可变长度数组 Vector

我们之前学习了普通的数组，但是普通数组最大的问题就是数组的长度必须事先定义好，对于很多我们不知道其确切长度的问题，我们则需要尽可能的建立足够的长度，随之带来的问题就是我们会浪费很多的空间。

可变长度数组由此应运而生，它可以使我们最大限度的建立合适的长度，避免空间浪费。

声明

同任何一个变量一样，我们在使用之前需要声明这个Vector数组，方法如下：

```
//声明一个没有初始化数据的可变长度数组
vector<int> a;
//声明一个长度为5的可变长度数组
vector<double> b(5);
//声明一个长度为5，并且内容被初始化为'a'
vector<char> c(5, 'a');
```

此外，我们还可以在程序的运行过程中去声明一个vector的长度，这样就更加方便了我们的操作，如下代码所示：

```
#include <bits/stdc++.h>

using namespace std;

int main() {
    int n;
    //读入数组长度
    cin >> n;
    //定义数组，并设置指定长度
    vector<long> vec(n);
    //输出数组的当前长度
    cout << vec.size();
    return 0;
}
```

访问

同普通的数组一样，我们也是通过索引对其进行访问，同样索引的大小不能超过当前的长度上限，起始索引依然为0。

注意：没有设置长度的可变数组，在没有通过push_back方法赋值前，将无法访问数据。详情参考下文的push方法。示例如下：

```
#include<bits/stdc++.h>

using namespace std;

int main() {
    //定义一个长度为5的可变长度数组
  vector<int> a(5);
    //和普通数组一样进行赋值
  a[0] = 10;
    a[1] = 100;
    //进行数组的访问
  cout<<a[0]<<endl;
    cout<<a[1]<<endl;
    return 0;
}
```

此外，我们还能通过begin和end函数获取vector的首末两个元素的迭代器。

```
#include<bits/stdc++.h>

using namespace std;

int main() {
    //定义一个长度为5的可变长度数组
   vector<int> a;
    //将数据插入到数组的末尾，并增长数组的长度
   a.push_back(10);
    a.push_back(100);

    vector<int>::iterator begin = a.begin();
    vector<int>::iterator end = a.end();

    //注意左闭右开
   cout << *begin << " " << *(--end);

    return 0;
}
```

push_back 向数组插入数据

这样看起来，我们的可变长度数组与普通数组似乎没有任何区别，那么它到底可变在哪里呢？

实际上，vector在初始化的时候，是不必须声明长度的。同时，我们将使用push_back函数对齐进行数据的写入，以达到长度可变的目的。如下所示：

```
#include<bits/stdc++.h>

using namespace std;

int main() {
    //定义一个长度为5的可变长度数组
  vector<int> a;
    //将数据插入到数组的末尾，并增长数组的长度
  a.push_back(10);
    a.push_back(100);
    //进行数组的访问
  cout<<a[0]<<endl;
    cout<<a[1]<<endl;
    return 0;
}
```

size 返回数组的长度

我们插入了很多的数据之后，想知道数组中到底有多少个元素时该怎么办呢？这个时候，我们就可以使用size函数了。示例如下：

```
#include<bits/stdc++.h>

using namespace std;

int main() {
    //定义一个长度为5的可变长度数组
  vector<int> a;
    //将数据插入到数组的末尾，并增长数组的长度
  a.push_back(10);
    a.push_back(100);
    //获取数组的长度
  int n = a.size();
    //进行数组的访问
  for (int i = 0; i < n; ++i) {
        cout << a[i] << endl;
    }

    return 0;
}
```

resize 重新调整数组大小

有些时候，我们需要根据题目来重新调整数组的大小，这个时候resize函数就发挥了作用。

例如：

resize(5)

这个命令将会把数组的长度调整为5，如果当前数组的实际长度小于5，则会自动增长到5。如果大于5，那么多余的部分将会被舍弃。

```
#include<bits/stdc++.h>
using namespace std;
int main() {
    //定义一个长度为5的可变长度数组
  vector<int> a;
    //将数据插入到数组的末尾，并增长数组的长度
  a.push_back(10);
    a.push_back(100);
    //输出当前的长度
  cout << a.size() << endl;
    //调整新的长度为5
    a.resize(5);
    //输出最新长度
  cout << a.size() << endl;
    return 0;
}
```

我们还可以利用这个函数在调整数组大小的时候，直接对多出来的部分进行赋值。

```
#include<bits/stdc++.h>
using namespace std;
int main() {
    //定义一个长度为5的可变长度数组
  vector<int> a;
    //将数据插入到数组的末尾，并增长数组的长度
  a.push_back(10);
    a.push_back(100);
    //输出当前的长度
  cout << a.size() << endl;
    //调整新的长度为5
    a.resize(5, 100);
    //输出内容
  for (int i = 0; i < 5; ++i) {
        cout << a[i] << endl;
    }
    return 0;
}
```

erase 删除数据

```
int main() {
    //定义一个长度为5的可变长度数组
    vector<int> a;
    //将数据插入到数组的末尾，并增长数组的长度
    a.push_back(10);
    a.push_back(100);
    //输出当前的长度
    cout << a.size() << endl;
    //删除第一个元素
    a.erase(a.begin());
    //输出修改后的长度
    cout << a.size() << endl;
    return 0;
}
```

二维vector

同普通的多维数组一样，vector也支持多维，下面是两个多维vector的示例，需要注意方法1最后的两个>>箭头，千万不要连在一起，否则就会被识别为位移运算。

```
#include <bits/stdc++.h>

using namespace std;

int main() {
    //方法1：
    //定义二维vec
    vector<vector<long> > vec1(5);
    vec1[0].push_back(1000);
    cout << vec1[0][0] << endl;

    //方法2：
    //当前vec2是由10个可变数组组成
    //即可变数组的数组
    vector<int> vec2[10];
    vec2[0].push_back(100);
    cout << vec2[0][0] << endl;

    return 0;
}
```

数组的特点和局限性

存储查询给定索引（下标）的数据：效率很高，复杂度 O(1)

将整个数组的一段数据进行插入或删除操作，或者搜索指定元素（如果没有排序）：效率很低，时间复杂度 O(n)。