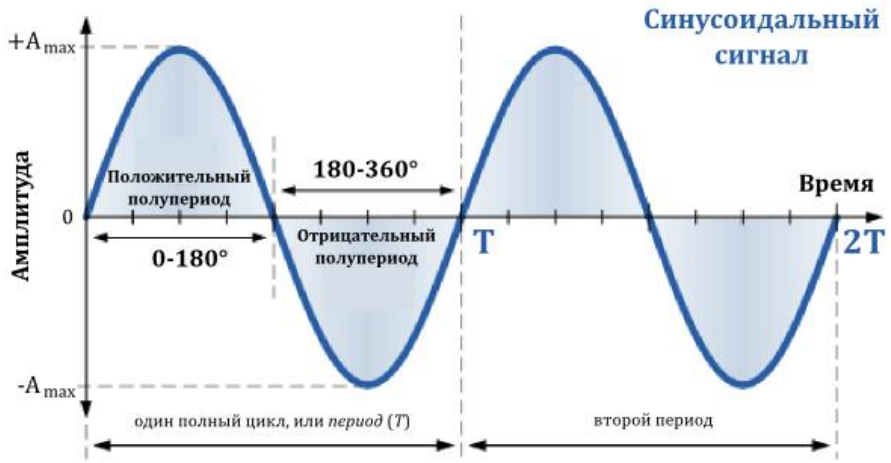
1. Текст задания, вариант.
2. Листинг программы.

3. Файл с отсчетами сложного сигнала

1.5. Варианты задания

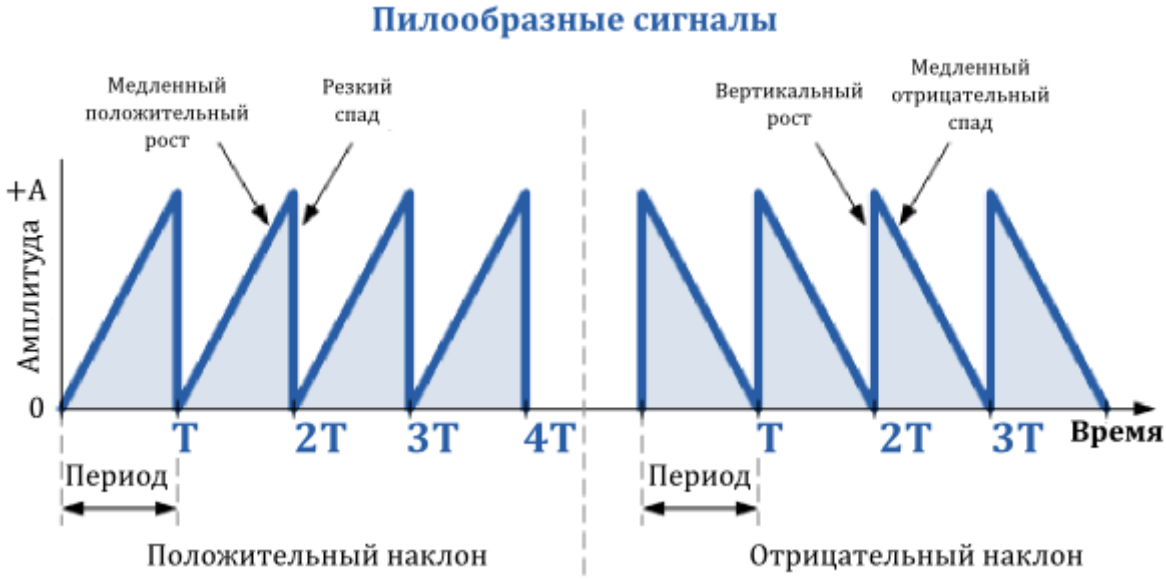
Простые сигналы представлены следующим перечнем:

1. SIN – синусоидальный сигнал вида S(t) = A*sin(2*pi*f*i/Fs)

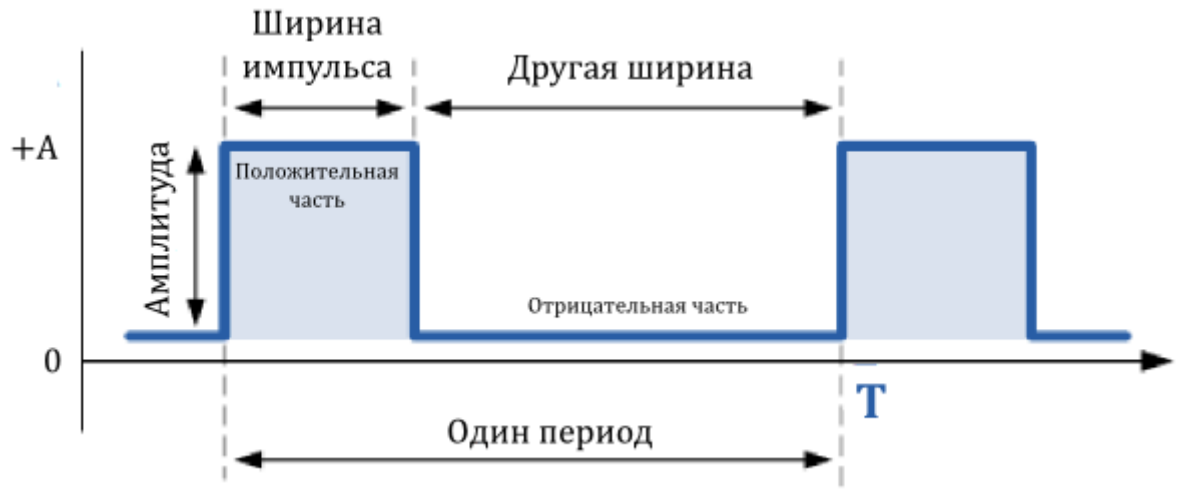


Где f = 1/T

2. COS – косинусоидальный сигнал вида S(t) = A*cos(2*pi*f*i/Fs)
3. PILA – сигнал пилообразной формы



4. CONST - постоянное напряжение уровня A
5. GTI – тактовые импульсы



Для определения исходных данных используется номер студенческого билета N и номер по списку группы K.

