acm.mipt.ru

олимпиады по программированию на Физтехе

Раздел «Язык Си» . NewCoffeFunc :

- Функции и их параметры
  - Адреса и указатели
    - Задачи
      - Задача 1. Размер
      - Задача 2. В два раза.
  - Функции.
    - Самая простая функция
    - Округление числа (математическое)
    - Сложение двух чисел.
    - Функции-датчики
    - Функции с указателями.
    - Задачи
      - Задача 3. Температура
      - Задача 4.Середина
      - Задача 5.Расстояние
      - Задача 6. Треть
      - Задача 7. Площадь треугольника
      - Задача 8. Лед
      - Задача 9. Максимум
      - Задача 10. Счастливый билет (I)
      - Задача 11. Восточное время.
      - Задача 12. Внутри или снаружи
      - Задача 13. Поворот
      - Задача 14. Шахматный конь.
      - Задача 15. Счастливый билет (II).

Функции и их параметры

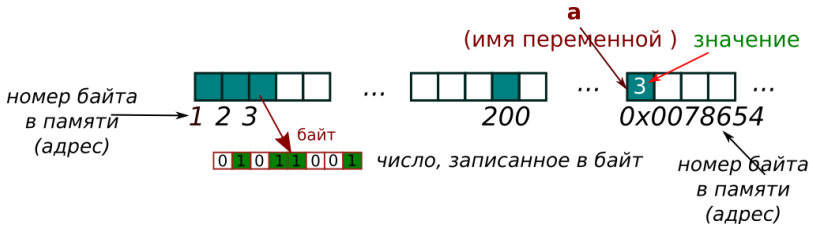
Адреса и указатели

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.

Байт – это минимальная адресуемая часть памяти. Нельзя просто так обратиться к биту.

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** для нашего компилятора **gcc** занимают 4 байта, а символы типа **char** занимают только один байт.

В различных системах размер для переменных конкретных типов может быть разным.

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

```
// 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
```

Видим, что целые числа типа **int** занимают **4** байта, а **long long** – **8**.

Адресом переменной считается с первого байта, в который будет помещаться значение этой переменной.

Адрес также можно поместить в специальную переменную, чтобы можно было его вычислять, изменять и т.д.

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 точкой)
// 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 a присвоен переменной pa

// печать адреса
// %p - печать адреса (номера первого байта переменной) в
// шестнадцатеричном виде
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; // значение (число) по адресу в pa присваивается переменной a
// печать адреса
printf("адрес: %p значение: %d\n",pa, a);
```

Напишем небольшую программу с указателями.

```
#include <stdio.h>
#include <stdlib.h>

int main(){
// создадим переменную для беззнакового целого.
// как мы уже видели, для целой переменной int нужны 4 байта
  unsigned int a;
// это указатель для целой переменной
// если увеличить pa на 1, то она будет указывать на
// СЛЕДУЮЩУЮ ПЕРЕМЕННУЮ после целой
// Значит, адрес увеличится на 4.
  int *pa;

// Эта переменная указывает на число,
// которое занимает 1 байт памяти
  char *pc;

// В каждом байте переменной типа unsigned int
// можно записать число от 0 до 255, если у нас 4 байта, то получается 4-значное
// число по основанию 256.
// запишем такое число: 2.7.5.3
  a = 3+256*(5+256*(2+256*7));
// передадим адрес переменной a в переменнную pa
  pa =&a;

// преобразуем указателя на int (4 байта)
// в указатель на char (1 байт).
  pc = (char*)pa;

// напечатаем значение a
  printf("a=%d\n",a);

// напечатаем значение первого, второго, третьего
// и четвертого байта переменной a;
// результат выведем в восьмеричном виде

// Первый байт имеет адрес pa и pc. Но, если увеличить pa (pa + 1), то
// pa увеличится сразу на 4 байта (int), поэтому  будем пользоваться
// pc (char, то есть 1 байт). pc + 1 увеличит адрес на 1 байт.
// a0 - первый байт, a1 - второй, a2 - третий, a3 - четвертый
  printf("a0=%o\n  a1=%o\n a2=%o\n a3=%o\n",*pc,*(pc + 1),*(pc + 2),*(pc + 3));
  return 0;
}
```

Результат работы программы

> *a=117572867*	вычисленное значение a
> *a0=3*	значение в первом байте (восьмеричное представление)
> *a1=5*	значение во втором байте (восьмеричное представление)
> *a0=7*	значение в третьем байте (восьмеричное представление)
> *a0=2*	значение в четвертом байте (восьмеричное представление)

Как видно, число записывается в «обратном» порядке.

Вообще порядок записи в память компьютера зависит от типа процессора.

«Обратную» запись числа удобно использовать при решении задач с «длинными» числами.

Задачи

Задача 1. Размер

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

Задача 2. В два раза.

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

Функции.

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

Таким образом мы получим **функцию** .

Каждая функция имеет:

1. собственное имя,
2. набор переменных, которые функции необходимо знать для вычислений,
3. значение, которое вычисляет функция в процессе своей работы,
4. набор инструкций, которые выполняет функция

Самая простая 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");
};
// Объявление или описание каждой функции должно заканчиваться ;

 // Запуск программы начинается с main
int main(){

// ВЫЗОВ функции
//  с этого места начнут выполняться инструкции функции
  hello();
// с этого места будут выполняться иструкции main

  return 0;
 }
```

