**Занятие 1. Программы линейной структуры.**

**Целый тип данных**

К числу стандартных целых типов относятся:

int – тип, представляющий целые со знаком от –231 до +231-1 (от -2147483648 до 2147483647), занимает 4 байта.

unsigned int – тип, представляющий целые без знака (на что указывает *модификатор* unsigned) от 0 до +232-1 (от 0 до 4294967295), занимает 4 байта.

// объявление целых переменных I и K как целых со знаком

int I, K=-5;

// объявление целых переменных i и k как целых без знака

unsigned int i, k=5;

//при выполнении программы значения переменных можно изменять

// объявление именованной целой константы со знаком

const int Nmax=10;

//при выполнении программы значения констант нельзя изменять

Для данных целого типа определены следующие арифметические операции (операторы, используемые только в выражениях, то есть как составная часть других операторов). Результат их выполнения также будет иметь тип целый:

изменение знака (унарный минус -), сложение (знак +),

вычитание (знак -), умножение (знак \*),

целочисленное деление (знак **/**), взятие по модулю (знак **%**).

Результатом выполнения операции / является целая часть частного, а операции % – остаток от целочисленного деления (знак остатка всегда совпадает со знаком делимого). Например,

int I, K;

I = -5 / -2; //I получит значение 2

K = -5 % -2; //K получит значение -1

Частью выражений целого типа могут быть также вызовы функций, возвращающих целые значения, и операторы присваивания целых значений (см. ниже). Если же хотя бы одна составляющая выражения имеет вещественный тип, то вычисленное значение всего выражения будет иметь вещественный тип.

**Вещественные типы данных**

К числу *стандартных вещественных* (*действительных*) *типов* относятся:

float - тип, представляющий вещественные числа со знаком с абсолютными значениями от 1.175494351·10 –38 до 3.402823466·10+38, занимает 4 байта.

double - тип, представляющий вещественные числа со знаком удвоенной точности с абсолютными значениями от 2.2250738585072014·10+308 до 1.7976931348623158·10+308, занимает 8 байтов.

Например, чтобы объявить переменные с именами r как float и R как double, в программе следует записать

float r;

double R;

Константы вещественного типа double записываются либо в *естественной форме*, например, -12.345, либо в *экспоненциальной форме*, в которой то же самое число можно записать по–разному, например, -0.12345E+2, или -0.12345E2, или -0.12345е+2, или –1.2345E+1, или –1.2345E1, или –12.345E0, или –1234.5E-2, или –12345E-3 и т.д. При представлении числа в такой форме безразлично, используется строчная или прописная латинская буква Е. Чтобы получить значение числа, представленного в экспоненциальной форме, нужно умножить *мантиссу*, то есть то, что стоит перед символом Е, на 10 в степени, значением которой является *порядок*, то есть целое число, записанное после Е.

Так, константу -0.12345E+2 следует читать как –0,12345·10+2, а константу -1234.5E-2 – как –1234,5·10-2. Константы вещественного типа float также записываются либо в естественной форме, либо в экспоненциальной форме, но с суффиксом f или F, например, -12.345f, или -0.12345E2f, или 12.345F, или -0.12345E2F и т.д. В большинстве случаев суффиксы не используются, так как компилятор распознаёт тип константы по месту её использования в программе.

Следующий фрагмент программы представляет объявления вещественных переменных Х и Y типа float, Z – типа double и именованной константы H со значением 0,00000025:

float Х , Y;

double Z;

const float H=2.5E-7;

Для данных вещественного типа определены следующие арифметические операции, результат выполнения которых также будет иметь вещественный тип, представляющий минимальный диапазон, включающий вычисленное значение:

изменение знака (унарный минус -), сложение (знак +), вычитание (знак -), умножение (знак \*), деление (знак /).

