Балтийский государственный технический университет «ВОЕНМЕХ» им. Д. Ф. Устинова

Кафедра О7 «Информационные системы и программная инженерия»

**Практическая работа №1**по дисциплине «Информатика: Основы программирования» на тему «Структура программы, основные типы данных, ввод/вывод»

Выполнил:  
Студент: Вяткин Н. А.  
Группа *:* О722Б  
  
Преподаватель:  
Назарова М. А.

Санкт-Петербург  
2022 г.

**Задание 1.**

Написать программу, которая будет находить сумму любых двух целых чисел, введенных с клавиатуры.

*Входные данные:* слагаемые, два целых числа. Обозначим их a и b, тип int.

*Выходные данные:* сумма, целое число. Обозначим как s, тип int.

Таблица тестирования:

|  |  |  |
| --- | --- | --- |
| **Входные данные** | **Ожидаемый результат** | **Результат работы программы** |
| а = 2, b = 2 | 4 | 4 |
| а = 2000, b = -2000 | 0 | 0 |
| а = 2000000000, b = 2000000000 | 4000000000 | -294967296 |

Текст программы:

#include <stdio.h>

#include <stdlib.h>

int main()

{

int a, b, s; /\* объявление переменных \*/

printf ("a = "); /\* печать сообщения \*/

scanf ("%d", &a); /\* ввод с клавиатуры вещественного числа и запись его в переменную a \*/

printf ("b = "); /\* печать следующего сообщения \*/

scanf ("%d", &b); /\* ввод с клавиатуры вещественного числа и запись его в переменную b \*/

s = a + b; /\* вычисление суммы и запись ее в переменную s \*/

printf ("%d + %d = %d\n", a, b, s); /\* вывод результата в формате число + число = число \*/

return 0;

}

*Выводы*: При сложении 2000000000 + 2000000000 = -294967296, так как результат данного сложения выходит за диапазон int, это связано с тем, что ожидаемый результат 4000000000 превышает последнее число из диапазона, также результат получился отрицательным из-за того, что после преодоления предела оставшаяся сумма прибавляется к минимальному пределу int.

**Задание 2.**

Написать программу деления одного целого числа на другое.

*Входные данные*: делимое и делитель a, bтип int.

*Выходные данные*: частное s тип int.

Таблица тестирования:

|  |  |  |
| --- | --- | --- |
| **Входные данные** | **Ожидаемый результат** | **Результат работы программы** |
| a = 4, b = 2 | 2 | 2 |
| a = 7, b = 3 | 2, 33 | 2 |
| a = 5, b = 3 | 1,67 | 1 |
| a = 1, b = 2 | 0.5 | 0 |
| a = 5, b = 0 | Сообщение об ошибке | Ошибка -1073741676 (0xC0000094) |

Текст программы:

#include <stdio.h>

#include <stdlib.h>

int main()

{

int a, b, s;

printf("a=");

scanf("%d", &a);

printf("b=");

scanf("%d", &b);

s=a/b;

printf("%d/%d=%d", a, b, s);

return 0;

}

Выводы: в паре чисел, где делимое делится на делитель без остатка, программа выводит ожидаемый результат. В паре чисел, где делимое делится на делитель с остатком, программа выводит только целую часть числа, а дробную часть отбрасывает, потому что программа запоминает только целую часть деления потому что операнды типа int. В паре чисел, где делимое меньше делителя, программа выводит ноль и отбрасывает дробную часть, так как целой частью деления является 0. При делении на ноль выводит ошибку -1073741676 (0xC0000094).

**Задание 3.**

Изменить тип переменных в предыдущей программе на *double* (стандартный вещественный тип). В функциях *scanf()* и *printf()* поменять спецификаторы формата на *%lf*.

*Входные* и *выходные данные* те же, что и в задании 2, обозначения переменных те же, тип всех переменных double.