Округление числа (математическое)

Рассмотрим функцию с одним параметром (аргументом). Эта функция должна самостоятельно округлять десятичную дробь до целого по правилам математики. То есть, **3.2 -> 3**, а **4.6 -> 5**.

```
// 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 truncate не видит
// Во время передачи параметра будет передаваться только значение
// переменной.

  int rez; // локальная переменная rez

// инструкции
  rez = a + 0.5;
```

```
// вернули результат
  return rez;
};

// Запуск программы начинается с main
// main() - тоже функция :)
int main(){
  float z; // дробное число
  int result; // целое число для результата

  scanf("%f",&z);
// вызов функции
// вычисленное значение (возвращаемое) присвоим переменной result
// значение переменной z будет присвоено локальной переменной a
  result = truncate(z);
// после выполнения инструкций функции trancate() выполняются инструкции main()
// и первая инструкция - это присваивание полученного от truncate()
// значения переменной result
  printf("result=%d\n",result);
 }
```

В функцию можно передать больше одного параметра (аргумента).

Сложение двух чисел.

```
// 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
*/

// Аргументы функции записываются так: тип передаваемого значения,
// имя локальной (для функции) переменной.
// Далее через запятую так указываются все передаваемые аргументы.

// Запомним, что при передаче не важны имена переменных, значения которых передаются,
// а ВАЖЕН ПОРЯДОК.
// Значение первой указанной в вызове переменной будет передано a, а второй – b

int add( int a, float b) // интерфейс функции
{
// переменные a и b - локальные.
// В эти переменные передаются значения при вызове функции

  int rez; // локальная переменная rez

// инструкции
// округлим b и преобразуем результат к целому значению.
  rez = a + (int)(b + 0.5);
// вернем результат
  return rez;
};

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

  scanf("%d%f",&x,&y);
// вызов функции
// вычисленное значение (возвращаемое) присвоим result
// значения переменных x и y будет присвоено локальным
// переменным внутри функции a и b соответственно
// a - первая указанная в описании переменная получит значение первой переменной в вызове (ожидается int),
// то есть x, a b - вторая переменная в описании (ожидается float) получит значение второй переменнной в вызове (y).
  result = add(x,y);
  printf("result=%d\n",result);
 }
```

При необходимости функции можно передавать и большее количество параметров. При этом правила передаче значения будут таким же: типы и значение передаваемых в функцию переменных должны соответствовать порядку этих переменных в описании функции.

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**. Имеются ввиду целые числа.

Напишем функцию для выяснения делится ли число на 5.

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

// Если число делится на 5, то функция вернет число, отличное от 0, у нас это 1
```

```
int isDiv5(int a){
  if(a % 10 == 0 || a % 10 == 5){
// здесь функция сразу завершит работу и вернет 1
    return 1;
// после return дальнейшие инструкции функции выполняться не будут.
  }
// 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 получает значение, которое вернет функция isDiv5().
// если это значение 0, то if считает, что получил false.
// Если не 0 (у нас 1), значит true
  if( isDiv5(digit)){
     printf("%d - делится на 5\n", digit);
  }else{
    printf("%d - не делится на 5\n",digit);
  }

}
```

Функции-датчики очень удобны, когда для выяснения соответствия чему-либо нужно выполнить множество инструкций, а иногда и со сложными алгоримами.

Функции с указателями.

Если мы передаем в функцию значения переменных, то значения передаются локальным переменным функциии, а сами, переменные указанные в вызове, не изменяются.

Тем не менее иногда требуется написать функции, которые должны именно менять значение переданных переменных.

Попробуем написать функцию, которая меняет переменные местами.

