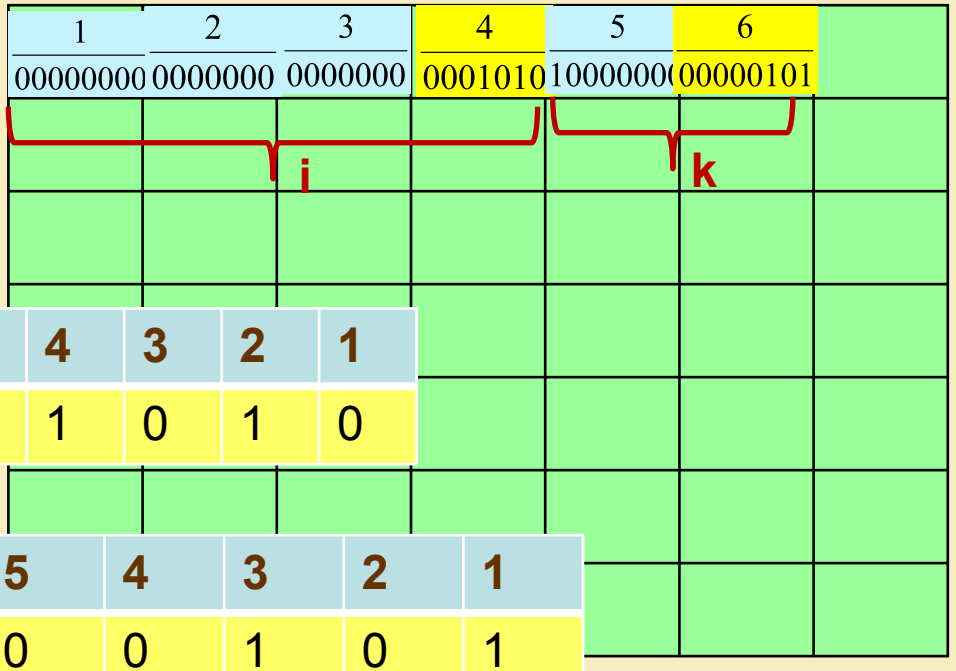
Функции, указатели, ссылки

Взаимоотношение имени переменной и её адреса в ОП

Текст программы

- int i = 10; // $10_{10} = 1010_2$
- short int k = -5; // $|5_{10}| = 101_2$

Оперативная память, каждая ячейка – 1 байт



32	31	30	…	9	8	7	6	5	4	3	2	1
0	0	0	…	0	0	0	0	0	1	0	1	0

16	15	14	…	8	7	6	5	4	3	2	1
1	0	0	…	0	0	0	0	0	1	0	1

Когда компилятор обрабатывает оператор определения переменной, например, *int i=10;* он выделяет память в соответствии с типом (*int*) и инициализирует её указанным значением (*10*).

Указатели

Когда компилятор обрабатывает оператор определения переменной, например int i = 10; он выделяет память в соответствии с типом (int) и инициализирует её указанным значением (10). Все обращения к переменной по её имени (i) *заменяются компилятором на адрес области памяти*, в которой хранится значение переменной.

Программист может определить собственные переменные для хранения адресов областей памяти. Такие переменные называются *указателями.*

Указатели, следовательно*, предназначены для хранения адресов областей памяти.*

В C^{++} различают три вида указателей – указатели на объект, на функцию и на void, отличающиеся свойствами и набором допустимых значений.

Описание указателя

Указатель – это объект, содержащий адрес начала области памяти, где хранится значение переменной.

Так, если **p** содержит адрес **x**, то говорят, что **p** **указывает** на **x**.

Переменная-указатель *объявляется* следующим образом:

тип имя_переменной;

Здесь ***тип*** определяет, тип данных на которые будет указывать этот указатель.

Пример:

int ip; // ip указатель на int

float fp; // fp указатель на float

Размер указателя зависит от модели памяти,
можно определять указатель на указатель и т.д.
Указатель может быть константой или переменной,
а также указывать на константу или переменную.