Таблица тестирования:

|  |  |  |
| --- | --- | --- |
| **Входные данные** | **Ожидаемый результат** | **Результат работы программы** |
| a = 4, b = 2 | 2 | 2.000000 |
| a = 7, b = 3 | 2,33 | 2.333333 |
| a = 5, b = 3 | 1,67 | 1.666667 |
| a = 1, b = 2 | 0,5 | 0.500000 |
| a = 5, b = 0 | Сообщение об ошибке | 1.#INF00 |
| a = 4.2, b = 2.1 | 2 | 2.000000 |
| a = 5.5, b = 2.2 | 2,5 | 2.500000 |
| a = 4.4, b = 0.1 | 44 | 44.000000 |

Текст программы:

#include <stdio.h>

#include <stdlib.h>

int main()

{

double a, b, s;

printf("a=");

scanf("%lf", &a);

printf("b=");

scanf("%lf", &b);

s= a/b;

printf("%lf / %lf=%lf\n", a, b, s);

return 0;

}

Выводы: При делении чисел мы получаем результат с точностью до 6 знаков после запятой, поэтому при делении чисел 7 на 3 и 5 на 3, мы получаем более точный результат по сравнению с ожидаемым. В других случаях все результаты совпадают с ожидаемыми.

При изменении формата вывода на %.8lf выводимое значение стало таким: При выводе чисел, после точки 8 цифр, это позволяет нам получить более точный результат.

При изменении формата вывода на %.2lfвыводимое значение стало таким: При выводе чисел. после точки 2 цифры, поэтому при получении результата, где после запятой больше двух цифр, происходит округление 2 цифры по математическим правилам, например, при делении 5 на 3 мы получим 1.67, а при делении 15 на 8 получится 1.88.

**Задание 4.**

В предыдущей программе изменить тип делимого и делителя обратно на int, результат оставить типа double.

1.До явного приведения.

*Входные данные*: делимое и делитель a, bтип int.

*Выходные данные*: частное s тип int.

Таблица тестирования:

|  |  |  |
| --- | --- | --- |
| **Входные данные** | **Ожидаемый результат** | **Результат работы программы** |
| a = 4, b = 2 | 2 | 2 |
| a = 7, b = 3 | 2, 33 | 2 |
| a = 5, b = 3 | 1,67 | 1 |
| a = 1, b = 2 | 0.5 | 0 |
| a = 5, b = 0 | Сообщение об ошибке | Ошибка -1073741676 (0xC0000094) |

Текст программы:

#include <stdio.h>

#include <stdlib.h>

int main()

{

int a, b, s;

printf("a=");

scanf("%d", &a);

printf("b=");

scanf("%d", &b);

s=a/b;

printf("%d/%d=%d", a, b, s);

return 0;

}

Выводы: в паре чисел, где делимое делится на делитель без остатка, программа выводит ожидаемый результат. В паре чисел, где делимое делится на делитель с остатком, программа выводит только целую часть числа, а дробную часть отбрасывает, потому что программа запоминает только целую часть деления потому что операнды типа int. В паре чисел, где делимое меньше делителя, программа выводит ноль и отбрасывает дробную часть, так как целой частью деления является 0. При делении на ноль выводит ошибку -1073741676 (0xC0000094).

2. С использованием явного приведения.

*Входные данные*: делимое и делитель a, b тип int.

*Выходные данные*: частное s тип double.

Таблица тестирования:

|  |  |  |
| --- | --- | --- |
| **Входные данные** | **Ожидаемый результат** | **Результат работы программы** |
| a = 4, b = 2 | 2 | 2.000000 |
| a = 7, b = 3 | 2,33 | 2.333333 |
| a = 5, b = 3 | 1,67 | 1.666667 |
| a = 1, b = 2 | 0,5 | 0.500000 |
| a = 5, b = 0 | Сообщение об ошибке | 1.#INF00 |

Текст программы:

#include <stdio.h>

#include <stdlib.h>

int main()

{

int a, b;

double s;

printf("a=");

scanf("%d", &a);

printf("b=");

scanf("%d", &b);

s=(double) a/b;

printf("%d / %d = %lf", a, b, s);

return 0;

}

Выводы: Деление двух чисел целочисленного типа и получение результат вещественного типа происходит при помощи явного приведения переменной a к типу double, переменная b приводится не явно. При делении чисел мы получаем результат с точностью до 6 знаков после запятой, поэтому при делении чисел 7 на 3 и 5 на 3, мы получаем более точный результат по сравнению с ожидаемым. В других случаях все результаты совпадают с ожидаемыми.

