Лабораторная работа №2.

ПОДПРОГРАММЫ В C++.

1. Цель и задачи работы

Цель: освоение методов программирования с использованием подпрограмм.

Задачи:

– проверка знаний по организации и использованию подпрограмм, понимание способов передачи параметров;

– знакомство с сообщениями транслятора при обнаружении ошибок в описаниях и/или обращениях к подпрограммам;

– знакомство с некоторыми видами ошибок при описании и вызове подпрограмм, тренировка умения поиска и устранения ошибок (отладка);

– получение практических навыков программирования с использованием подпрограмм.

2. Основные сведения

Назначением подпрограмм (процедур и функций) является уменьшение трудоемкости алгоритмизации и переход к структурным методам программирования задач, содержащих повторяющиеся последовательности операций (возможно, с различными операндами) путем организации программ с иерархической структурой.

Подпрограммой на языке C++ называется именованная последовательность операторов, которая может быть вызвана одним оператором, называемым оператором подпрограммы.

Если подпрограмма даёт одно результирующее значение, которое присваивается имени подпрограммы, то такая она называется *функцией*. Данное значение через имя функции может использоваться в выражениях. В C++ все подпрограммы являются функциями. Они делятся на:

- функции типа `void` – в качестве их аналога можно привести процедуры языка Паскаль;
- функции с возвращаемым значением.

Описание данной поименованной последовательности операторов называется *описанием подпрограммы*. Описание подпрограммы состоит из заголовка и блока. *Заголовок* состоит из типа возвращаемого значения со следующим за ним идентификатором – именем подпрограммы и, возможно, списка формальных параметров:

```
void FuncName1();       // функция типа void без параметров
void FuncName2 (список_формальных_параметров>);
                        // функция типа void с параметрами
ИмяТипа FuncName3 ();  // функция без параметров
ИмяТипа FuncName4 (список_параметров); // функция с параметрами
```

Функция может вернуть любое значение, кроме массива. Однако функция может вернуть массив в составе структуры или объекта.

Блок или *тело* подпрограммы состоит из определений и операторов, организованных в единый блок с помощью скобок `{` и `}` и выполняющих требуемые действия, ради которых создана подпрограмма. Обязательным требованием к описанию функции является наличие в её теле оператора `return` (кроме функция типа `void`), который передаёт в вызывающую подпрограмму результат:

```
return выражение/значение_типа_функции;
```

Оператор вызова подпрограммы состоит из имени подпрограммы и, возможно, списка фактических параметров. Список фактических параметров должен соответствовать списку формальных параметров по порядку и типу.

Тип каждого фактического параметра определяется типом соответствующего формального параметра, который специфицируется в заголовке подпрограммы. Помимо этого, в заголовке подпрограммы при описании списка формальных параметров указывается *способ подстановки фактических параметров вместо формальных*. Рассмотрим следующие способы подстановки:

1. *Передача параметров по значению (используется по умолчанию в C++)*. Способ подстановки значением, при котором вычисленное значение фактического параметра (выражения)

подставляется вместо соответствующего формального параметра:

– заголовок: `int OutParamValue (float X, double Eps);`
`// X и Eps являются параметрами-значениями`

– вызов: `OutParamValue (Y+3.123, EpsFact);`
`// Y+3.123 и EpsFact есть выражения`

При использовании *параметра-значения* при вызове подпрограммы сначала вычисляется значение фактического параметра (выражения Y+3.123 и EpsFact), потом это значение передается в формальный параметр (X и Eps), поэтому изменение формального параметра (Eps) внутри подпрограммы не изменяет фактический параметр (EpsFact), даже, если их имена одинаковые.