**Стандартные функции для обработки числовых данных**

Основные элементарные математические функции реализованы в C с помощью стандартных *библиотечных подпрограмм* - *математических функций*, подключаемых к программе директивой #include "math.h".(

Библиотечные математические функции возвращают результаты вещественного типа либо double, либо float, а их вещественные аргументы (то есть фактические параметры, которыми могут быть константы, переменные, выражения) должны быть соответствующего типа.

acos(X) – возвращает значение арккосинуса аргумента, asin(X) – возвращает значение арксинуса аргумента, atan(X) – возвращает значение арктангенса аргумента

в диапазоне от –π/2 до +π/2, ceil(X) – возвращает в вещественной форме наименьшее целое значение,

большее или равное аргументу, cos(X) – возвращает значение косинуса аргумента, exp(X) – возвращает значение ex, fabs(X) – возвращает абсолютное значение аргумента, floor(X) – возвращает в вещественной форме наибольшее целое значение,

меньшее или равное аргументу, log(X) – возвращает значение натурального логарифма аргумента, log10(X) – возвращает значение логарифма аргумента X по основанию 10, sqrt(X) – возвращает значение квадратного кореня аргумента, pow(X,Y) – возвращает значение X, возведенное в степень Y, sin(X) – возвращает значение синуса аргумента, tan(X) – возвращает значение тангенса аргумента,

Имена представленных стандартных функций соответствуют типу double. Для стандартных функций, возвращающих значения типа float, используются те же имена, но с суффиксом f, например acosf(X), fabsf(X), powf(X,Y). Если не указывать суффикс f, то компилятор будет распознавать тип результата, вычисляемого функцией, по типу аргументов. Для функций, имеющих несколько аргументов, допустимы лишь определённые сочетания типов параметров. Получить быструю подсказку по допустимым типам аргументов и результата, вычисляемого функцией, можно во всплывающем окне, если подвести курсор мыши к оператору вызова функции (см. рисунок – пример подсказки для функции pow). Из справки видно, что вызов оформлен правильно (компилятор не выдаст сообщение об ошибке), если X имеет любой вещественный тип и второй параметр целого типа, а результат будет иметь тот же тип, что и X.

**Оператор присваивания и его сокращенные формы**

Одним из знаков оператора присваивания, как указывалось ранее, является знак =. Справа от него записывается выражение (в частном случае константа, переменная или вызов функции), а слева – переменная, которая получит вычисленное значение выражения (правила записи и вычисления выражений представлены в следующем разделе). Например,

int I=5, K;

float X , Y=2.3;

double Z = 1.0e-2;

const float pi = 3.1415926535;

X = tan(pi/3); //X получит значение 1.732

Типы переменной и выражения могут не совпадать. В этом случае перед присваиванием значение выражения автоматически преобразуется к типу переменной. Если переменная имеет тип целый, а выражение – вещественный, то перед преобразованием типа дробная часть отбрасывается, например,

K = -tan(pi/3); //K получит значение -1

Есть и *сокращенные формы операторов присваивания*. Они бывают двух видов: одноместные и двуместные.

Знаками *одноместных операторов присваивания* являются составные знаки ++ (рядом два знака +) и -- (рядом два знака -), обозначающие соответственно увеличение на 1 (инкрементацию) и уменьшение на 1 (декрементацию) переменной, рядом с которой они расположены. Эти знаки могут располагаться либо перед, либо после переменной, которую требуется увеличить или уменьшить на 1, и это не имеет значения, если такой оператор не является частью выражения, где следует учитывать приоритеты операций. Например,

I--; // то же, что и оператор I = I-1;, и оператор –-I;

++I; // то же, что и оператор I = I+1;, и оператор I++;

Знаками *двуместных операторов присваивания* являются составные знаки, состоящие из знака операции (для числовых данных это знаки + - \* / %) и знака =. Так, если X - переменная, которой присваивается новое значение, а W выражение, то оператор

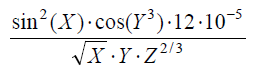
X += W ; // эквивалентен оператору X = X + W, X -= W; // эквивалентен оператору X = X – W, X \*= W; // эквивалентен оператору X = X \* W, X /= W; // эквивалентен оператору X = X / W, X %= W; // эквивалентен оператору X = X % W.

**Арифметические выражения**

Последовательность выполнения операторов в выражении определяется их приоритетами и порядком их следования, а при необходимости изменения порядка вычислений, как уже упоминалось, используют круглые скобки. Сейчас рассмотрим использование в выражениях только арифметических операций.