Параметры для SIN: A = 1; f = 1000 + 100*N Гц; Fs = 48 кГц;

Параметры для COS: A = 1; f = 12000 - 50*N Гц; Fs = 48 кГц;

Параметры для PILA: A = 1; T = N%63 отсчетов;

Параметры для CONST: A = 0.01*N + K/2;

Параметры для GTI: Период N%16 отсчетов; Ширина K%15 отсчетов;

Номер студенческого билета	Последовательность простых сигналов в составе сложного	Номер студенческого билета	Последовательность простых сигналов в составе сложного
1.	SIN PILA CONST GTI	2.	CONST GTI SIN PILA
3.	SIN PILA CONST COS	4.	CONST GTI SIN COS
5.	SIN PILA GTI CONST	6.	CONST GTI PILA SIN
7.	SIN PILA GTI COS	8.	CONST GTI PILA COS
9.	SIN PILA COS CONST	10.	CONST GTI COS SIN
11.	SIN PILA COS GTI	12.	CONST GTI COS PILA
13.	SIN CONST PILA GTI	14.	CONST COS SIN PILA
15.	SIN CONST PILA COS	16.	CONST COS SIN GTI
17.	SIN CONST GTI PILA	18.	CONST COS PILA SIN
19.	SIN CONST GTI COS	20.	CONST COS PILA GTI
21.	SIN CONST COS PILA	22.	CONST COS GTI SIN
23.	SIN CONST COS GTI	24.	CONST COS GTI PILA
25.	SIN GTI PILA CONST	26.	GTI SIN PILA CONST
27.	SIN GTI PILA COS	28.	GTI SIN PILA COS
29.	SIN GTI CONST PILA	30.	GTI SIN CONST PILA
31.	SIN GTI CONST COS	32.	GTI SIN CONST COS

33.	SIN GTI COS PILA	34.	GTI SIN COS PILA
35.	SIN GTI COS CONST	36.	GTI SIN COS CONST
37.	SIN COS PILA CONST	38.	GTI PILA SIN CONST
39.	SIN COS PILA GTI	40.	GTI PILA SIN COS
41.	SIN COS CONST PILA	42.	GTI PILA CONST SIN
43.	SIN COS CONST GTI	44.	GTI PILA CONST COS
45.	SIN COS GTI PILA	46.	GTI PILA COS SIN
47.	SIN COS GTI CONST	48.	GTI PILA COS CONST
49.	PILA SIN CONST GTI	50.	GTI CONST SIN PILA
51.	PILA SIN CONST COS	52.	GTI CONST SIN COS
53.	PILA SIN GTI CONST	54.	GTI CONST PILA SIN
55.	PILA SIN GTI COS	56.	GTI CONST PILA COS
57.	PILA SIN COS CONST	58.	GTI CONST COS SIN
59.	PILA SIN COS GTI	60.	GTI CONST COS PILA
61.	PILA CONST SIN GTI	62.	GTI COS SIN PILA
63.	PILA CONST SIN COS	64.	GTI COS SIN CONST
65.	PILA CONST GTI SIN	66.	GTI COS PILA SIN
67.	PILA CONST GTI COS	68.	GTI COS PILA CONST
69.	PILA CONST COS SIN	70.	GTI COS CONST SIN
71.	PILA CONST COS GTI	72.	GTI COS CONST PILA
73.	PILA GTI SIN CONST	74.	COS SIN PILA CONST
75.	PILA GTI SIN COS	76.	COS SIN PILA GTI
77.	PILA GTI CONST SIN	78.	COS SIN CONST PILA
79.	PILA GTI CONST COS	80.	COS SIN CONST GTI
81.	PILA GTI COS SIN	82.	COS SIN GTI PILA
83.	PILA GTI COS CONST	84.	COS SIN GTI CONST
85.	PILA COS SIN CONST	86.	COS PILA SIN CONST
87.	PILA COS SIN GTI	88.	COS PILA SIN GTI
89.	PILA COS CONST SIN	90.	COS PILA CONST SIN
91.	PILA COS CONST GTI	92.	COS PILA CONST GTI
93.	PILA COS GTI SIN	94.	COS PILA GTI SIN
95.	PILA COS GTI CONST	96.	COS PILA GTI CONST
97.	CONST SIN PILA GTI	98.	COS CONST SIN PILA
99.	CONST SIN PILA COS	100.	COS CONST SIN GTI
101.	CONST SIN GTI PILA	102.	COS CONST PILA SIN
103.	CONST SIN GTI COS	104.	COS CONST PILA GTI
105.	CONST SIN COS PILA	106.	COS CONST GTI SIN
107.	CONST SIN COS GTI	108.	COS CONST GTI PILA
109.	CONST PILA SIN GTI	110.	COS GTI SIN PILA
111.	CONST PILA SIN COS	112.	COS GTI SIN CONST
113.	CONST PILA GTI SIN	114.	COS GTI PILA SIN
115.	CONST PILA GTI COS	116.	COS GTI PILA CONST
117.	CONST PILA COS SIN	118.	COS GTI CONST SIN
119.	CONST PILA COS GTI	120.	COS GTI CONST PILA