2. *Передача параметров по ссылке (по адресу, способ подстановки переменной*) определяется с помощью передачи ссылки на переменную. В качестве формального параметра в заголовке функции должна присутствовать ссылка на переменную, а фактический параметр всегда должен быть адресом расположения необходимого значения, заменяющим соответствующий формальный параметр:

– заголовок: `double GetParamValue (float &Z, short int &N);`
`// Z и N являются параметрами-ссылками`
– вызов: `GetParamValue (M[I-1], Count)`
`// M[I-1] и Count - фактические переменные, являющиеся адресами памяти`

При использовании параметра-ссылки при вызове подпрограммы ей передаётся адрес фактической переменной, поэтому любое изменение значения через параметр-ссылку (`Z` и `N`) внутри подпрограммы немедленно изменяет фактическую переменную (`M[I-1], Count`).

При *выборе способа подстановки* и его реализации следует учитывать следующие общие правила:

- если параметр является аргументом (исходным данным) функции, то предпочтительнее подстановка значений (или передача по значению);

- если параметр обозначает результат подпрограммы, то необходима передача параметров по ссылке (подстановка переменной);

- рекомендуется в качестве фактических параметров, подставляемых вместо различных формальных параметров, как переменные использовать *различные* объекты (то есть переменные, имеющие различные идентификаторы). В противном случае возможно получение неверного результата. Например, функция с заголовком Mult (**int** X[], **int** Y[], **int** Z[]), предназначенная для умножения двух целочисленных матриц X и Y и получающая результат в Z, осуществляет данную операцию путем умножения соответствующих элементов матриц с последующим сложением произведений. Её вызов вида Mult (A, B, A) приведёт к тому, что в массиве A будет получен неверный результат;

- при передаче в функцию сложных структур данных, занимающих большой объём памяти, рекомендуется использовать передачу по ссылке.

Умалчиваемые значения параметров. При определении функции можно задать начальные значения параметров непосредственно в заголовке функции через знак равенства после имени параметра, например:

```
void Display (int Value=1, char Name[]="Номер дома: ") {
     cout << "\n" << Name << Value;
}
```

Эти начальные значения или значения параметров по умолчанию используются, если при вызове функции соответствующий фактический параметр опущен. Однако параметры можно опускать, начиная только с конца их списка.

Пример обращений к функции `Display`:

```
Dislpay ();      // Выводит:  Номер дома: 1
Dislpay (15);    // Выводит:  Номер дома: 15
Dislpay (6, "Размерность массива=");
                 // Выводит:  Размерность массива=6
```

Если объект (константа, тип, переменная) имеет смысл только в пределах подпрограммы, то этот объект называется *локальным* и он должен быть описан в теле функции, желательно перед

операторами тела функции.

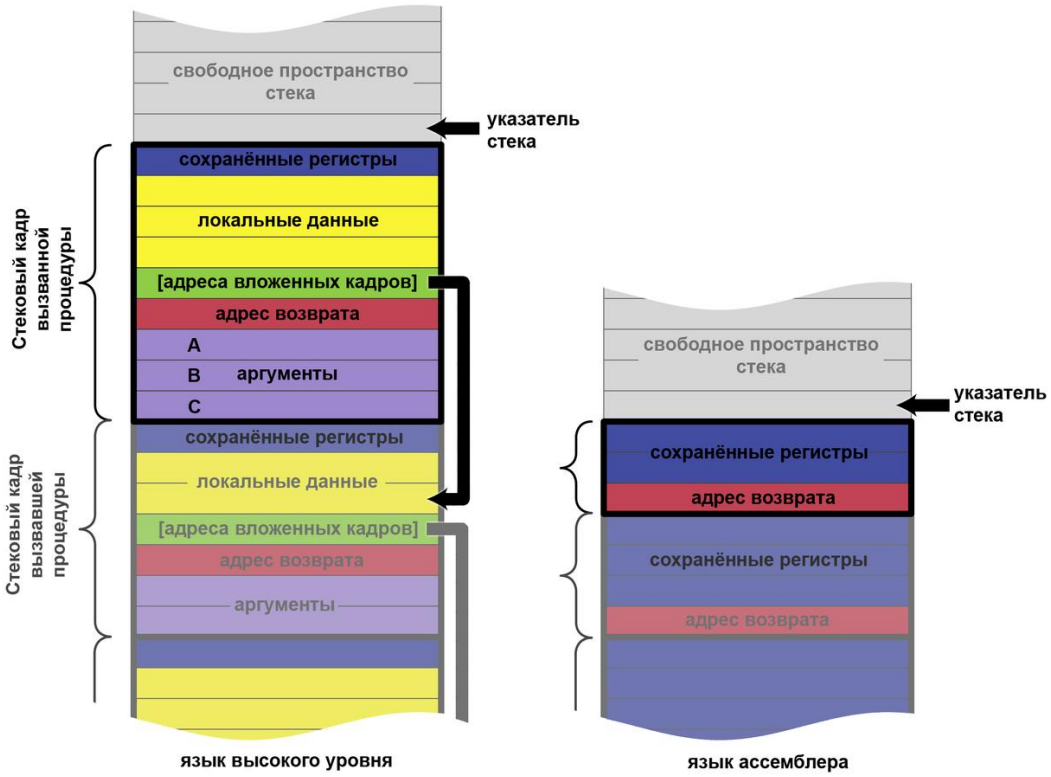
Областью существования локального объекта является все тело подпрограммы, где он определён. На локальные объекты нельзя ссылаться вне подпрограммы, где они объявлены, так как там они не определены.

