中北大学软件学院

**实验报告**

|  |  |
| --- | --- |
| 专 业： | 软件工程 |
| 课程名称： | 数据结构与算法 |
| 班 级： | 21130420 |
| 学 号： | 2113042015 |
| 姓 名： | 程楚晋 |
| 辅导教师： | 马巧梅 |

2022年3月制

成绩：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 实验时间 | 2022年3 月11日 8 时至 10时 | 学时数 | 2学时 | |
| 1.实验名称  抽象数据类型的表示和实现 | | | | |
| 2.实验目的  （1）复习巩固C语言的结构体、指针、函数、动态分配内存等知识点。  （2）掌握抽象数据类型的定义、表示和实现，为后续实验打好基础。 | | | | |
| 3.实验内容  **基本要求：**  设计实现抽象数据类型“三元组”，要求动态分配内存。每个三元组由任意三个实数的序列构成，基本操作包括：创建一个三元组，取三元组的任意一个分量，置三元组的任意一个分量，求三元组的最大分量，求三元组的最小分量，显示三元组，销毁三元组等。  **选作内容：**  实现两个三元组的对应分量相加或相减，给三元组的各分量同乘一个比例因子等操作。 | | | | |
| 4.数据类型描述（此处写清楚三元组的抽象数据类型描述）  ADT(三元组名称)Triplet  Triplet包括：  数据对象: T={e1,e2,e3|属于ElemType};  数据关系：R={<e1,e2>,<e2,e3>}(线性关系)；  **基本操作：**  initTriplet(&T,v1,v2,v3) （初始化三元组）  //操作结果：创建三元组T，将e1,e2,e3分别赋予v1,v2,v3的值。  getElem(T,i,&e) （取三元组的任意元）  //初始条件：三元组T已经存在，1<=i<=3  操作结果：用e返回三元组T的第i个元素；  putElem(&T,i,e) （设置三元组的任意元）  //初始条件：三元组T已经存在，1<=i<=3  操作结果：用e值取代三元组的第i个元素；  getMax(T,&e) （求三元组分量最大值）  //初始条件：三元组T已经存在  操作结果：用e返回T中的3个元素中的最大值；  getMin(T,&e) （求三元组分量最小值）  //初始条件：三元组T已经存在  操作结果：用e返回T中的3个元素中的最大值；  printTriplet(T) （显示三元组）  //初始条件：三元组T已经存在  操作结果：呈现三元组；  DestoryTriplet(&T) （销毁三元组）  //初始条件：三元组T已经存在  操作结果：销毁三元组T；  选做内容：  Add(T1,&T2) (三元组的对应分量相加或相减）  //初始条件：T1,T2均存在  mulCoef(T1,coef) (给三元组的各分量同乘一个比例因子)  //初始条件：T1,T2均存在 | | | | |
| 5.实验源代码  方法一：   1. #include<stdio.h> 2. #include<stdlib.h> 3. #include "stdbool.h" 4. #define OK 1 5. #define  ERROR 0 6. #define N 3  //开辟N四个字符空间,实现N元组 7. **typedef** **int** Status; 8. **typedef** **float** ElemType; 9. **typedef** ElemType \*Triplet;  //给ElemType类型的指针类型起别名Triplet 11. //N元组的初始化 12. Status initTriplet(Triplet &T,ElemType v[]) 13. { 14. //动态内存分配 15. T=(ElemType \*)malloc(**sizeof** (ElemType) \* N); 16. **if**(!T) exit(-2); 17. **int** i; 18. **for**(i=0;i<N;i++) 19. { 20. T[i]=v[i]; 21. } 22. } 23. //输出N元组 24. Status printTriple(Triplet T) 25. { 26. **int** i; 27. **for**(i=0;i<N;i++) 28. { 29. printf("%.1f\t",T[i]); 30. } 31. printf("\n"); 32. } 33. Status getELem(Triplet T,**int** i,ElemType &e) 34. { 35. **if** (i < 1 || i > N)   //先判断是否超出了给定的空间 36. **return** ERROR; 37. **else**  e=T[i-1]; 38. **return**  OK; 39. } 40. Status putElem(Triplet &T,**int** i,ElemType e) 41. { 42. **if** (i < 1 || i > 3) 43. **return** ERROR; 44. **else**    T[i-1]=e; 45. **return** OK; 46. } 47. Status getMax(Triplet T,ElemType &max) 48. { 49. **int** i; 50. max=T[0]; 51. **for**(i=1;i<N;i++) 52. **if**(T[i]>max) max=T[i]; 53. **return** max; 54. } 55. //getMin 于此同理 56. Status getMin(Triplet &T,ElemType &min) 57. { 58. **int** i; 59. min=T[0]; 60. **for**(i=1;i<=N;i++) 