**Задание 5.**

Проанализировать ошибки при вызове функций scanf() и printf().

|  |  |
| --- | --- |
| **Ошибка** | **Поведение программы** |
| отсутствие & перед именем переменной в scanf() | Программа ничего не выводит, так как программа не сможет записать значение в переменную, & указывает на адрес переменной, на ее место в памяти, туда куда нужно запомнить вводимое значение. |
| наличие & перед именем переменной в printf() при выводе значения переменной | Программа выводит не то число, это связано с тем что при наличии & мы просим программу вывести не значение переменной, а ее адрес. |
| тип спецификатора формата ввода не совпадает с типом переменной: переменная типа *int*, спецификатор *%lf* | Программа ведет себя неопределённо, выводит хаотичное число, отличное от введённого, это связанно с тем, что мы пытаемся поместить в переменную типа int, значение вещественного типа с 8-ми байтовым представлением в 4 байтовое представление. |
| тип спецификатора формата ввода не совпадает с типом переменной: переменная типа *double*, спецификатор *%d* | Программа ведет себя неопределённо, выводит хаотичное число, отличное от введённого, это связанно с тем, что при вводе числа 4-х байтового представления в 8-ми байтовое, целочисленное число заполняет 4 байта из 8, а оставшиеся 4 байта заполняются случайным образом. |
| тип спецификатора формата ввода не совпадает с типом переменной: переменная типа *double*, спецификатор *%f* | Программа ведет себя неопределённо, выводит хаотичное число, отличное от введённого, это связанно с тем, что спецификатор f предназначен для вещественных чисел одинарной точности, а double- вещественное число двойной точности, поэтому запись вещественного числа одинарной точности будет отличаться от его представления в double. |
| тип спецификатора формата вывода не совпадает с типом значения: значение типа *int*, спецификатор *%lf* | Программа выводит неверное число, потому что не может верно представить число типа int, как вещественное, так как представление числа типа double отличается от предоставления числа типа int |
| тип спецификатора формата вывода не совпадает с типом значения: значение типа *double*, спецификатор *%d* | Программа выводит неверное число, даже может вывести отрицательный результат, потому что не может представить корректно вещественное число как целое, представление вещественного числа отличается от целого. Мы просим вывести вещественное число(8 байтовое представление) как целочисленное(4 байтовое представление), поэтому записываются только младшие разряды и число отличается от введённого и может быть отрицательным. |
| количество спецификаторов формата ввода меньше количества вводимых значений переменных | Программа дает ввести значения только в переменные, на которые хватает спецификаторов |
| количество спецификаторов формата ввода больше количества вводимых значений переменных | Программа дает вводить значение столько раз сколько есть спецификаторов, но записывает эти значения только пока есть переменные, то есть мы ввели три числа с клавиатуры, и у нас две переменные, то первое введённое число было присвоено первой переменной, а второе введённое число второй, а третье число было потеряно. |
| количество спецификаторов формата вывода меньше количества выводимых значений | Программа выводит все переменные, на которые хватило спецификаторов, а переменные, на которые не хватило спецификаторов, программа не выводит |
| количество спецификаторов формата вывода больше количества выводимых значений | Программа выводит все переменные, а вместо лишних спецификаторов программа выводит различные числа |

**Задание 6.**

Познакомьтесь с типами данных *char* и *unsigned char*.

#include <stdio.h>

#include <stdlib.h>

#include <limits.h>/\*эта библиотека включает определение характеристик общих типов переменных\*/

int main()

{

char c;

unsigned char uc;

printf("sizeof(c)=%d\tsizeof(uc)=%d\n\n", sizeof(c),sizeof(uc));/\*выводим на экран объём хранения данных в байтах

для переменной типа char и unsigned char\*/

uc=c=CHAR\_MAX;/\* присваиваем переменным uc и с значение 01111111

char\_max(с помощью библиотеки limits.h)- предельное значение типа данных char\*/

printf("CHAR\_MAX : c=%d uc=%d\n", c, uc);/\*выводим на экран максимальное значение в типе данных char равное 127\*/

c = c + 1; uc = uc + 1;/\*прибавляем единицу в типе данных char и получаем минимальное в типе char(01111111+1=11111111=-128),

в типе данных unsigned char 127+1=128\*/

printf("CHAR\_MAX+1 : c=%d uc=%d\n", c, uc);

uc = c = CHAR\_MIN;/\*присваиваем переменным c и uc минимальное значение

в типе данных char\_min(с помощью библиотеки limits.h)