Например:

```
int L;                          // целая переменная
const int ci = L;               // целая константа
int * pi ;                      // указатель на целую
                                переменную
const int * pci;                // указатель на целую
                                константу
int * const cp = & i;           // указатель-
                                константа на целую
                                переменную
const int * const cpc = & ci;   // указатель-
                                константа на целую
                                константу
```

С указателями можно выполнять следующие операции:

- *разыменование* (*) – получение значения величины, адрес которой хранится в указателе;

- взятие адреса (&);

- присваивание;

- арифметические операции

- сложение указателя только с константой,

- вычитание: допускается разность указателей и разность указателя и константы,

- инкремент (++) увеличивает значение указателя на величину sizeof(тип);

- декремент (--) уменьшает значение указателя на величину sizeof(тип);

-сравнение;

-приведение типов.

Краткие итоги:

- *Для экономии памяти и времени, затрачиваемого на обращение к данным, в программах используют указатели на объекты.*
- *Указатель не является самостоятельным типом, он всегда связан с другим типом.*
- *Указатель может быть константой или переменной, а также указывать на константу или переменную.*
- *Указатель типа void указывает на область памяти любого размера. Разыменование такого указателя необходимо проводить с операцией приведения типов.*
- *До первого использования в программе объявленный указатель необходимо проинициализировать.*
- *Над указателями определены операции: разыменование, взятие адреса, декремент, инкремент, увеличение (уменьшение) на целую константу, разность, определение размера.*
- *Над указателями определены операции сравнения.*

Ссылки

Ссылки представляют собой *синоним имени*, указанного при инициализации ссылки. Ссылку можно рассматривать как указатель, который всегда разыменовывается. Формат объявления ссылки:

Тип & Имя;

где Тип это тип величины, на которую указывает ссылка, & — оператор ссылки, означающий, что следующее за ним Имя является именем переменной ссылочного типа, например:

int kol;	
int & pal = kol;	**// ссылка pal - альтернативное имя для kol**
const char& CR = ' \ n ';	**// ссылка на константу**

Правила работы со ссылками

- Переменная-ссылка должна явно инициализироваться при ее описании, кроме некоторых случаев, когда она является, например, параметром функции.
- После инициализации ссылке не может быть присвоена другая переменная.
- Тип ссылки должен совпадать с типом величины, на которую она ссылается.
- Не разрешается определять указатели на ссылки, создавать массивы ссылок и ссылки на ссылки.

Ссылки применяются чаще всего в качестве параметров функций и типов возвращаемых функциями значений.

Пример на применение операции разыменования

Это означает, что по адресу переменной в памяти мы переходим к действиям над значением, хранимом по данному адресу. Это операция и называется разыменованием.

```
int main()
{
    int  i_val = 7;
    int* i_ptr = &i_val; //Используя унарную операцию
            //взятия адреса &, мы извлекаем адрес переменной
            //i_val и присваиваем ее указателю.

    // выведем на экран значение переменной i_val
    cout <<  i_val << endl; //используем саму переменную
    cout << *i_ptr << endl; //обращаемся к значению переменной
                        // i_val через указатель: здесь используется
                        //операция разыменования:
                        //она позволяет перейти от адреса к значению.
        system("pause");
  return 0;
}
```

Операторы указателей

В предыдущем примере для работы с указателями предусмотрены два специальных оператора: **&** и □

Оператор **&** является унарным: он возвращает адрес ячейки памяти, в которой находится значение переменной.

Пример: Если **ptr** является указателем (например, int □ptr), то оператор **ptr = &total;** помещает в **ptr** адрес ячейки памяти, где находится переменная **total**.

Операция **&** возвращает адрес переменной, перед которой она указана.

Оператор □, является обратным по отношению к **&**.

Этот унарный оператор возвращает значение переменной, расположенной по адресу, который указан в качестве операнда этого оператора.

Пример.

ptr = &total; *// указатель ptr получает значение адреса переменной total*

value = □ptr; *// value получает значение, расположенное по адресу ptr*

// т.е. значение total

Операцию □ можно назвать "**по адресу**"

ПРИМЕР: программа выполняет последовательность описанных выше операций.

```
#include <iostream>
using namespace std;

int main( )
{
      int  total;
      int  *ptr;    // указатель на int
      int  val;
      total = 3200;  // присвоим total значение 3200
      ptr = &total;   // получим адрес total
      val = *ptr;      //  получим значение по этому адресу
      cout  <<  "val  =  "   << val   << '\n';
      return 0;
}
```

```
val =  3200
```

Возврат нескольких значений из функции

Для того чтобы и в вызывающей программе и в функции работать с одной и той же переменной необходимо осуществлять передачу в функцию адреса памяти, где размещена данная переменная.