В выражениях в первую очередь вычисляются обращения к функциям и содержимое круглых скобок, затем – унарные операции изменения знака (-), затем - операции типа умножения (**\***, **/**, **%**) в порядке слева направо, затем – операции типа сложения (+ и -) в порядке слева направо. Если двуместная операция выполняется над вещественными данными разных типов (float и double), то результат будет иметь тип, представляющий больший диапазон значений (double).

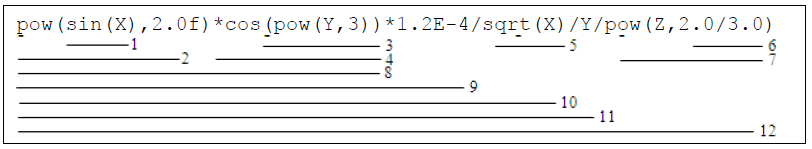
Например, для вычисления выражения



в программе можно записать

pow(sin(X),2.0f)\*cos(pow(Y,3))\*1.2E-4/sqrt(X)/Y/pow(Z,2.0/3.0).

Порядок вычислений поясняет следующий рисунок



Как и унарный минус (–), унарные операторы *инкрементации* (++) и *декрементации* (--) имеют наивысший приоритет. Однако если операторы ++ и -- меняют значение переменной, для которой они используются, то унарный минус не меняет значения переменной, а просто использует её значение со знаком минус. Операторы ++ и -- могут быть как *префиксными* (знак операции записываются перед переменной), так и *постфиксными* (знак операции записываются после переменной). От этого зависит, будет ли соответственно или новое, или исходное значение переменной использоваться для вычисления выражения. Например, если j равно 2 и целое k равно -3, то после выполнения вычислений

i=2\*k--+-++j

значением переменной i будет -9, k станет равным -4, а j – равным 3. Действия будут выполнены в следующем порядке: вначале будет выполнено ++j и j получит значение 3, затем, так как унарные операции выполняются в порядке справа налево, унарный минус сохранит значение –j, то есть -3, в некоторой дополнительной переменной, затем по части выражения 2\*k при исходном значении k будет вычислено значение -6 и прибавлено к -3 (значению -j), и только потом k будет уменьшено на 1 и станет равным -4. Эти же результаты можно получить, выполнив следующие три оператора присваивания:

j=j+1; i=2\*k-j; k=k-1;

Последний способ описания процесса вычисления значений переменных i, k и j обладает большей наглядностью и его следует использовать, если это возможно, чтобы уменьшить вероятность появления ошибки.

Если в рассмотренном выражении сделать декрементацию k префиксной:

i=2\*--k+-++j;

то i получит значение -11, так как до умножения на 2 переменная k будет уже иметь значение -4.

Двуместные формы оператора присваивания имеют одинаковый низший приоритет и выполняются в порядке справа налево. После выполнения каждого из них переменная получает новое значение, которое и используется для продолжения вычисления выражения. Так, одним выражением можно задать последовательность вычислений значений нескольких переменных, причем значением всего выражения будет значение, присвоенное последней, самой левой переменной.

Например, площадь S основания и объём V цилиндра радиуса R и высотой L можно вычислить одним выражением

V=L\*(S=3.1415926535\*R\*R)

Одним выражением можно присвоить одно и то же значение нескольким переменным, например, после вычисления выражения

X=Y=Z=3.5

переменные X, Y и Z будут иметь значение 3,5. Такая форма уменьшает текст программы и делает её более наглядной.

**Вывод десятичных чисел в окно программы**

В C для форматированного вывода данных в окно программы используется стандартная функция printf. Эта функция позволяет выводить данные различных типов (числовых, символьных, строковых), однако сейчас ограничимся рассмотрением вывода только целых и вещественных десятичных чисел, а также текстов, размещаемых непосредственно в функции printf, и некоторых символов, управляющих размещением данных в строках окна программы.

Первым в списке параметров функции printf является *управляющая строка*, а далее – список вывода, в котором через запятую записываются выражения (в частности переменные, константы), значения которых должны быть выведены. Список вывода может отсутствовать.

Основными элементами управляющей строки являются форматы вывода, которые состоят из одной или пары букв после знака %. Для десятичных данных есть следующие форматы вывода:

%d – для целых со знаком,

%u – для целых без знака,

%f – для вещественных типа float в естественной форме,

%lf – для вещественных типа double в естественной форме,

%e – для вещественных в экспоненциальной форме,

%g – для вещественных в наиболее компактной форме: экспоненциальной или в естественной форме.