Если для какой-либо подпрограммы имена глобальных и локальных объектов совпадают, то такие объекты переопределяются в пределах данной функции без изменения их внешних определений. Если в описании подпрограммы используется объект, который не определен в подпрограмме, то он должен быть определён за её пределами в качестве глобального.

Областью хранения локальных переменных функции и её параметров является область памяти, называемая *стеком вызовов* функций (структура типа LIFO). Основное назначение стека вызовов – отслеживать место, куда каждая из вызванных функций должна вернуть управление после своего завершения. Для этого при вызове функции в стек заносится адрес команды, следующей за командой вызова («адрес возврата»).

Кроме адресов возврата в стеке могут сохраняться другие данные (рисунок), например:

- значения регистров центрального процессора с их последующим восстановлением;
- аргументы, переданные в функцию;
- локальные переменные – временные данные функции;
- и другие произвольные данные.



Возможное «наполнение» стека вызовов функций

По завершении вызванная функция выполняет команду возврата (return) для перехода по адресу из стека. При этом указатель стека перемещается к данным вызывающей функции и, как следствие, локальные переменные вызываемой функции становятся не доступными, теряются.

Если процедура или функция изменяет значения глобальных переменных, то говорят, что она имеет *побочный эффект*. Бесконтрольное использование глобальных переменных может привести к нежелательным побочным эффектам, при которых подпрограмма производит незапланированные во внешней программе, вызывающей данную, изменения переменных. При описании функций вообще не рекомендуется использовать глобальные переменные.

В целях обеспечения безопасности от побочных эффектов желательно: 1) не использовать одинаковые имена объектов в областях видимости разного уровня; 2) описывать вновь вводи-

мые объекты сразу по мере их появления.

В программе на языке С++ могут использоваться обращения к стандартным подпрограммам, которые считаются описанными в программной среде, окружающей программу пользователя. К стандартным относятся некоторые арифметические функции (например, abs(X), sin(X)), функции работы с файлами (например, FileName.close()), функции преобразования и другие.

Пример программы, составленной с использованием функций:

```
// Test22A
#include <iostream>
using namespace std;
int A,B,T,P,I;
int M[2];

void P1();
void P2();
void P3(int X, int Y);
void P4(int & X, int & Y);
int Kv(int & A);

void P5(int & X, int & Y) { // параметры-ссылки
int T;
T=Kv(X); X=Kv(Y); Y=T; // используем значения,
                       // на которые они ссылаются
}

int main() {
      cout << "Введите целое число в интервале 1..9: ";
      cin >> P;

      I=1; M[0]=0; M[1]=1; A=1; B=2; T=3;
      switch (P) {
            case 1: P1(); break;
            case 2: P2(); break;
            case 3: P3 (A,B); break;
            case 4: P3 (A+1,B+2); break;
            case 5: P4 (A,B); break;
            case 6: P4 (I,M[I]); A=I; B=M[0]; T=M[1]; break;
            case 7: P5 (A,B); break;
            case 8: P5 (B,B); break;
}
      cout << "A=" << A << " B=" << B << " T=" << T;
   return 0;
}

void P1() { // без параметров
T=A; A=B; B=T;
}
void P2() { // без параметров
      int T;
      T=A; A=B; B=T;
}
void P3(int X, int Y) { // параметры-значения
      T=X; X=Y; Y=T;
}
void P4(int & X, int & Y) { // параметры-ссылки
      int T;
      T=X; X=Y; Y=T;
}
int Kv(int &A) {  // параметр-ссылка
      return A * A;
}
```