в типе данных unsigned char(0 до 255), поэтому uc получила значение 128,так как(10000000=128 в unsigned char) \*/

printf("CHAR\_MIN : c=%d uc=%d\n", c, uc);

c = uc = UCHAR\_MAX;/\*присваиваем максимальное значение типа unsigned char перменным

uc=255(11111111)(так uc типа unsigned char), c=-1, так как в char 11111111=-1\*/

printf("UCHAR\_MAX : c=%d uc=%d\n", c, uc);

c = c + 1; uc = uc + 1;/\* в переменной с -1+1=0(11111111+1=100000000 первая единица отбрасывается, так как 8 битовое представление),

в uc типа unsigned char 11111111+1=100000000 первой единицы не где храниться, и она отбрасывается(переполнение) uc=0\*/

printf("UCHAR\_MAX+1 : c=%d uc=%d\n", c, uc);

uc = c = -5;/\*присваиваем переменной c и uc -5, с=-5=11111011, uc=11111111-100=11111011=251\*/

printf("-5 : c=%d uc=%d\n", c, uc);

c = -5; uc = 5;/\*присваиваем переменной c=-5 и переменной uc=5\*/

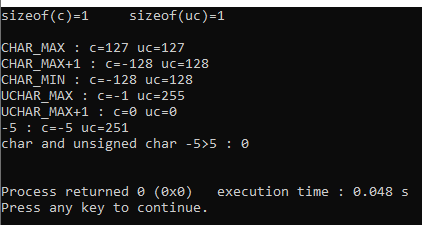
printf("char and unsigned char -5>5 : %d\n\n", c>uc);/\* при сравнении 5 в типе данных unsigned char и -5 в char получаем ложное неравенство между с>uc

все приводится к типу int \*/

return 0;

}

Результаты работы программы:



**Задание 7.**

Познакомьтесь с типами данных *int, short int, long int* и *unsigned int*.

#include <stdlib.h>

#include <stdio.h>

#include <limits.h>

int main()

{

char c;

unsigned char uc;

int i;

unsigned u;

short s;

long l;

printf("sizeof(i)=%d\tsizeof(u)=%d\tsizeof(s)=%d\tsizeof(l)=%d\n\n",

sizeof(i), sizeof(u), sizeof(s), sizeof(l));/\*выводим на экран объем типов данных в байтах\*/

c = s = SHRT\_MAX;/\*присваиваем переменным максимальное значение типа данных short int, в переменную с типа char записывается младший байт(11111111=-1 в типе char)\*/

uc = s;/\*в переменную uc записывается младший байт максимального значения short int (11111111=255 в типе unsigned char)\*/

printf("SHRT\_MAX : c=%d uc=%d s=%d\n", c, uc, s);

s = s + 1;/\*полуичли минимальное значение типа short int 0111111111111111+1=1000000000000000=-32768(первая 1- знаковая) \*/

printf("SHRT\_MAX+1 : s=%d\n", s);

c = s; uc = s;/\*присваиваем 1 байтовым переменным типа char и unsigned char минимальное значение типа short int(1000000000000000) в переменные c и uc записывается только младший разряд (с=0, uc=0)\*/

printf("%d : c=%d uc=%d\n", SHRT\_MIN, c, uc);

s = 0; c = s; uc = s;/\*обнуляем переменные \*/

printf("0 : c=%d uc=%d s=%d\n", c, uc, s);

i = INT\_MAX;/\*присваиваем максимальное значение типа данных int \*/

l = i; u = i;/\* присваиваем максимальное значение типа int переменным l (тип long int) и u (тип unsigned int) \*/

printf("INT\_MAX : i=%d u=%u l=%ld\n", i, u, l);

i = i + 1; l = l + 1; u = u + 1;/\*переменные i и l становятся равны минимальному значению типа int, так как под int и long int выделяется 4 байта(01111111111111111111111111111111+1=10000000000000000000000000000000), и первая цифра в машинном коде отвечает за знак i=l=-2147483648, переменная u при прибавлении 1 увеличилось на единицу u=2147483648 в типе unsigned int(нету единицы под знак) \*/

printf("INT\_MAX+1 : i=%d u=%u l=%ld\n", i, u, l);

