数据在内存中的存储

程序在运行的时候，系统根据程序中定义变量的类型，在计算机的内存中给变量分配一定长度的空间。

一般来说，字符型占 1 个字节，整型数占 4 个字节，双精度浮点数占 8 个字节，.....。

内存中的每个字节都有一个唯一的编号，我们称之为地址。

图1. 数据在内存中的存储

变量的直接访问和间接访问

【变量的直接访问】

像下面的代码，按变量名存取变量的值，就叫做变量的直接访问。

```
1.  i = 100;
```

这个操作实际上就是把数值 100 存放到定义 i 单元的地址中。

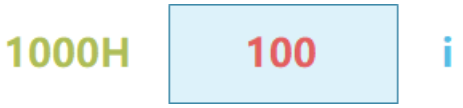


图2. 变量的直接访问

【变量的间接访问】

将变量 i 的地址存放在另一个单元 p 中，通过 p 取出变量 i 的地址，再针对变量 i 操作，就叫做变量的间接访问。

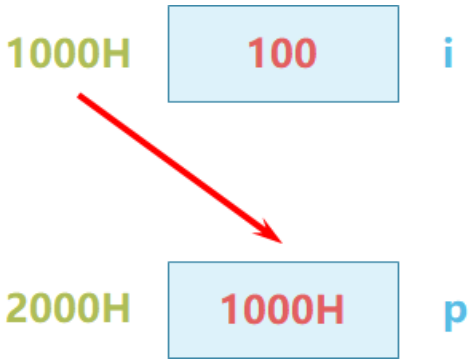


图3. 变量的间接访问

变量的指针

变量的指针就是变量的地址。当变量定义后，其指针（地址）就是一个确定的常量。

例如，

```
1. int i;
```

就在内存中为变量 i 产生了一个存储空间。比如这个存储空间的起始地址是 1000H。那么，无论将来变量 i 的取值是什么，这个地址都不会再发生变化。

我们可以利用取地址运算符“&”对变量运算，得到变量的地址。

例如，下面的代码就表示变量 i 的地址：

```
1. &i
```

指针变量的定义

可以定义一个变量专门用来存放另一变量的地址，这种变量我们称之为指针变量。

指针变量在编译时，同样也被分配一定字节的存储单元。未赋初值时，该存储单元内的值是随机的。

指针变量定义的一般形式为：

```
1. 类型标识符 *指针变量名;
```

例如，在下面的代码中 pa 是一个整型指针，pb 是一个双精度实型指针。

```
1. int *pa;
2. double *pb;
```

指针变量的初始化

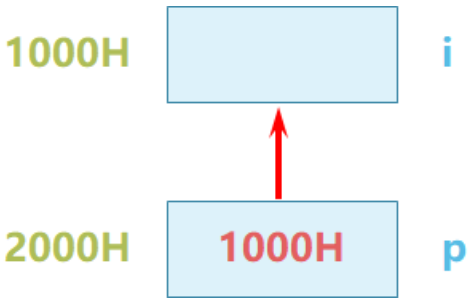
指针变量有以下三种初始化的方式。

第一，初始化为空指针。即 p 的内容初始化为 NULL，也就是 0。表示 p 没有指向任何内容。

```
1.  int *p = NULL;
```

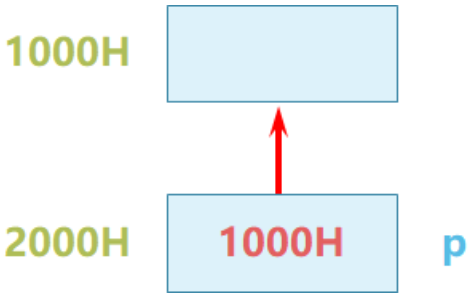
第二，初始化为某个变量的地址。

```
1.  int i;
2.  int *p = &i;
```



