1 Вычисления

Чтобы выполнить какие-либо действия с числами, необходимо их запоминать и записывать для передачи другим. Для этого люди придумывают различные системы обозначения чисел и различные способы их записать: начертания на поверхностях, завязвнные узелки, выкладывания камней, намагничевание и другие способы.

Понятно, что для запоминания и передачи числа (как и любой другой информации) в нашем мире необходим физический носитель. Даже если просто запомнить что-то в уме, нужно иметь как минимум голову, которая это запомнит.

Кроме того, все то, что записано и запомнено, должно быть доступно и для чтения. При этом прочитанная информация должна одинаково восприниматься всеми: и тем, кто ее записал, и теми, кто будет ее читать и использовать далее.

Люди используют носители, позволяющие записать только ограниченный объем информации. Например, в тетрадном листке на одной строке можно записать число, не превышающее 10^{28}, если писать каждую цифру в отдельную клетку.

При записи чисел в тетрадях обычно делают пометки для указания что за величина имелась ввиду (**длина экватора** – 40000 **км., билет на трамвай** – 3 **коп. и т.д.**).

В современных вычислительных системах также используются различные носители памяти. Вся информация, которая может быть записана на них кодируется с помощью чисел. Для каждой записи, выделяется определенное место и количество памяти. К этой памяти необходимо иметь доступ. А при чтении информации нужно быть уверенным, что прочитана вся информация и не прочитано лишнее, иначе мы получим искаженную информацию.

Например, есть известная фраза **«Мы не рабы, рабы немы»**. Если убрать все пробелы (для экономии места) и считать что все слова нам и так известны, то можно прочитать ее как **«Мы не рабы, рабы не мы»**. Слово «немы» прочитано не до конца. А это уже совсем другой смысл.

В программировании используется память, в которой информация может быть перезаписана на одном и том же месте. Необходимо знать где расположено это место в памяти и количество памяти, используемое для одной такой записи. Место в памяти может иметь имя.

Определение. **Переменные**

Переменные служат для записи и изменения чисел. Все другие значения также представляются числами. Для записи разных классов чисел используются разные ТИПЫ переменных. Каждая запись числа занимает определенное место в памяти компьютера. Разные типы переменных могут занимать разное количество памяти. Значения, записанные в память обрабатываются компьютером по-разному в зависимости от типа переменной. У каждой переменной есть определенное место в памяти – АДРЕС. Это место может иметь имя (ИМЯ ПЕРЕМЕННОЙ)

Рассмотрим примеры задач с разными числами и типами переменных.

Задача. Периметр прямоугольника

Необходимо вычислить периметр прямоугольника, заданного величиной его двух соседних сторон. Стороны прямоугольника не превышают значения 10^4.

Для решения напишем программу

Решение:

```
// Начало мантры
#include <stdio.h>
#include <stdlib.h>
// Конец мантры

// Запуск программы начинается с main
int main(){
 int a;     // целое число, имя a
 int b;     // целое число, имя b
 int perem;     // целое число, имя perem
/*
Операции сложения и умножения. Результат
ПРИСВАИВАЕТСЯ perem.
При присваивании значение переменной изменяется
*/
 perem = ( a + b ) * 2;
// Печать результата. Вместо %d будет
// подстaлено значение переменной
 printf("perem=%d\n", perem);
}
```

Проверка результата.

a	b	ожидаемый результат	результат программы	комментарий
10	15	50	50	правильно
0	14	28	28	это не прямоугольник, вопрос к условию
9999	10000	39998	39998	все правильно

Рассмотрим другие типы переменных:

```
// Начало мантры
#include <stdio.h>
#include <stdlib.h>
// Конец мантры

// Запуск программы начинается с main
int main(){
 int a,b;          // целые числа
 float fa;       // дробные числа
// char для символов, могут использоваться
// для записи небольших <=255 целых
 char z;         // символы
 long long al; // длинные целые числа
 double df;     // длинные дробные числа
 int resi;        // целый результат
 float resf;
 long long resl;
 // результат целый
 resi = a * a; //квадрат числа
// результат дробный
 resf = fa +10;
//вывод дробного
 printf("resi=%0.2f\n",resl);
 // результат вычислений - дробный
// а при присваивании - целый
// resl = b / 10;
//вывод длинного целого
 printf("resl=%lld\n",resl);
  z = a % 10;// остаток от деления
// на каждое число 2 знака, левые пустые
```

```
//заменяются  на 0
  printf("%02d\n",z);

}
```

Компиляция и запуск программы:

```
>gcc myprog.c -o myprog.exe
>./myprog.exe
```

Задачи.