i = INT\_MIN;/\*присвоим переменной i минимальное значение типа int \*/

l = i; u = i;/\*присвоим переменной l типа long int минимальное значение типа int (минимум long int совпадет с минимумом int, так, как и int и long int 4 байта и равен-2147483648), переменная u будет равна 2147483648, так как 10000000000000000000000000000000=2147483648 в типе unsigned int\*/

printf("INT\_MIN : i=%d u=%u l=%ld\n", i, u, l);

u = UINT\_MAX;/\*присваиваем u максимальное значение типа unsigned char\*/

i = u; l = u;/\*переменным i и u присваиваем максимальное отрицательное значение (в двоичном коде все 1) 11111111111111111111111111111111 в типе int и long int(4 байта) будет равно -1(первая единица под знак), 11111111111111111111111111111111 максимальное значение в типе unsigned int\*/

printf("UINT\_MAX : i=%d u=%u l=%ld\n", i, u, l);

u = i = -5;/\*переменная i=-5 так как int, переменная u типа unsigned int станет равна 11111111111111111111111111111111 - 100(4) =11111111111111111111111111111011=4294967291\*/

printf("-5 : i=%d u=%u\n", i, u);

i = -5; u = 5;/\*переменная i типа int неявно приводится к типу unsigned int и -5 становится 4294967291 и оно будет> 5(4294967291(-5 после неявного приведения)>5), поэтому выражение истинно\*/

printf("int and unsigned int -5>5 : %d\n", i > u);

c = -5; u = 5;/\*переменная c типа char неявно приводится к типу unsigned int и -5 становится 4294967291 и оно будет > 5(4294967291(-5 после неявного приведения)>5), поэтому выражение истинно\*/

printf("char and unsigned int -5>5 : %d\n\n", c > u);

i = 5.1;/\*при присваивании переменной i типа int программа запоминает только целую часть числа, так как дробную часть числа ей хранить негде\*/

printf("i=5.1 : i=%d\n", i);

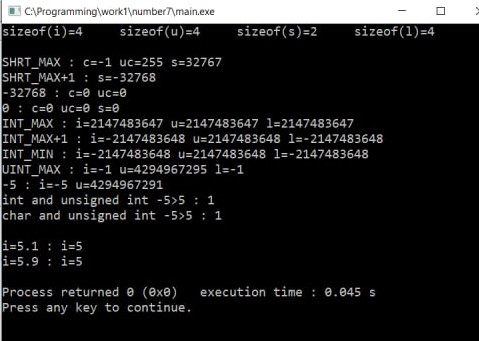
i = 5.9;/\*при присваивании переменной i типа int программа запоминает только целую часть числа, так как дробную часть числа ей хранить негде\*/

printf("i=5.9 : i=%d\n", i);

return 0;

}

Результаты работы программы:



**Задание 8.**

Познакомьтесь с типами данных *float* и *double*.

#include <stdlib.h>

#include <stdio.h>

#include <float.h>

int main()

{

float f;

double d;

printf("sizeof(f)=%d\tsizeof(d)=%d\n\n", sizeof(f), sizeof(d));/\*выводим на экран объем данных типов float и double в байтах\*/

d = f = FLT\_MAX;/\*присваиваем переменным d и f максимальное значение типа float\*/

printf("FLT\_MAX : f=%g d=%g\n", f, d);

d = f = FLT\_MIN;/\*присваиваем переменным d и f минимальное значение типа float \*/

printf("FLT\_MIN : f=%g d=%g\n", f, d);

d = f = FLT\_EPSILON;/\*точность типа float - 7\*/

printf("FLT\_EPSILON : f=%g d=%g\n", f, d);

f = 12345678;/\*присвоим переменной f типа float c точность 7-8 знаков восьмиразрядное число\*/

printf("12345678 : f=%f\n", f);

f = 123456789;/\*присвоим переменной f типа float c точность 7-8 знаков девятиразрядное число, которое будет искажено, из-за недостаточной точности(цифру после 8 негде хранить).\*/

printf("123456789 : f=%f\n", f);

f = 1234567890;/\*присвоим переменной f типа float c точность 7-8 знаков десятиразрядное число, которое будет искажено, из-за недостаточной точности типа float(цифры после 8 негде хранить).\*/

printf("1234567890 : f=%f\n", f);

d = DBL\_MAX;/\*присвоим максимальное значение типа double\*/

printf("DBL\_MAX : d=%g\n", d);