第三，申请一个新空间，空间内容不确定。

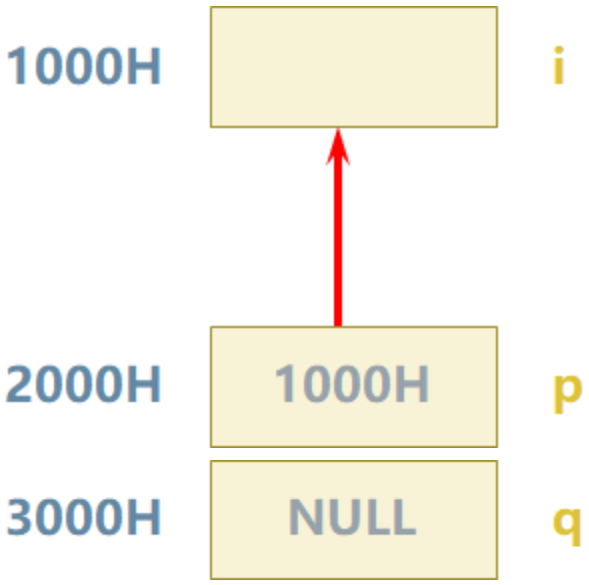
```
1.  int *p = new(int);
```



指针变量的赋值

指针变量可以被赋值：

```
1. int i;
2. int *p;
3. p = &i;
4. q = NULL;
```



也可以在定义指针变量时赋初值：

```
1. int i;
2. int *p = &i;
3. int *p = null;
```

指针变量只能存放地址，不要将非地址数据赋给指针变量。因此下面的代码是非法的：

```
1. int *p;
2. p = 100;
```

通过指针间接访问变量

如果 p 是一个指针变量，我们可以通过*p 的方式，间接访问 p 指向的数据内容。

我们要注意，一个指针变量只能指向同一类型的变量。即整型指针变量只能放整型数据的地址，而不能放其他类型数据的地址。

【参考程序】

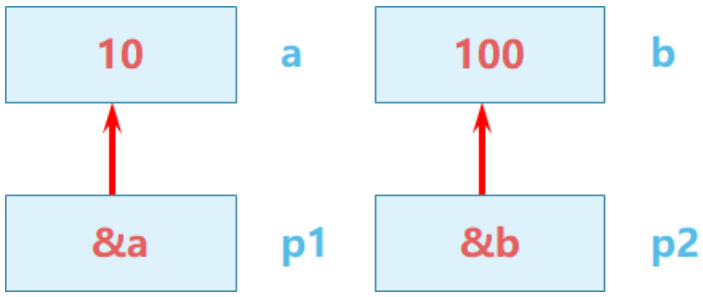
```
#include<iostream>
using namespace std;
int main()
{
    int a = 10;
    int b = 100;
    int *p1 = &a;
    int *p2 = &b;
    cout << a << '\t' << b << endl;
    cout << *p1 << '\t' << *p2 << endl;
    return 0;
}
```

【输出结果】

```
10      100
10      100
```

【分析】

指针 p1 指向的是 a，指针 p2 指向的是 b，因此*p1 访问的就是 a，*p2 访问的就是 b。



通过指针间接赋值变量

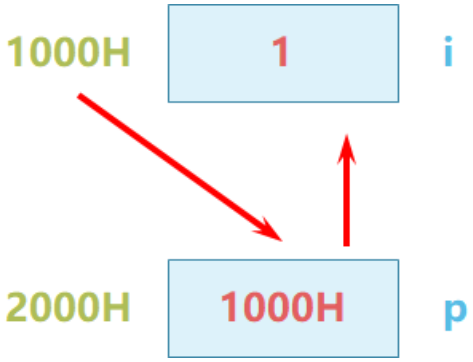
如果 p 是一个指针变量，我们可以通过下面的方式，间接给 p 指向的数据内容赋值。

```
*p = 表达式;
```

例如，

```
int i;
int *p = &i;
*p = 1;
```

执行过程，如下图所示：



“*”在定义语句中只表示变量的类型是指针，没有任何计算意义。

“*”在其他语句中表示“指向”。

“&”表示“地址”。

【参考程序】

```
#include<iostream>
using namespace std;
int main()
{
    int a, b;
    int *p1 = &a;
    int *p2 = &b;
    *p1 = 10;
    *p2 = 100;
    cout << a << '\t' << b << endl;
    cout << *p1 << '\t' << *p2 << endl;
```