Такая передача называется **передачей по ссылке** (вместо передачи по значению). Это достигается с помощью **параметра-ссылки.**

Для этого в определении функции и в прототипе перед именем соответствующей переменной необходимо поставить знак операции **&**, возвращающей адрес переменной, перед которой она указана.

При использовании параметра-ссылки в функцию передается адрес (а не значение) аргумента.

Внутри функции при операциях над параметром-ссылкой автоматически выполняется снятие ссылки, поэтому нет необходимости указывать при аргументе оператор &.

В следующей программе в функции вычисляются объем и площадь коробки и передаются в основную функцию.

```
#include <iostream>
using namespace std;
void box(int  length,  int  width,  int  height,  int  &vol,  int &ar);
int main( )
{      int volume;
       int area;
       box(7, 20, 4, volume, area); cout << volume << "  " << area << endl;
       box(50, 3, 2, volume, area);  cout << volume << "  " << area << endl;
       box(8, 6, 9, volume, area);    cout << volume << "  " << area << endl;
       return 0;
}
void box(int  length,  int  width,  int  height,  int   &vol,  int &ar)
{
       vol = length * width * height;
       ar = length * width;
       return;
}
```

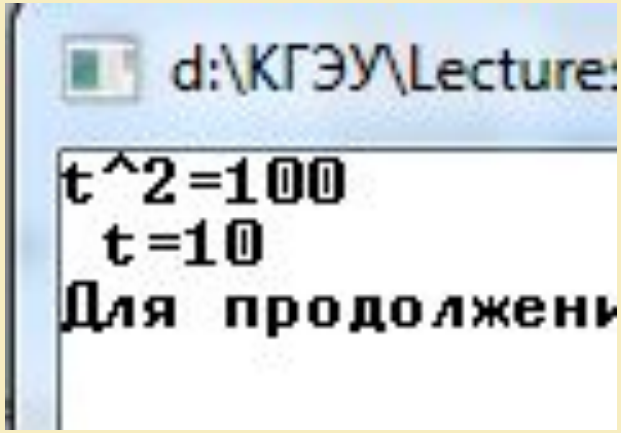
Резюме:

При передаче фактических аргументов *по значению* в вызываемой функции создаются *копии* передаваемых значений. Поэтому любые операции с соответствующими формальными параметрами в теле функции *не изменят фактические значения в вызывающей функции*.

При передаче *по ссылке*, вызываемая функция принимает адрес той переменной, которая описана в вызывающей программе, поэтому все операции в теле функции *приводят к изменению значения переменной*, переданной в качестве фактического значения.

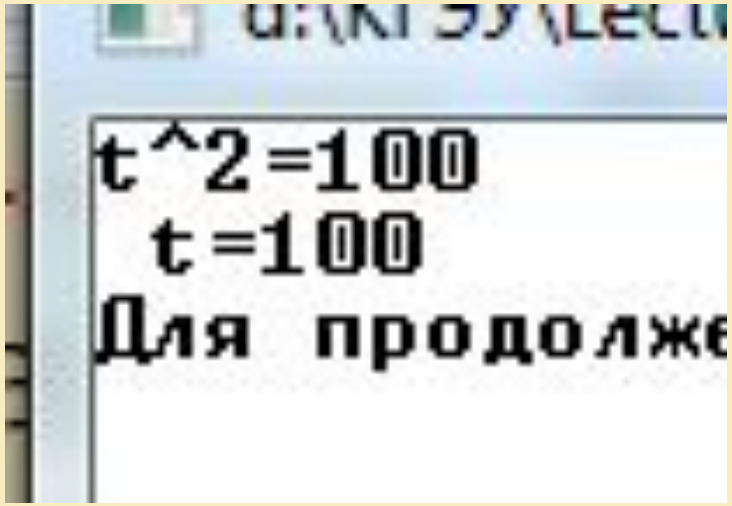
Пример передачи параметров по значению

```
#include<iostream>
#include<ctime>
using namespace std;

int sqr (int  x);
int main(void)
{
int t=10;
cout<<"t^2="<<sqr(t)<<endl;
cout<<" t=" <<t<<endl;
system("pause");
return 0;
}
int sqr (int  x) {
x = x*x; return x;
}
```



