

## 分割實作

### 壹、分割

```
1 #include <iostream>
2 #include <cstdlib>
3 #include <ctime>
4
5 template <class Type> inline void exchange(Type* a, Type* b)
6 { Type c = *a; *a = *b, *b = c; }
7
8 /**
9  * @brief array of pointer to use
10  * @tparam Type array contain type
11  * @tparam Size array length type
12  */
13 template <class Type, class Size> struct Array
14 {
15     Type* start; Size length;
16     Array(Size _length)
17     {
18         this->start = new Type[_length];
19         this->length = _length;
20     }
21     Array(Type* _start, Size _length)
22     {
23         this->start = _start;
24         this->length = _length;
25     }
26 };
27
28 /**
29  * @brief Set random element to a array (Does not include setting random seed) (`std::srand(std::time(nullptr));`)
30  * @tparam Type array contain type
31  * @tparam Size array length type
32  * @param array modified target
33  */
34 template <class Type, class Size> inline void setRandomElement_Array(Array<Type, Size>& array)
35 { for (size_t i=0; i<array.length; i++) *(array.start+i) = (std::rand()^std::rand())%100; }
36
37 template <class Type, class Size> std::ostream& operator<< (std::ostream& outStream, Array<Type, Size>& array)
38 {
39     outStream<< "[";
40     for (size_t i=0; i<array.length; i++)
41     {
42         if (i) outStream<< ", ";
43         outStream<< *(array.start+i);
44     }
45     outStream<< "]";
46     return outStream;
47 }
```

## 一、分割

```

45 /**
46  * @brief Cut Partition. Use the specified index value as the split pivot,
47  * move the position of the element so that the position greater than the pivot value is behind the pivot value
48  * @tparam Type array contain type
49  * @tparam Size array length type
50  * @param array modified target
51  * @param pivot initial pivot position
52  * @return end pivot position
53  */
54 template <class Type, class Size> Size singlePivotSplit_Array(Array<Type, Size>& array, Size pivot=0)
55 { // std::cout<<"singlePivotSplit_Array: "<<array<<std::endl;
56   // Normalized pivot position
57   for (; pivot!=0; pivot--) exchange((array.start+pivot), (array.start+pivot-1));
58   /**
59   * @brief main action: Continue to move all values greater than the pivot value to the back,
60   * and finally move the pivot to the front of the first value greater than the pivot value
61   * @param i The position retainer of pivot,
62   * and elements greater than this position are greater than pivot value
63   * @param j Forward explorer,
64   * will exchange value with The position retainer of pivot when explored element is greater than pivot value
65   */
66   for (size_t i, j=array.length-1; j<=i && i*j!=0; )
67   {
68     // The explorer moves The position retainer of pivot forward
69     for (; *(array.start+j)>*(array.start+pivot) && i==j && j>0; i--j) ;
70     // The explorer moves to a position greater than the pivot value
71     for (; *(array.start+j)<=(array.start+pivot) && j>0; j--) ;
72     // Exchange the value of the explored position to The position retainer of pivot
73     if (*(array.start+j)>*(array.start+pivot) && j>0)
74       exchange((array.start+j), (array.start+i)), i--, j--;
75     // Explore to the end
76     if (j==0)
77       exchange((array.start+pivot), (array.start+i)), pivot = i;
78   }
79   return pivot;
80 }