Форматы, записанные в управляющей строке, ставятся в соответствие элементам списка вывода порядком их следования: первый по порядку формат соответствует первому элементу списка вывода, второй – второму элементу и т.д.

Например, если в программе есть переменные

int k=-25;

unsigned int j=7;

float x=-4.55;

double y=3.125E-2;

то оператором

printf("%d%u%f%lf",k,j,x,y);

будет выведена следующая строка чисел

-257-4.5500000.031250

Для разделения чисел в окне программы в управляющую строку между форматами следует вставлять либо тексты, поясняющие выводимые данные, либо простые печатные символы (пробелы, подчерки и другие) в качестве разделителей, либо управляющие символы (табуляции, перехода на новую строку), либо их сочетания. Далее представлены примеры их использования.

1. Использование поясняющих текстов: оператор

printf("k=%d j=%u x=%f y=%lf",k,j,x,y); выведет следующую строку в окно программы k=-25 j=7 x=-4.550000 y=0.031250

2. Использование пробелов: оператор

printf("k=%d %u %f %lf",k,j,x,y); выведет следующую строку в окно программы -25 7 -4.550000 0.031250

3. Использование управляющих знаков табуляции (\t) : оператор

printf("\n\t%d\t%u\t%f\t%lf\n",k,j,x,y); выведет следующую строку в окно программы -25 7 -4.550000 0.031250 4. Использование управляющих знаков (\n) перехода на новую строку: оператор

printf("%d\n%u\n%f\n%lf\n",k,j,x,y); выведет следующие строки в окно программы, в том числе пустую строку после чисел.

**Пример 1.** Составить программу вычисления тангенса угла, значение которого в градусах (целое число) вводится с клавиатуры в ответ на приглашение к вводу. Вывести на экран с пояснениями введенное значение угла в градусах, соответствующее ему значение в радианах и вычисленное значение тангенса этого угла.

#include <stdafx.h>

#include <math.h>

int \_tmain(int argc, \_TCHAR\* argv[])

{

int Fi;

float R;

//Вывод приглашения к вводу угла в градусах

printf("input Fi:");

//Ввод значения угла в переменную Fi

scanf("%d",&Fi);

//Вывод значения угла в градусах и пропуск строки

printf("Fi = %d\n",Fi);

//Перевод угла в радианы и присвоение переменной R

R=Fi\*atan(1.0)/45;

//Вывод значения R (угла в радианах) и пропуск строки

printf("\nR = %f rad\n",R);//

//Вывод tg(R) с поясняющим текстом

printf("\ntg(%lf) = %lf\n",R, tan(R));

return 0;

}

**Ввод десятичных чисел с клавиатуры**

В C для форматированного ввода данных с клавиатуры используется стандартная функция scanf. Эта функция позволяет вводить данные различных типов (числовых, символьных, строковых), однако сейчас ограничимся рассмотрением ввода только целых и вещественны десятичных чисел.

Первым параметром функции scanf является управляющая строка, а следующими - *указатели* на вводимые переменные, то есть имена переменных с предшествующим знаком &. В управляющей строке указываются форматы ввода одной или парой букв после знака %:

%d – для целых со знаком,

%u – для целых без знака,

%f – для вещественных типа float,

%lf – для вещественных типа double.

Например, если в программе есть переменные

int k;

unsigned int j;

float x;

double y;

и требуется ввести их значения, то можно использовать оператор

scanf("%d %u %f %lf",&k,&j,&x,&y);

**Приемы, используемые для минимизации вычислений**

Одним из критериев, характеризующих качество составленной программы, является объем выполняемых ею вычислений для достижения требуемого результата. Чем он меньше, тем, как правило, быстрее будет работать программа. Существуют разные приемы, применение которых позволяет сократить объем вычислений за счет уменьшения в первую очередь количества вызовов функций, затем – количества операций типа умножения, и, наконец, – количества операций типа сложения. Вот некоторые из них, рассмотренные отдельно, хотя, как правило, они используются в сочетании.