Задача С.1.0.1. Середина отрезка

Дано: Отрезок задан целыми числами a и b.

Написать программу, которая вычисляет середину отрезка с округлением по арифметическим правилам

Задача С.1.0.2. Конвертация скоростей

Дано: Скорость задана в формате **км/час.**. Скорость не более 10^5

Написать программу, которая переводиит скорость в формат **м/сек.**

Задача С.1.0.3. Время на часах(I)

Дано: Время отображается на 24-часовом циферблате. Вначале часы показывали h часов, min минут.

Написать программу, которая вычисляет показания часов через h_l часов, min_l минут.

Задача С.1.0.4. Время на часах(II)

Дано: Время отображается на 24-часовом циферблате. Вначале часы показывали h часов, min минут.

Написать программу, которая вычисляет показания часов до того как прошло h_l часов, min_l минут.

Задача С.1.0.5. Длина отрезка

Дано: Отрезок на плоскости задан двумя точками. Координаты точек: (x_1, y_1) и (x_2, y_2) – целые числа. $-10^5 \leq$ число $\leq 10^5$.

Написать программу, котора вычисляет длину этого отрезка. Результат может быть не целым.

Задача С.1.0.6. Бегуны

Дано: По кругу бегают два бегуна. Длина дистанции – L. Скорость первого бегуна – $\frac{z_1}{k_1}L$ в единицу времени, скорость второго – $\frac{z_2}{k_2}L$ в единицу времени. Первый бегун пробежал n кругов.

Написать программу, которая вычисляет сколько раз бегуны встретятся на дистанции.

1. Задачу решить в целых числах (с простыми дробями)
2. Задачу решить в дробных числах.
3. Сравнить результаты для разных n

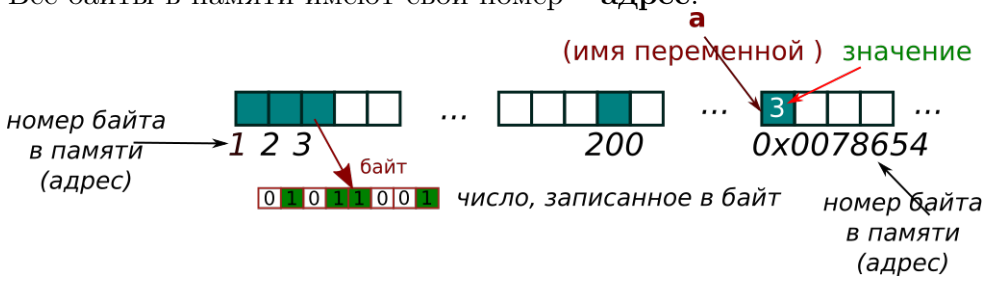
2 Функции и их параметры

2.1 Адреса и указатели

Все значения переменных хранятся в памяти компьютера. Вся память разбита на ячейки – байты. На сегодняшний момент в каждом байте по 8 бит. Бит может принимать значений либо 0, либо 1.

Из нулей и единиц можно составлять числа в двоичной системе счисления.

Все байты в памяти имеют свой номер – **адрес**.



$$0 \times 2^7 + 1 \times 2^6 + 1 \times 2^5 + 0 \times 2^4 + 0 \times 2^3 + 1 \times 2^1 + 0 \times 2^0$$

Переменные разных типов занимают разное количество байт. Например, целые числа, типа **int** занимают 4 байта, а символы типа **char** занимают только один байт.

Можно написать программу для проверки сколько байт занимают те или иные переменные.

```
// Начало мантры
#include <stdio.h>
#include <stdlib.h>
// Конец мантры
```

```
// Запуск программы начинается с main
int main(){
 int a;         // целые числа
 float fa;      // дробные числа
 char z;        // символы
 long long al; // длинные целые числа
 double df;     // длинные дробные числа
 int lin;       // для размера числа
 // Для определения размера воспользуемся sizeof:
 lin = sizeof(a);
 printf("размер int: %d\n", lin);
 lin = sizeof(long long);
 printf("размер long long: %d\n", lin);
}
```

Результат работы программы:

```
>./check_lin
>размер int: 4
>размер long long:8
```

Для указания адреса, то есть номера байта, с которого записаны значения служат переменные – указатели адреса или просто **указатели**.