```

```

81 /**
82  * @brief Cut Partition. Divide all adjacent 'groupNumber' elements into one group,
83  * after each group obtains the value of the specified order, take out all the values again,
84  * take out the specified order as the pivot value, and finally use this value for single pivot division.
85  * @tparam Type array contain type
86  * @tparam Size array length type
87  * @param array modified target
88  * @param groupNumber divide all adjacent 'groupNumber' elements into one group. Automatically initialized to 5.
89  * @param groupOrder order of pivot values in each group, Automatically initialized to "SIZE_MAX",
90  * if it is greater than groupNumber during execution, it will be automatically regarded as groupNumber/2.
91  * @param order initial pivot position, Automatically initialized to "SIZE_MAX",
92  * if it is greater than groupNumber during execution, it will be automatically regarded as groupNumber/2.
93  * @return end pivot position
94  */
95 template <class Type, class Size> Size multiPivotSplit_Array(Array<Type, Size>& array, Size groupNumber=5, Size groupOrder=SIZE_MAX, Size order=SIZE_MAX)
96 { // std::cout<<"multiPivotSplit_Array: "<<array<<std::endl;
97   // parameter correction
98   if (groupNumber==0 || groupNumber>array.length) groupNumber = 5;
99   if (groupOrder>groupNumber) groupOrder = groupNumber>>1;
100   Size g_full=array.length/groupNumber, g_remains=array.length%groupNumber;
101   if (order>groupNumber) order = g_full>>1;
102   // group information
103   Array<Type, Size> pivotArray(g_full+(g_remains?1:0));
104   // qsort each group of data to get pivot of group
105   for (Size g=0; g<pivotArray.length; g++)
106   {
107     Size g_length = ((g!=g_full)?groupNumber:g_remains);
108     std::qsort((array.start+(groupNumber*g)), g_length, sizeof(Type), [](const void* a, const void* b)->
109       int{ if ( *(Type*)a < *(Type*)b ) return -1; if ( *(Type*)a > *(Type*)b ) return 1; return 0; });
110     *(pivotArray.start+g) = *(array.start+(groupNumber*g)+(groupOrder<g_length?groupOrder:(g_length-1)));
111   }
112   // qsort pivot of group to get pivot of all element
113   std::qsort(pivotArray.start, pivotArray.length, sizeof(Type), [](const void* a, const void* b)->
114     int{ if ( *(Type*)a < *(Type*)b ) return -1; if ( *(Type*)a > *(Type*)b ) return 1; return 0; });
115   Type pivot = *(pivotArray.start+order);
116   // search pivot index
117   Size pivotIndex = groupOrder;
118   for (; pivotIndex<array.length; pivotIndex+=groupNumber) if (*(array.start+pivotIndex) == pivot) break;
119   if (pivotIndex==array.length) pivotIndex=pivotIndex-groupOrder+g_remains-1;
120   // Use the found pivot index value to perform a single pivot split, and return the split pivot index value
121   return singlePivotSplit_Array(array, pivotIndex);
122 }

```



```
124 int main()
125 {
126     std::srand(std::time(nullptr));
127     size_t length; std::cout<<"data length: "; std::cin>>length;
128     std::cout<<std::endl;
129
130     struct Array<size_t, size_t> Data(length);
131     size_t pivot;
132
133     setRandomElement_Array(Data);
134     std::cout<<Data<<std::endl;
135     std::cout<<"pivot index: "; std::cin>>pivot;
136     std::cout<<"pivot value: "<<*(Data.start+pivot)<<std::endl;
137     std::cout<<std::endl; pivot = singlePivotSplit_Array(Data, pivot);
138     std::cout<<Data<<std::endl;
139     std::cout<<"pivot index: "<<pivot<<std::endl;
140     std::cout<<"pivot value: "<<*(Data.start+pivot)<<std::endl;
141
142     std::cout<<std::endl<<std::endl;
143
144     setRandomElement_Array(Data);
145     std::cout<<Data<<std::endl;
146     size_t groupMumber; std::cout<<"groupMumber: "; std::cin>>groupMumber;
147     std::cout<<std::endl; pivot = multiPivotSplit_Array(Data, groupMumber);
148     std::cout<<Data<<std::endl;
149     std::cout<<"pivot index: "<<pivot<<std::endl;
150     std::cout<<"pivot value: "<<*(Data.start+pivot)<<std::endl;
151 }
```