d = DBL\_MIN;/\*присвоим минимальное значение типа double\*/

printf("DBL\_MIN : d=%g\n", d);

d = DBL\_EPSILON;/\*точность типа double до 16 знаков \*/

printf("DBL\_EPSILON : d=%g\n", d);

d = 1e15 + 1;/\*прибавленная 1 сохраняется, так как точность до 16 знаков \*/

printf("1e15+1 : d=%lf\n", d);

d = 1e16 + 1;/\*прибавленная 1 теряется (ее негде хранить, так как тогда число становится 17-ти значным, и 17 цифру программа теряет, точность до 16 знаков\*/

printf("1e16+1 : d=%lf\n", d);

d = 10000 \* 100000 + 1 - 4 \* 250000000;/\*получаем верный результат, так как все операнды целочисленные, получаем целочисленную 1, которая неявно приводится к double \*/

printf("1 : d=%lf\n", d);

d = 1e20 \* 1e20 + 1000 - 1e22 \* 1e18;/\*при вычислении результата данного выражения мы прибавляем к 40-значному числу 1000, которую компьютер не запоминает, так как точность идет до 16 знаков, поэтому на промежуточном действие 1000 была потеряна и в конечном результате получился 0\*/

printf("1000 : d=%lf\n", d);

d = 1e20 \* 1e20 - 1e22 \* 1e18 + 1000;/\*при вычитании одинаковых 40-значных чисел программа получает промежуточный результат 0(ничего не теряет, так как они одинаковые(1е40-1е40=0) и ничего к ним не прибавляется),поэтому при 0+1000, мы получаем 1000, \*/

printf("1000 : d=%lf\n", d);

f = d = 0.3;/\*погрешность из-за точности в float до 7 знаков, в double до 16 знаков, так как мы просим вывести в таком формате(8 и 17 цифр), где происходят искажения\*/

printf("0.3 : f=%.8f d=%.17f\n", f, d);

f = 0;/\*присваиваем переменной f =0\*/

while (f < 10)/\*запускаем цикл, для просмотра за изменением числа f при +0.2 до того момента, пока оно не будет >=10,заметим, что переменная f получает не точно 0.2, а 0.20000003, и то что количество цифр в представление числа 0.2, превышает точность(7 знаков), эта не точность с каждым проходом цикла увеличивается, поэтому на выходе мы получаем не ровное значение 10,а превышающее его 10.199995, из-за данной не точности в моменте когда должен был быть последний проход цикла, значение f=9.99999523,а если бы программа считала без этой погрешности, то f было бы равно 9.8 и цикл был бы завершен, когда f==10\*/

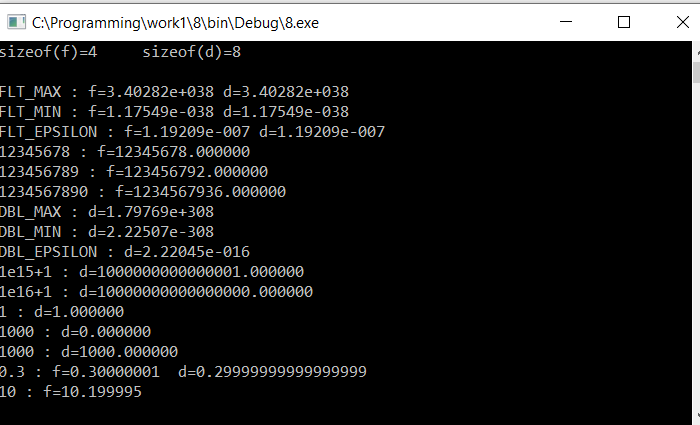
f += 0.2;

printf("10 : f=%f\n", f);

return 0;

}

Результаты работы программы:



**Задание 9.**

Проверить порядок выполнения операций в каждом выражении, содержащем несколько операций присваивания, разделив каждый оператор-выражение на несколько операторов, выполняемых последовательно.