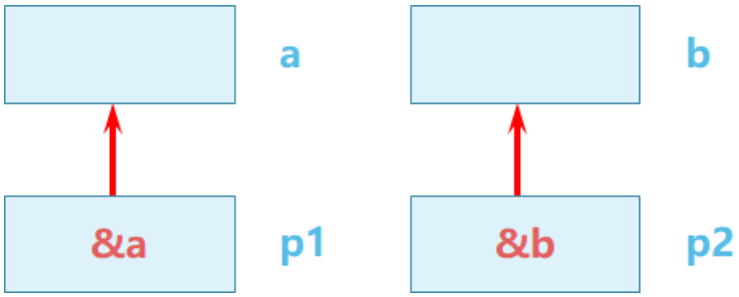
```
12.     return 0;
13. }
```

【输出结果】

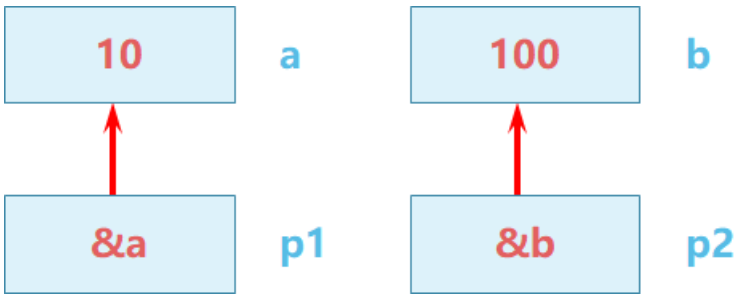
```
10      100
10      100
```

【分析】

赋值前:



赋值后:

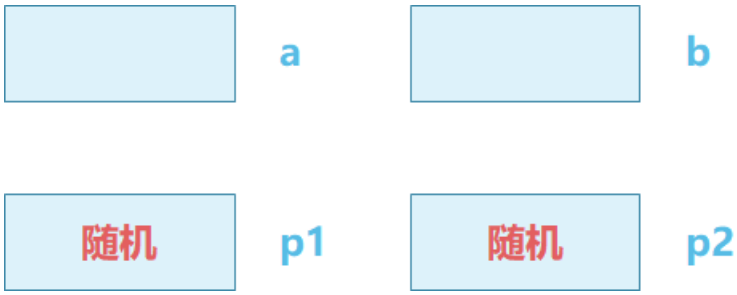


绝对不能使用未赋值的指针

下面程序的执行结果是什么?

```
#include<iostream>
using namespace std;
int main()
{
    int a, b;
    int *p1, *p2;
    *p1 = 10;
    *p2 = 100;
    cout << a << '\t' << b << endl;
    cout << *p1 << '\t' << *p2 << endl;
    return 0;
}
```

程序直接崩溃了!



这是因为指针变量未赋值，即指针指向未知地址。

向未知地址写数据，会造成系统的崩溃。

所以，我们在用指针变量前，必须对指针变量赋值。绝对不能使用未赋值的指针变量。

指针操作总结

根据以下代码片段，对指针操作进行总结。

```
1. int a = 10;
2. int *p;
3. p = &a;
4. *p = 20;
5. cout << p;
6. cout << *p;
```

第 2 行，是指针定义，指针定义的一般格式如下：

```
1. 类型标识符 *指针变量名;
```

第 3 行，“&”是地址运算符。

第 4 行，“*”是指针运算符。

第 5 行，指针变量直接存取的是内存地址，因此输出结果是类似 0x12345678 这样的数字。

第 6 行，间接存取的才是储存类型的值，因此输出是 20。

一般的，我们可以这样理解指针（int *p）和普通变量（int a）的对应关系：

p ---- &a

*p ---- a

*p=10 ---- a=10