Данная программа содержит описания шести функций **P1**, **P2**, **P3**, **P4**, **P5** и **Kv**, которая является вспомогательной для **P5**.

Функция **P1** не имеет формальных и локальных параметров. В функции**P2** определен локальный параметр **T**. В функции **P3** используются как формальные параметры, подставляемые значением, так и глобальный параметр **T**. В функции **P4** один локальный параметр **T** и два формальных параметра, подставляемые по ссылке. Функция **P5** использует функцию **Kv**, имеющую формальный параметр, подставляемый по ссылке.

В программе **Test22A** в зависимости от вводимого значения переменной **P** осуществляется вызов той или иной функции с подстановкой фактических параметров.

Результатом работы программы являются значения переменных **A**, **B** и **T**, выводимые на экран дисплея.

3. Ошибки, сообщения компилятора и отладка

При программировании с использованием функций все допустимые ошибки можно классифицировать следующим образом:

1. *Синтаксические ошибки* или ошибки кодирования – нарушение правил записи описаний или операторов вызова функций. Ошибки данного вида обнаруживаются при трансляции программ, а сообщения о месте и типе ошибки выдаются компилятором на экран дисплея путем выдачи сообщения с номером ошибки и позиционирования текстового курсора в позицию обнаружения ошибки.

Синтаксические ошибки устраняются путем редактирования текста программы и повторной трансляции.

Компилятор может отмечать как ошибочные фрагменты программы, не содержащие синтаксических ошибок. Такие сообщения транслятора называются *наведёнными* и они связаны с ошибками, допущенными ранее по тексту программы.

Устранение наведенных ошибок осуществляется путем поиска и устранения всех синтаксических ошибок в предыдущем тексте программы, не являющихся наведенными (т.е. поддающихся интерпретации на предмет смысла сделанной ошибки).

2. *Семантические или смысловые ошибки* связаны либо с неправильно заданным способом передачи параметров, либо с наличием нежелательного побочного эффекта. Они выявляются только на этапе выполнения программы путем сравнения выдаваемых конечных результатов, получаемых с помощью операторов контрольного вывода, с рассчитанными вручную для тестового примера.

При поиске семантических ошибок в процедурах рекомендуется вставлять операторы контрольного вывода значений фактических и глобальных параметров непосредственно перед оператором вызова функции и после него или перед оператором присваивания, содержащем в правой части выражение, в котором используется обращение к функции. После данного оператора присваивания выводится значение переменной в левой части. В сочетании с операторами контрольного вывода, расположенными внутри тела функции, данный способ используется для уточнения места ошибки.

Ниже приводится программа **Test22B** на языке C++, составленная с использованием функции с именем **S**. Назначением данной подпрограммы является обмен значений переменных **A** и **B** без изменения значения переменной **T.** Следовательно, результатом работы программы **Test22B**, обращающейся к подпрограмме **S**, должна быть следующая строка на экране дисплея:

2 1 3.

В данной программе и функции сделаны не менее трёх синтаксические и двух семантических ошибок. Кроме того, при некорректном исправлении одной из синтаксических ошибок при выполнении возможен нежелательный побочный эффект.

```
// Test22B;
#include <iostream>
int A;
float B;
void S (int X,Y) {
```

```
      T=X; X=Y; Y=T;
}

int main () {
      A=1; B=2; T=3;
      S(&A);
      std::cout << std::setw(7) << A
            << std::setw(7) << B
            << std::setw(7) << T;
      return 0;
}
```

4. Индивидуальные задания

Составить программу для вычисления **F** по следующей формуле (символ "^" означает возведение в степень):

F = (A^2 + B^2)^2 + (C^2 + D^2)^2 . (1)

Вариант 1.