Текст измененной программы:

#include <stdio.h>

#include <stdlib.h>

int main()

{

int a, b = 5, c;

double x, y = -.5, z;

printf("a=");

scanf("%d", &a);

//x = c = a;

c=a;

x=c;

printf("x = c = a : a=%d c=%d x=%f\n", a, c, x);

a += b;

printf("a += b : a=%d\n", a);

//x \*= b + a;

x= x\*(b+a);

printf("x \*= b+a : x=%lf\n", x);

//b += a--;

b=b+a;

a=a - 1;

printf("b += a-- : a=%d b=%d\n", a, b);

//x -= ++c;

c=c+1;

x=x-c;

printf("x -= ++c : c=%d x=%lf\n", c, x);

c = a / b;

printf("c = a/b : c=%d\n",c);

c = a % b;

printf("c = a%%b : c=%d\n",c);

//y += ( a + 1 ) / a++;

y+=(a+1)/a;

a=a+1;

printf("y += (a+1)/a++ : a=%d y=%.3lf\ty=%.0lf\n", a, y, y);

//b = 3 \* ( y -= .6 ) + 2 \* b + 1;

y-=.6;

b=3\*y+2\*b+1;

printf("b = 3\*(y-=.6)+2\*b+1 : b=%d y=%.1lf\n", b, y);

z = a / 2;

printf("z = a/2 : z = a/2 : z=%lf\n", z);

z = (double)a / 2;

printf("z = (double)a/2 : z=%lf\n", z);

//y = ( x = 5.7 ) / 2;

x=5.7;

y=x/2;

printf("y = (x = 5.7)/2 : x=%lf y=%lf\n", x, y);

y = (int)x / 2;

printf("y = (int)x/2 : y=%f\n", y);

//z = ( b - 3 ) / 2 - x / 5 + ( c /= 2 ) + 1 / 4 \* z - y++ + ++b / 3.;

c=c/2;

z=(b - 3)/2 - x / 5 + c;

b=b+1;

z=z+1 / 4 \* z - y + b / 3.;

y=y+1;

printf("z = (b-3)/2 - x/5 +(c/=2) + 1/4\*z - y++ + ++b/3. :\n\

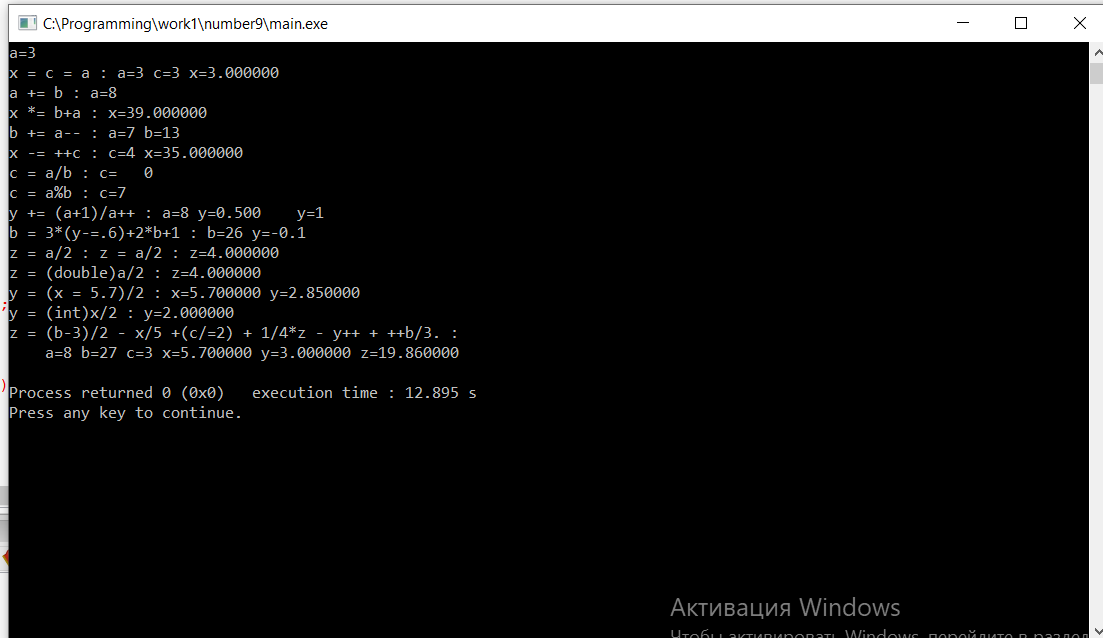
a=%d b=%d c=%d x=%lf y=%lf z=%lf\n", a, b, c, x, y, z);

return 0;

