1: typedef int ElemType;

2:

3: #define LIST\_INIT\_SIZE 100

4: #define LIST\_INCREMENT 10

5:

6: typedef struct {

7: ElemType \*elem;

8: int length;

9: int listsize;

10: }SqList;

11:

12: int InitList(SqList \*L)

13: {

14: L->elem = (ElemType \*)malloc(sizeof(ElemType) \* LIST\_INIT\_SIZE);

15: if (!L->elem)

16: return -1;

17: L->length = 0;

18: L->listsize = LIST\_INIT\_SIZE;

19:

20: return 0;

21: }

22:

23: void DestroyList(SqList \*L)

24: {

25: free(L->elem);

26: L->length = 0;

27: L->listsize = 0;

28: }

29:

30: void ClearList(SqList \*L)

31: {

32: L->length = 0;

33: }

34:

35: bool ListEmpty(SqList \*L)

36: {

37: return (L->length == 0);

38: }

39:

40: int ListLength(SqList \*L)

41: {

42: return L->length;

43: }

44:

45: int GetElem(SqList \*L, int i, ElemType \*e)

46: {

47: if (i < 0 || i >= L->length)

48: return -1;

49:

50: \*e = L->elem[i];

51:

52: return 0;

53: }

54:

55: int LocateElem(SqList \*L, ElemType e, bool (\*compare)(ElemType a, ElemType b))

56: {

57: int i;

58:

59: for (i = 0; i < L->length; ++i) {

60: if (compare(L->elem[i], e))

61: return i;

62: }

63:

64: return -1;

65: }

66:

67: int PriorElem(SqList \*L, ElemType e, ElemType \*pre\_e)

68: {

69: int i;

70:

71: for (i = 1; i < L->length; ++i) {

72: if (e == L->elem[i]) {

73: \*pre\_e = L->elem[i-1];

74: return 0;

75: }

76: }

77:

78: return -1;

79: }

80:

81: int NextElem(SqList \*L, ElemType e, ElemType \*next\_e)

82: {

83: int i;

84:

85: for (i = 0; i < L->length - 1; ++i) {

86: if (e == L->elem[i]) {

87: \*next\_e = L->elem[i+1];

88: return 0;

89: }

90: }

91:

92: return -1;

93: }

94:

95: int ListInsert(SqList \*L, int i, ElemType e)

96: {

97: if (i < 0 || i > L->length) {

98: return -1;

99: }

100:

101: if (L->length >= L->listsize) {

102: ElemType \*newbase = (ElemType \*)realloc(L->elem, sizeof(ElemType) \* (L->listsize + LIST\_INCREMENT));

103: if (!newbase) {

104: return -1;

105: }

106: L->elem = newbase;

107: L->listsize += LIST\_INCREMENT;

108: }

109:

110: ElemType \*p, \*q;

111: q = L->elem + i;

112: for (p = L->elem + L->length - 1; p >= q; --p)

113: \*(p+1) = \*p;

114:

115: \*q = e;

116: L->length += 1;

117:

118: return 0;

119: }

120:

121: int ListDelete(SqList \*L, int i, ElemType \*e)

122: {

123: if (i < 0 || i >= L->length) {

124: return -1;

125: }

126:

127: \*e = L->elem[i];

128:

129: ElemType \*p;

130: for (p = L->elem + i; p < L->elem + L->length - 1; ++p) {

131: \*p = \*(p+1);

132: }

133: L->length -= 1;

134:

135: return 0;

136: }

137:

138: int ListTraverse(SqList \*L, int (\*visit)(ElemType e))

139: {

140: int i, ret;

141:

142: for (i = 0; i < L->length; ++i) {

143: ret = visit(L->elem[i]);

144: if (ret != 0)

145: return ret;

146: }

147:

148: return 0;

149: }