## 二、快速排序

```
123  /**
124   * @brief quick sort a Array<Type, Size>
125   * @tparam Type array contain type
126   * @tparam Size array length type
127   * @param array modified target
128   */
129  template <class Type, class Size> void quickSort_Array(Array<Type, Size>& array)
130  {
131      // Get segmentation
132      Size pivotIndex = multiPivotSplit_Array(array);
133      // If less than or equal to the pivot value the part length is greater than 1
134      if (pivotIndex>1)
135      {
136          Array<Type, Size> front(array.start, pivotIndex);
137          quickSort_Array(front);
138      }
139      // If the length of the part greater than the pivot value is greater than 1
140      if (array.length-pivotIndex-1>1)
141      {
142          Array<Type, Size> rear(array.start+pivotIndex+1, array.length-pivotIndex-1);
143          quickSort_Array(rear);
144      }
145  }
```

```
101  int main()
102  {
103      std::srand(std::time(nullptr));
104      size_t length; std::cout<<"data length: "; std::cin>>length;
105      std::cout<<std::endl;
106
107      struct Array<size_t, size_t> Data(length);
108      setRandomElement_Array(Data);
109
110      std::cout<<Data<<std::endl;
111
112      std::cout<<std::endl; quickSort_Array(Data);
113
114      std::cout<<Data<<std::endl;
115  }
```

### 三、找指定大小順序值(從 0 開始)

```

123  /**
124   * @brief find order from a Array<Type, Size>
125   * @tparam Type array contain type
126   * @tparam Size array length type
127   * @param array find target
128   * @param order order of array
129   * @return element values in the specified order
130   */
131  template <class Type, class Size> Type findOrder_Array(Array<Type, Size>& array, Size order)
132  {
133      // Create a copy
134      Array<Type, Size> _array(array.length);
135      for (Size i=0; i<_array.length; i++) *(_array.start+i) = *(array.start+i);
136      // Get segmentation
137      Size pivotIndex = multiPivotSplit_Array(_array);
138      // If the target is before the split point
139      if (pivotIndex>order)
140      {
141          Array<Type, Size> front(_array.start, pivotIndex);
142          return findOrder_Array(front, order);
143      }
144      // If the target is after the split point
145      else if (pivotIndex<order)
146      {
147          Array<Type, Size> rear(_array.start+pivotIndex+1, _array.length-pivotIndex-1);
148          return findOrder_Array(rear, order-pivotIndex-1);
149      }
150      // If the split point is the target
151      return *(_array.start+pivotIndex);
152  }

```

```

108  int main()
109  {
110      std::srand(std::time(nullptr));
111      size_t length; std::cout<<"data length: "; std::cin>>length;
112      std::cout<<std::endl;
113
114      struct Array<size_t, size_t> Data(length);
115      setRandomElement_Array(Data);
116
117      std::cout<<Data<<std::endl;
118
119      size_t order; std::cout<<"order: "; std::cin>>order;
120      std::cout<<"element: "<<findOrder_Array(Data, order)<<std::endl;
121
122      std::cout<<Data<<std::endl;
123  }

```

#### 四、找中間值(從 0 開始，若於兩元素之間，回傳最小的元素)

```

153  /**
154   * @brief find middle element from a Array<Type, Size>
155   * @tparam Type array contain type
156   * @tparam Size array length type
157   * @param array find target
158   * @return middle element values from array
159   */
160  template <class Type, class Size> Type findMiddle_Array(Array<Type, Size>& array)
161  { // return findOrder_Array(array, array.length>>1);
162    // Create a copy
163    Array<Type, Size> _array(array.length);
164    for (Size i=0; i<_array.length; i++) *(_array.start+i) = *(array.start+i);
165    Size order = _array.length>>1;
166    // Get segmentation
167    Size pivotIndex = multiPivotSplit_Array(_array);
168    // If the target is before the split point
169    if (pivotIndex>order)
170    {
171        Array<Type, Size> front(_array.start, pivotIndex);
172        return findOrder_Array(front, order);
173    }
174    // If the target is after the split point
175    else if (pivotIndex<order)
176    {
177        Array<Type, Size> rear(_array.start+pivotIndex+1, _array.length-pivotIndex-1);
178        return findOrder_Array(rear, order-pivotIndex-1);
179    }
180    // If the split point is the target
181    return *(_array.start+pivotIndex);
182  }

```

```

138  int main()
139  {
140      std::srand(std::time(nullptr));
141      size_t length; std::cout<<"data length: "; std::cin>>length;
142      std::cout<<std::endl;
143
144      struct Array<size_t, size_t> Data(length);
145      setRandomElement_Array(Data);
146
147      std::cout<<Data<<std::endl;
148
149      std::cout<<"middle element: "<<findMiddle_Array(Data)<<std::endl;
150
151      std::cout<<Data<<std::endl;
152  }

```