Для каждого типа переменных – свой тип указателя.

Описываются они так:

```
 int a;      // целое число
// для хранения адреса целого числа
 int *pa;    // указатель на целое число

 float fa;  // дробное число (с плавающей тточкой)
// для хранения адреса числа с плавающей точкой
 float *pfa; // указатель на дробное число.

 char z;     // символ
// для хранения адреса символа
 char *pz;   // указатель на символ
```

Можно передать адрес переменной указателю:

```
// знак "амперсент" - оператор вычисления адреса
 pa = &a; // адрес а присвоен переменной ра
// печать адреса
printf("адрес: %p\n",pa);
```

Можно получить значения переменной, адрес которой находится в переменной:

```
// знак "звездочка" - оператор вычисления значения, переменной по этому адресу
 a = *pa; // адрес a присвоен переменной pa
// печать адреса
printf("адрес: %p значение: %d\n",pa, a);
```

Задача С.2.1.1. Размер

Напишите программу для определения какая переменная **double** или **long long** занимает больше памяти.

Задача С.2.1.2. В два раза

Напишите программу, которая увеличивает в два раза число, находящееся по адресу **pa**

2.2 Функции.

Часть программы можно описать отдельно, дать этой части собственное имя и исполнять как отдельную инструкцию.

Таким образом мы получим **функцию**.

Каждая функция имеет:

- собственное имя,
- набор переменных, которые функции необходимо знать для вычислений,
- значение, которое вычислила функция в процессе своей работы,
- набор инструкций, которые будет выполнять функция

```
// Начало мантры
#include <stdio.h>
#include <stdlib.h>
// Конец мантры

// имя функции hello, переменных нет, значений не вычисляет
 void hello() // интерфейс функции
{
// здесь описание функции - инструкции для выполнения
 printf("Hello\n");
```

```
};

 // Запуск программы начинается с main
int main(){
// вызов функции
  hello();
 }



// Начало мантры
#include <stdio.h>
#include <stdlib.h>
// Конец мантры

/*
имя функции - truncate. Она получает дробную переменную a (аргумент),
а результатом работы функции должно быть округленное по
математическим правилам целое число.
*/
int truncate( float a) // интерфейс функции
{
// переменная float a уже описана как аргумент и является
// внутренней (локальной) переменной.
// имя a - это локальное имя, видимое только внутри функции
// все переменные с таким же именем, описанные в другом месте
// функция trunc не видит
// Во время передачи параметра будет передаваться только значение
// переменной.

  int rez; // локальна япеременная rez

// инструкции
  rez = a + 0.5;
// вернули результат
  return rez;
};

 // Запуск программы начинается с main
int main(){
  float z; // дробное число
```

```
  int result; // целое число для результата

  scanf("%f",&z);
// вызов функции
// вычисленное значение (возвращаемое) присвоим result
// значение переменной z будет присвоено локальной переменной a
  result = truncate(z);
  printf("result=%d\n",result);
 }



// Начало мантры
#include <stdio.h>
#include <stdlib.h>
// Конец мантры

/*
имя функции - add. Она получает два аргумента - целых числа,
а результатом работы функции должна быть сумма чисел

*/
int add( int a, int b) // интерфейс функции
{
// переменные a и b - локальные.
// Передаются значения этих переменных

  int rez; // локальная переменная rez

// инструкции
  rez = a + b;
// вернули результат
  return rez;
};

 // Запуск программы начинается с main
int main(){
  int x, y; // дробное число
  int result; // целое число для результата

  scanf("%d%d",&x,&y);
```

```
// вызов функции
// вычисленное значение (возвращаемое) присвоим result
// значения переменных x и y будет присвоено локальным
// переменным a и b соответственно
  result = add(z);
  printf("result=%d\n",result);
 }
```

Функции-датчики Иногда ответ типа «да» или «нет» требует выполнения множества действий. Для решения таких проблем удобно использовать функции-датчики. Это функции, которые возвращают целое число (0 или 1).

Конструкция **if** в языке C использует 0 как **false**, а все остальные числа как **true**.