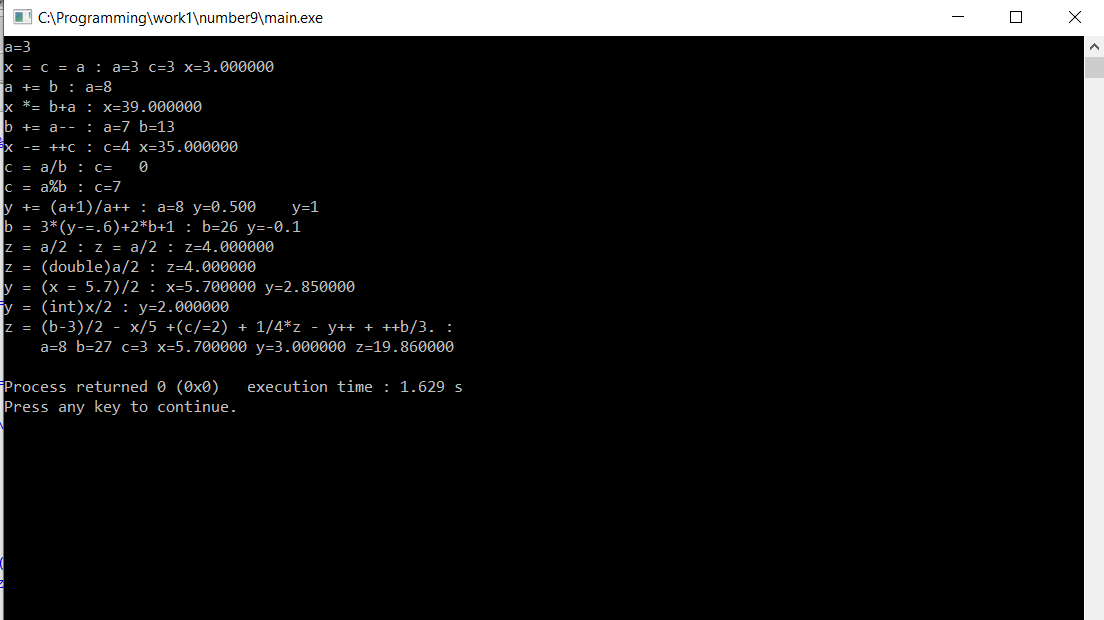
}

Результаты работы программ:

до изменения



после изменения



**Задание 10.**

Написать программу для вычисления значений следующих выражений:  
a=5, c=5

a=a+b-2

c=c+1, d=c-a+d

a=a\*c, c=c-1

a=a/10, c=c/2, b=b-1, d=d\*(c+b+a)

Выражения, записанные в одной строке, записывать одним оператором-выражением, не содержащим запятой. Использовать расширенные операции присваивания, операции инкремента и декремента. Переменные c и d объявить как целые, переменные a и b – как вещественные. Значения переменных b и d вводить с клавиатуры. После вычисления каждого выражения выводить на экран значения всех переменных.

Таблица тестирования:

|  |  |  |
| --- | --- | --- |
| **Входные данные** | **Ожидаемый результат** | **Результат работы программы** |
| b=3 d=8 a=5 c=5 | a=3.6 b=2 c=2.5 d=64,8 | a=3.60 b=2.00 c=2 d=60 |
| b = 5.5 d=10 a=5 c=5 | a=5.1 b=4.5 c=2.5 d=90.75 | a=5.10 b=4.50 c=2 d=81 |
| b=-3.5 d=-3 a=5 c=5 | a=-0.3 b=-4.5 c=2.5 d=-8.75 | a=-0.30 b=-4.50 c=2 d=-8 |

Текст программы:

#include <stdlib.h>

#include <stdio.h>

#include <float.h>

int main()

{

int c, d;

double a, b;

c=a=5;

printf("a=%.2lf c=%d\n", a, c);

printf("b=");

scanf("%lf", &b);

printf("d=");

scanf("%d", &d);

a+=b-2;

printf("a=%.2lf\n", a);

d+=++c-a;

printf("a=%.2lf c=%d d=%d\n", a, c, d);

a\*=c--;

printf("a=%.2lf c=%d\n", a, c);

d\*=(a/=10)+--b+(c/=2);

printf("a=%.2lf b=%.2lf c=%d d=%d\n", a, b, c, d);

return 0;

}