Вынесение общих множителей за скобки. Например, вместо оператора

Z=sin(X)\*Y-sin(X)\*sqrt(Y)\*Y+sqrt(X)\*X+X;

лучше использовать оператор

Z=sin(X)\*Y\*(1-sqrt(Y))+X\*(sqrt(X)+1);

Использование схемы Горнера для полиномов. Например, полином

,

преобразованный по схеме Горнера, примет вид

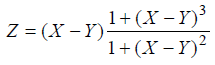
((((2X-5)X+2)X+7)X-4)X+6,

где явно меньше число операций умножения, если считать, что возведение в степень выполняется через умножение. В программе по второму варианту записи полинома будем иметь выражение

((((2\*X-5)\*X+2)\*X+7)\*X-4)\*X+6,

а по первому –

2\*pow(X,5)-5\*pow(X,4)+2\*pow(X,3)+7\*pow(X,2)-4\*X+6

Использование дополнительных переменных. Например, при вычислении значения функции 

целесообразно предварительно вычислить и , а исходную формулу преобразовать к виду . В этом случае в программе будут использованы три оператора присваивания:

A=X-Y;

B=A\*A;

Z=A\*(1+A\*B)/(1+B);

Пример 2.

Найти коэффициенты k0, k1, k2, k3 представления числа Х (0≤X≤80) в троичной системе счисления:

,

используя операции получения частного целочисленного деления (/) и остатка (%). Для контроля результатов выполнить вычисление Х непосредственно по формуле для найденных коэффициентов, а также после преобразования выражения в формуле по схеме Горнера. Вывести все результаты вычислений в наглядной форме с поясняющими текстами. Проверить работу программы на значениях

X=(0; 1; 2; 10; 27; 48; 80).

#include <stdafx.h>

int \_tmain(int argc, \_TCHAR\* argv[])

{

unsigned int X, k0, k1, k2, k3, X1, X2;

//Ввод исходных данных

printf("input X: ");

scanf("%u",&X);

//Вычисление коэффициентов разложения

k0=X%3; X=X/3; //k0:=остаток от деления X на 3;

k1=X%3; X=X/3; //k1:=остаток от деления X/3 на 3;

k2=X%3; //k2:=остаток от деления X/9 на 3;

k3=X/3; //k3:=X/27;

//Вывод вычисленных коэффициентов разложения

//числа Х по степеням 3 с поясняющими текстами

printf("\nk3=%u k2=%u k1=%u k0=%u",k3,k2,k1,k0);

// Проверки результатов вычислений

// 1.

//Вычисление непосредственно по формуле (2.1)

X1=k3\*27+k2\*9+k1\*3+k0;

//и вывод результата

printf("\nX1 = %u",X1);

// 2.

//Вычисление по формуле (2.1),

//преобразованной по схеме Горнера

X2=((k3\*3+k2)\*3+k1)\*3+k0;

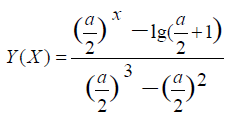
//и вывод результата

printf("\nX2 = %u\n",X2);

return 0;

}

**Пример 3.** Найти значение функции

,

упростив вычисления за счет использования скобочных форм и/или дополнительных переменных. Для контроля правильности результата выполнить вычисление по формуле без использования скобочных форм и дополнительных переменных.

Протестировать программу на значениях X=(0,5; 2) и A=(1; -1; 2; -2; 4; -4). Обратить внимание на вывод значений выражений при значениях аргументов, не принадлежащих областям определения входящих в выражения функций.

#include "stdafx.h"

#include "math.h"

int \_tmain(int argc, \_TCHAR\* argv[])

{

double A=2.0, X=0.5, B, C, Y1, Y2;

//Ввод исходных данных

printf("input X: ");

scanf("%lf",&X);

printf("input A: ");

scanf("%lf",&A);

//Вычисление выражения

// - с использованием дополнительных переменных

B=A/2;

C=B\*B;

Y1=(pow(B,X)-log10(B+1))/(B\*C-C);

// - непосредственно по формуле (2.2)}

Y2=(pow(A/2,X)-log10(A/2+1)) / (pow(A/2,3) -pow(A/2,2));

//Вывод вычисленных значений с надписями

printf("\n\t\tY1\t\t\tY2");

printf("\n\t\t%e\t\t%e\n",Y1,Y2);

return 0;

}