Пример передачи параметров по ссылке

```
#include<iostream>
#include<ctime>
using namespace std;

int sqr (int & x);
int main(void)
{
int t=10;
cout<<"t^2="<<sqr(t)<<endl;
cout<<" t=" <<t<<endl;
system("pause");
return 0;
}
int sqr (int & x) {
x = x*x; return x;
}
```



Примеры применения функций

Даны два вектора с координатами {1,-2,0}, {2, 7,-4}. Найти модуль каждого вектора, сумму векторов и их скалярное произведение.

```
int main()
{
    int a,b,c, x,y,z, u,v,w;//переменные для координат3-х векторов
    float mod_1, mod_2, scal_pr;// модули векторов и их скалярное произведение
cout<<" Input 3 coordinates of the vector:";cin >>a>>b>>c;
cout<< "Vector: "<<'\t'<<a<<'\t'<<b<<'\t'<<c<<endl;
cout<<" Input 3 coordinates of the vector:";cin >>x>>y>>z;
cout<< "Vector: "<<'\t'<<x<<'\t'<<y<<'\t'<<z<<endl;
```

```
mod_1= sqrt(float(a*a+b*b+c*c));// модуль первого вектора
mod_2= sqrt(float(x*x+y*y+z*z));// модуль второго вектора
     cout<<"\nmod_1 = "<< mod_1<<endl;
     cout<<"\nmod_2 = "<< mod_2<<endl;

     u=a+x; v=b+y;w= c+z;// определение суммы векторов
    cout<<”\n new”;
    cout<< "Vector: "<<'\t'<<u<<'\t'<<v<<'\t'<<w<<endl;


     scal_pr=(a*x+b*y+c*z)/mod_1/mod_2;
    cout<< "\nscal_pr: "<<scal_pr<<endl;
system("pause");
     return 0;
}
```

Решение задачи в рамках структурного подхода

```
#include <iostream>
#include <cmath>
using namespace std;
 void inp_vect(int & a, int & b, int & c)// функция для
      ввода вектора с консоли
 {cout<<"\n Input 3 coordinates of the vector:";cin >>a>>b>>c;}
 void print(int a, int b, int c)// функция для вывода
      вектора на консоль
 {cout<< "\nVector: "<<'\t'<<a<<'\t'<<b<<'\t'<<c<<endl;}
 float modul(int a, int b, int c)// функция  вычисления
                 модуля вектора
 {return sqrt(float(a*a+b*b+c*c));}
```

```
 int main()
{
int a,b,c, x,y,z, u,v,w; //переменные для координат3-х
      векторов
float mod_1, mod_2, scal_pr; // модули векторов и их скалярное
                произведение


      inp_vect(a,b,c); print(a,b,c); // ввод и вывод векторов
      inp_vect(x,y,z); print(x,y,z);

      mod_1= modul(a,b,c); // модуль первого вектора
      mod_2= modul(x,y,z); // модуль первого вектора

      cout<<"\nmod_1 = "<< mod_1<<endl;
      cout<<"\nmod_2 = "<< mod_2<<endl;

u=a+x; v=b+y;w= c+z; // определение суммы векторов
    cout<<"\n new "; print(u,v,w);
```

```
scal_pr=(a*x+b*y+c*z)/(modul(a,b,c)*modul(x,y,z)); // обращение к функции внутри вычисляющего оператора

    cout<< "\ns
system("pause
    return 0;
}
```

```
 Input 3 coordinates of the vector: 1 -2 0
Vector:         1       -2      0
 Input 3 coordinates of the vector: 2 7 -4
Vector:         2       7       -4
mod_1 = 2.23607
mod_2 = 8.30662
 new
Vector:         3       5       -4
scal_pr: -0.646058
Для продолжения нажмите любую клавишу . . .
```

Описание к программе

По сути, после компиляции не будет никакой разницы для процессора, как для первого кода, так и для второго. Однако программисту после создания отдельных функций - для ввода/вывода вектора, вычисления его модуля - код главной программы стал,

во-первых, более читабельным,

во-вторых, при необходимости изменить формат вывода на консоль достаточно сделать изменения лишь в функции, и эти изменения распространятся на вывод любого количества векторов,

в-третьих, при вычислении скалярного произведения нет необходимости заранее вычислять модуль вектора и записывать его значение в от-дельную переменную: при наличии функции modul, её можно включить непосредственно в математическое выражение так, как это делается обычно со стандартными функциями.