```
// 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);
   return 0;
 }
```

При запуске убедимся, что x и y меняются. Результат работы программы

> 7 9	a будет 7, b будет 9
> в swap0: a = 9 b = 7	значение, локальных переменных в функции swap0
> в main: x = 7 x = 9	значение переменных в main(). Как видим, они не поменялись значениями

Обратим еще внимание на то, что printf() и scanf() – тоже функции. При этом функция scanf() меняет значения передаваемых ей переменных. Значит, она тоже требует адреса переменных.

Чтобы добиться нужного 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 swap1( 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 swap1: a = %d b = %d\n", a,b);
};
 // Запуск программы начинается с main
int main(){
  int x, y; // целые числа, которые будем менять
// получили значения.
  scanf("%d%d",&x,&y);

  // передаем АДРЕСА
  swap1(&x,&y);
   printf("в main: x = %d y = %d\n", x,y);
 return 0;
 }
```

Запускаем программу Результат работы программы

> 7 9	a будет 7, b будет 9
> в swap1: a = 9 b = 7	значение, локальных переменных в функции swap1
> в main: x = 9 y = 7	значение переменных в main(). Как видим, они не изменились

Задачи

Задача 3. Температура

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

Формула для перевода: Градусы по Цельсию = (градусы по Фаренгейту - 32) / 1.8

Задача 4.Середина

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

Задача 5.Расстояние

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

Примечание. Для вычисления квадратного корня используется уже существующая функция **double sqrt(double)**.

Пример ее использования

```
#include <stdio.h>
#include <stdlib.h>
// добавили математику
#include <math.h>
// Конец мантры

int main(){
  int x; //
  float rez;
  scanf("%d",&x);
// вызов sqrt()
  rez = sqrt(x);
// печать результата (4 знака после запятой)
  printf("rez: %0.4f\n", rez);
 return 0;
 }
```

Копмпиляция и запуск программы. При компиляции нужно дрбавить **-lm**

```
> gcc sq.c -o sq -lm
> ./sq
25
rez: 5.0000
```

Задача 6. Треть

Отрезок на прямой задан своими координатами (**float**).

Написать функцию **void oneTr(float* x1, float* x2)**, которая "обрезает" отрезок сначала на **1/3** и с конца на **1/3**. При этом **x1** и **x2** получают новые координаты.

Задача 7. Площадь треугольника

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 8. Лед

Кусок льда, весом **m_1** кг. и температурой **T_1** положили в воду, температурой **T_2**. Удельная теплота плавления льда ==3.3x10^5 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.

Задача 9. Максимум

Написать функцию **float max(float a, float b)** , которая получает два дробных числа и возвращает большее из них.

Задача 10. Счастливый билет (I)

Написать функцию **int isLuckyTicket(long long a)**, которая определяет, является ли этот шестизначный билет счастливым. (Счастливым считается билет, у которого сумма первых трех цифр его номенра равна сумме последних трех цифр. Например, 123321 - счастливый)

Задача 11. Восточное время.

Время в 24-часовом формате задано часами и минутами. Написать функцию **void toMidnight(int h, int min)**, которая печатает время в старинном восточном формате:

Мышь	23.40 - 01.40
Бык	01.40 - 3.40
Тигр	3.40 - 5.40
Заяц	5.40 - 7.40
Дракон	7.40 - 9.40
Змея	9.40 - 11.40
Лошадь	11.40 - 13.40
Овца	13.40 - 15.40
Обезьяна	15.40 - 17.40
Петух	17.40 - 19.40
Собака	19.40 - 21.40
Свинья	21.40 - 23.40

Задача 12. Внутри или снаружи

Стороны прямоугольника параллельны осям координат. Прямоугольник задан своими вершинами (по диагонали). Точка лежит на плоскости. Все координаты целые числа (-200 ≤ x,y ≤ 200).

Написать функцию-датчик **int isInside(int x1, int y1, int x2, int y2, int px, int py)**, которая выясняет лежит ли точка внутри прямоугольника (включая границы). Первые 4 переменные – координаты вершин прямоугольник, последние 2 – координаты точки.

Задача 13. Поворот

Стороны прямоугольника параллельны осям координат. Прямоугольник задан своими вершинами (по диагонали). Все координаты целые числа (-200 ≤ x,y ≤ 200).

Написать функцию **void Rotate90L(int *x1, int *y1, int *x2, int *y2)**, которая поворачивает прямоугольник влево на 90^0 относительно его центра.

Задача 14. Шахматный конь.

На шахматной доске стоит одинокий конь. Его координаты заданы буквой и числом без пробелов (например, a8). Написать функцию-датчик **int canGO(char *horseLet, int *horseDigit, char tryLet, int tryDigit)**, которая выясняет может ли конь пойти на клетку с заданными координатами. Если возможно пойти на заданную клетку, координаты коня меняются – совершается ход. Координаты клетки и позиция коня задаются одинаково: <число> <буква>

Задача 15. Счастливый билет (II).

Дан 6-значный счастливый билет. Написать функцию **long long Next(long long bilet)**, которая возвращает номер ближайшего следующего счастливого билета по возрастанию номеров. Циклы не использовать!!

-- TatyanaOvsyannikova2011 - 10 Oct 2016

Attachment	Action	Size	Date	Who	Comment
addr1.png	manage	39.3 K	10 Oct 2016 - 13:55	TatyanaOvsyannikova2011	
form1.png	manage	6.9 K	10 Oct 2016 - 13:55	TatyanaOvsyannikova2011	