В программе должны быть описаны и использованы две подпрограммы: Квадрат и Сумма.

Вариант 2. Составить программу для вычисления **F** по формуле (1) с описанием и вызовом подпрограммы вычисления **T** без использования её прототипа:

T = (P^2 + L^2)^2

При этом:

а) подпрограмма T – функция типа **void** с формальными параметрами P и L.

или

б) подпрограмма T – функция с возвращаемым значением типа **float**.

При описании функций в основной программе следует обратить особое внимание на способ передачи параметров (по значению или ссылке), использование локальных и глобальных переменных и возможность нежелательного побочного эффекта.

Оба варианта задания должны содержать прототипы функций Квадрат и Сумма. Используемые переменные перечислены в табл.1 и индивидуальном задании, но не более чем A, B, C, D, X, Y, Z.

Возможные способы описания подпрограмм представлены в табл. 1.

Таблица 1

Подпрограммы вычисления квадрата и суммы и способы их описания

N1: Y = X*X	**N2: Z = X+Y**
1.Функция	1.Функция
2. Функция типа void без формальных параметров	2. Функция типа void без формальных параметров
3. Функция типа void с формальным параметром X	3. Функция типа void с формальными параметрами X, Y
4. Функция типа void с формальным параметром Y	4. Функция типа void с формальным параметром Z
5. Функция типа void с формальными параметрами X, Y	5. Функция типа void с формальными параметрами X, Y, Z

5. Подготовка к работе

1. Изучить описание лабораторной работы.
2. Проанализировать текст программы **Test22A**. По результатам анализа заполнить столбцы 2, 3, 4 и 8 табл. 2 при заданных в первом столбце значений входного параметра **P**.

Таблица 2

Анализ программы Test22A

	Результаты						Передача параметров (по ссылке или по значению)
	Анализа программы **Test22A**			Выполнения программы **Test22A**			
1	2	3	4	5	6	7	8
P	**A**	**B**	**T**	**A**	**B**	**T**	
1							
2							
3							
4							
5							
6							
7							
8							
9							

3. Переписать текст программы **Test22B**, проанализировать его, письменно указать обнаруженные ошибки и письменно пояснить суть ошибки.

4. Написать программу по индивидуальному заданию. Предусмотреть ввод исходных данных, контрольный вывод их, вывод результата **F** и предусмотреть операторы отладочного вывода значений переменных. Заготовить таблицу с исходными (тестовыми) данными для проверки работоспособности программы.

Выполняемое задания определяется тремя цифрами, зависящими от **N** – номер по списку в журнале преподавателя:

1-я цифра – номер варианта (см. п. «Индивидуальное задание»):

=1, если N – нечётное;

= 2а, если (N+1) **%** 4 == 1;

= 2б, если (N+1) **%** 4 == 3:

2-я цифра – номер N1 способа описания подпрограммы Квадрат:

N1 = (N-1) / 5 + 1,

3-я цифра – номер N2 способа описания подпрограммы Сумма:

N2 = (N-1) % 5 + 1.

6. Порядок выполнения работы

1. Запустить программу **Test22A** на выполнение, задавая различные значения P. По результатам работы, получаемым на экране дисплея, заполнить столбцы 5, 6, 7 табл. 2. Сравнить полученные результаты с результатами анализа.

2. Транслировать программу **Test22B**. Исправить все синтаксические ошибки, оттранслировать и запустить программу на выполнение. По результатам трансляции заполнить табл. 3 и часть табл. 4 (колонки «Результаты выполнения после устранения ошибок: *синтаксических*»). Проанализировать правильность результатов выполнения.

