<https://www.cnblogs.com/chengliangsheng/p/3596943.html>

C语言也能面向对象？不是C++是面向对象的么？其实C语言也能抽象成简单的面向对象方法，在Linux内核源码当中，底层的驱动代码、文件系统等皆采用了面向对象的封装技术，这样的好处是将客观的东西抽象出来，以接口的方式管理。

　　C++完全包容C语言的语法特点，C++中类：class和C语言中的结构体:struct是等效的，不过C++是一种完全面向对象的模式，其中域、对象名，都封装在类里面，而C语言没有明确规定，只是结构体是一种根据设计需要来构造的一种特殊的数据类型。C++中每个类都提供一个默认的构造函数和析构函数（当然也可以自定义一个构造函数）。下面是用纯C语言实现一个C++的vector容器：

[复制代码](javascript:void(0);)

1 #include <stdio.h>

2 #include <stdlib.h>

3 #include <assert.h>

4 #include <string.h>

5 typedef int DataType;

6 typedef struct array

7 {

8 DataType \*Data;

9 int size,max\_size;

10 　 void (\*Constructor)(struct array \*);//构造函数

11 void (\*Input)(DataType ,struct array \*);//输入数据

12 　 int (\*get\_array\_size)(struct array \*);//获取arr的大小

13 　 int (\*return\_index\_value)(struct array \*,int);//返回下标为index的值

14 void (\*print)(struct array \*);//打印结果

15 void (\*Destructor)(struct array \*);//析构函数

16 }Array;

17

18 void Init(Array \*this);

19 void \_print(struct array \*this);

20 void \_constructor(Array \*this);

21 void \_denstructor(Array \*this);

22 void \_input(DataType data,Array \*this);

23 int \_get\_szie(Array \*this);

24 int \_return\_index\_value(Arrary \*this,int index);

25

26 void Init(Array \*this)

27 {

28 this->Input =\_input;

29 this->print =\_print;

30 　 this->get\_array\_size = \_get\_size;

31 　 this->return\_index\_value = \_return\_index\_value;

32 this->Constructor =\_constructor;

33 this->Destructor =\_denstructor;

34 this->Constructor(this);

35 }

36

37 void \_constructor(Array \*this)

38 {

39 this->size=0;

40 this->max\_size=10;

41 this->Data=(DataType \*)malloc(this->max\_size\*sizeof(DataType));

42 memset(this->Data,0,10);

43 }

44

45 void \_input(DataType data, Array \*this)

46 {

47 int i;

48 DataType \*ptr;

49

50 if(this->size >= this->max\_size)

51 {

52 this->max\_size +=10;

53 ptr=(DataType \*)malloc(this->max\_size\*sizeof(DataType));

54 for(i=0;i<this->size;i++)

55 ptr[i]=this->Data[i];

56 free(this->Data);

57 this->Data=ptr;

58 }

59 this->Data[this->size]=data;

60 this->size +=1 ;

61 }

62

63 void \_print(struct array \*this)

64 {

65 assert(this != NULL);

66 struct array \*ptr=this;

67 int i=0;

68 for(i=0;i<ptr->size;i++)

69 printf("data is %d\n",ptr->Data[i]);

70

71 return ;

72 }

73 int \_get\_array\_size(Array \*this)

74 {

75 　　assert(this != NULL);

76 　　return this->size+1;

77 }

78 int \_return\_index\_value(Array \*this,int index)

79 {

80 　　assert(this != NULL);

81 　　return (this->Data[index]);

82 }

83 void \_denstructor(Array \*this)

84 {

85 int i=0;

86 　 assert(this != NULL);

87 for(i=0;i<this->max\_size;i++)

88 this->Data[i]=0;

89 free(this->Data);

90 }

91

92 int main()

93 {

94 Array MyArray;

95

96 Init(&MyArray); //使用对象前必须初始化，间接调用构造函数

97 // MyArray.Data[]={1,2,3,4,5};

98 MyArray.Input(1,&MyArray);

99 MyArray.Input(2,&MyArray);

100 MyArray.Input(3,&MyArray);

101 MyArray.Input(4,&MyArray);

102 MyArray.Input('5',&MyArray);

103 MyArray.print(&MyArray);

104 　 printf("the array size is :%d\n",MyArray.get\_array\_size(&MyAarray));

105 　 printf("the index value in array is:%d\n",MyArray.return\_index\_value(&MyArray,3));

106 MyArray.Destructor(&MyArray); //使用对象后必须显式调用析构函数

107

108 return 0;

109 }