Напишем функцию для выяснения делится ли число на 5.

```
 // Начало мантры
#include <stdio.h>
#include <stdlib.h>
// Конец мантры

int is5(int a){
  if(a % 10 == 0 || a % 10 == 5){
// здесь функция сразу завершит работу и вернет 1
    return 1;
  }
// это конец функции.
// Если она не завершилась раньше, значит
// число на 5 на 5 не делится
  return 0;
};
// Запуск программы начинается с main
int main(){
 int digit; // символ для буквы
  scanf("%d",&digit);

  if( is5(digit)){
     printf("%d - делится на 5\n", digit);
  }else{
```

```
    printf("%d - не делится на 5\n",digit);
  }

}
```

Функции с указателями. Если мы передаем в функцию значения переменных, то их значения копируются во внутренние (**локальные**) переменные функциии, а сами переменные не изменяются.

```
// Начало мантры
#include <stdio.h>
#include <stdlib.h>
// Конец мантры

// попробуем поменять значения переменных местами
// функция никаких значений не вычисляет
void swap0( int a, int b) // интерфейс функции
{
// переменные a и b - локальные.
// Передаются значения этих переменных

  int c; // локальная переменная c
  c = a;
  a = b;
  b = c;

  printf("в swap0: a = %d b = %d\n", a,b);
};

 // Запуск программы начинается с main
int main(){
  int x, y; // дробное число

  scanf("%d%d",&x,&y);
  swap0(x,y);
   printf("в main: x = %d x = %d\n", x,y);
 }
```

При запуске убедимся, что x и y не изменились.

Чтобы добиться желаемого результата нужно в вместо значений переменных передать их **адреса**.

```
// Начало мантры
#include <stdio.h>
#include <stdlib.h>
// Конец мантры

// попробуем поменять значения переменных местами
// функция никаких значений не вычисляет
// первый аргумент - указатель (адрес) целого числа
// второй аргумент - указатель (адрес) целого числа
void swap( int *a, int *b) // интерфейс функции
{
// переменные a и b - локальные.

 int c;  // локальная переменная c
 c = *a; // значение по АДРЕСУ a -> c
 *a = *b;// значение по АДРЕСУ b переменной по АДРЕСУ a
 *b = c; // переменной по адресу b присваевается c

// для отладки. Потом нужно убрать
  printf("в␣swap:␣a␣=␣%d␣b␣=␣%d\n", a,b);
};
 // Запуск программы начинается с main
int main(){
  int x, y; // дробное число

  scanf("%d%d",&x,&y);

  // передаем АДРЕСА
  swap(&x,&y);
   printf("в␣main:␣x␣=␣%d␣y␣=␣%d\n", x,y);
 }
```

Задачи.

Задача С.2.2.1. Температура

Написать функцию **int temFC(int faren)**, которая переводит температуру, измеренную по Фаренгейну, в температуру, измерянную по Цельсию. Измерения - в целых числах.

Задача С.2.2.2. Длина

Отрезок на прямой задан своими координатами (float). Написать функцию **int middle(int x1, int x2)**, которая вычисляет длину отрезка.

Задача С.2.2.3. Отражение точки

Написать функцию **void mirror(int* x1, int *y1)**, которая отображает отрезок зеркально относительно оси OY.

Задача С.2.2.4. Отражение отрезка

Дано: Отрезок на плоскости задан координатами своих вершин.

Написать функцию **void mirrorL(int* x1, int *y1, int* x2, int* y2)**, которая отображает отрезок зеркально относительно оси OX.

Задача С.2.2.5. Треть

Дано: Отрезок на прямой задан своими координатами (float). Написать функцию **void oneTr(float* x1, float* x2)**, которая «обрезает» отрезок сначала на $\frac{1}{3}$ и с конца на $\frac{1}{3}$. При этом x1 и x2 получают новые координаты.

Задача С.2.2.6. Расстояние

Написать функцию **float getDistance(int x1, int y1, int x2, int y2)**, которая вычисляет расстояние между двумя точками на плоскости. Координаты точек - целые.

Задача С.2.2.7. Площадь треугольника

Дано: Треугольник задан координатами вершин на плоскости (целые числа, не более 10^5).

Написать функцию **float triaS(int x1, int y1, int x2, int y2, int x3, int y3)**, которая вычисляет площадь треугольника.

Задача С.2.2.8. Лед

Дано: Кусок льда, весом m_1 кг. и температурой T_1 положили в воду, температурой T_2. Удельная теплота плавления льда $3.3 \cdot 10^5$ Дж/кг, удельная теплоемкость – 2100, удельна теплоемкость воды – 2400.

Написать функцию **float timeIce(float m1, float T1, float T2)**, которая вычисляет время таяния льда. При выводе результата время конвертироать в часы и минуты.