Таблица 3

Сообщения компилятора при отладке программы

№ строки программы	Текст сообщения об ошибке		Классификация ошибки (синтаксическая, семантическая, наведённая) и причина возникновения
	на английском языке	на русском языке	

Найти и исправить семантические ошибки. Произвести повторную трансляцию и запуск на выполнение. Заполнить часть табл. 4 (колонки «Результаты выполнения после устранения ошибок: *семантических*»).

Проанализировать результат выполнения на наличие побочного эффекта и, при наличии, устранить его. Произвести трансляцию и запуск на выполнение. Завершить заполнение табл. 4.

Таблица 4

Результаты выполнения после устранения ошибок:								
синтаксических			семантических			побочного эффекта		
A	**B**	**T**	**A**	**B**	**T**	**A**	**B**	**T**

3. Напечатать, оттранслировать и выполнить программу по индивидуальному заданию при следующих исходных данных:

а) A=1, B=2, C=3, D=4 (стандартный набор);

б) A, B, C и D – произвольные (4 набора).

Зафиксировать результаты выполнения и проанализировать правильность работы программы. Если необходимо, отладить программу.

7. Содержание отчета

1. Титульный лист.
2. Цель работы
3. Материалы в соответствии с подготовкой к работе (пункты 2-4 «Подготовки к работе»).
4. Текст и заполненная табл. 2 входных и выходных данных программы **Test22A**.
5. Исправленный текст программы **Test22B** и заполненные табл. 3 и 4.
6. Индивидуальное задание.
7. Текст программы, написанный по индивидуальному варианту задания. Результаты работы программы для стандартного и произвольных наборов данных.
8. Выводы по работе.

8. Пример программы

Задание для N=34.

Выполняемое задания:

1-я цифра = (34+1)%4 = 3 → вариант 2б: подпрограмма T – функция с возвращаемым значением типа **float**;

2-я цифра – способ описания подпрограммы Квадрат: N1 = 33 / 5 + 1 = 7 → берём способ 2, т.к. даны всего 5 способов, считаем «по кругу» → подпрограмма Квадрат: функция типа **void** без формальных параметров;

3-я цифра – способ описания подпрограммы Сумма: N2 = (34-1) % 6 + 1 = 4 → подпрограмма Сумма: функция типа **void** с формальным параметром Z.

Индивидуальное задание для N=30:

Вычислить F = (A^2+B^2)^2 + (C^2+D^2)^2:

(34+1) **%** 4 = 3 -> вариант 2б: подпрограмма T – функция с возвращаемым значением типа **float**;

(34-1) / 5 +1 = 7, берём N1=2 -> подпрограмма Квадрат: функция типа **void** без формальных параметров;

(34-1) **%** 5 + 1 = 4 = N2 -> подпрограмма Сумма: функция типа **void** с формальным параметром Z

```
// Exz_Proc_Func_22
#include <iostream>
```

```
using namespace std;
short int A,B,C,D,F;
short int X,Y;
// прототипы фукнций
void Kvadrat ();              // Y = X^2
void Summa (short int & Z);  // Z = X+Y

float FuncT (short int P, short int L) {
      // P^2 -> Y; X и Y глобальные
    X=P;   Kvadrat();   P=Y;
      // L^2 -> Y; X и Y глобальные
    X=L;   Kvadrat();
      X=Y;   Y=P;
      // X+Y -> P
      Summa(P);
      // X^2 -> Y глобальный
      X=P;   Kvadrat();
      return Y;
}

int main () {
      std::cout << " Введите 4 целых числа: ";
      std::cin >> A >> B >> C >> D;
      // 1 4 3 2 -> F=458
      // (A^2+B^2)^2 -> F, в X и Y нельзя, заняты в FuncT()
      F = FuncT(A, B);
      // (C^2+D^2)^2 -> A глобальный
      A = FuncT(C, D);
      // F+A -> F
      X=F; Y=A;  Summa(F);
      // Выводим результат
      std::cout << "F=" << F;
      return 0;
}

void Kvadrat () {
      Y = X*X;  // X,Y - глобальные
}

void Summa (short int & Z) {
      Z = X + Y;  // X,Y - глобальные
}
```