61. **if**(T[i]<min) min=T[i]; 62. **return** min; 63. } 64. **void** destroyTriplet(Triplet &T) 65. { 66. free(T); // 第三空 67. printf("分配内存已释放！"); 68. exit(0); 69. } 70. Status add(Triplet T1,Triplet T2) 71. { 72. **int** i; 73. **for**(i=0;i<N;i++) 74. T2[i]+=T1[i]; 75. } 76. Status mulCoef(Triplet &T1,**float** coef) 77. { 78. **for**(**int** i=0;i<N;i++) 79. T1[i]\*=coef; 80. } 81. Status isAscending(Triplet T) 82. { 83. **return** (T[0]<=T[1])&&(T[1]<=T[2]); 84. } 85. Status isDscending(Triplet T) 86. { 87. **return** (T[0]>=T[1])&&(T[1]>=T[2]); 88. } 90. **int** main() 91. { 92. Triplet T1, T2; 93. **int** funNum, i; 94. ElemType v[N], e, flag, coef; 95. printf("------N元组操作------\n"); 96. printf("1.N元组初始化\n"); 97. printf("2.输出三元组"); 98. printf("3.用e获取T的第i个元素的值\n"); 99. printf("4.置T的第i元的值为e\n"); 100. printf("5.升序排列\n"); 101. printf("6.降序排列\n"); 102. printf("7.取最大值\n"); 103. printf("8.取最小值\n"); 104. printf("9.输入另一个三元组相加\n"); 105. printf("10.N元组元素倍增或倍减\n"); 106. printf("退出\n"); 107. **int** m=10 ; 108. **while**(m--)  //循环检测各项功能 109. { 110. printf("输入功能序号\n"); 111. scanf("%d", &funNum); 112. **switch** (funNum) { 113. **case** 1: 114. printf("请输入N元组的每个元素(N=%d)\n", N); 115. **for** (i = 0; i < N; i++) 116. scanf("%f", &v[i]); 117. initTriplet(T1, v); 118. **break**; 119. **case** 2: 120. printf("初始化好的N元组为：\n"); 121. printTriple(T1); 122. **break**; 123. **case** 3: 124. printf("请输入要获取的第几个元素的值：\n"); 125. scanf("%d", &i); 126. getELem(T1, i, e); 127. printf("第%d个元素的值为%.1f\n",i,e); 128. **break**; 129. **case** 4: 130. printf("请输入要改变的位置和更改的值：\n"); 131. scanf("%d%f", &i, &e); 132. putElem(T1, i, e); 133. printf("新的N元组为：\n"); 134. printTriple(T1); 135. **break**; 136. **case** 5: 137. flag = isAscending(T1); 138. **if** (flag == 1) printf("升序排序\n"); 139. **else** printf("非升序排列\n"); 140. **break**; 141. **case** 6: 142. flag = isDscending(T1); 143. **if** (flag == 1) printf("降序排列\n"); 144. **else** printf("非降序排列\n"); 145. **break**; 146. **case** 7: 147. getMax(T1, e); 148. printf("最大值为%.1f\n", e); 149. **break**; 150. **case** 8: 151. getMin(T1, e); 152. printf("最小值为%.1f\n", e); 153. **break**; 154. **case** 9: 155. printf("请输入另一个N元组的每个元素（N=%d)\n", N); 156. **for**(i=0;i<N;i++) 157. scanf("%f",&v[i]); 158. initTriplet(T2,v); 159. add(T1,T2); 160. printf("相加之后的N(N=%d)元组为：\n",N) ; 161. printTriple(T2); 162. **break**; 163. **case** 10: 164. printf("请输入所乘系数\n"); 165. scanf("%f", &coef); 166. mulCoef(T1, coef); 167. printf("没项乘系数后的N元组为;\n"); 168. printTriple(T1); 169. **break**; 170. **case** 11: 171. destroyTriplet(T1); 172. destroyTriplet(T2); 173. **break**; 174. **default**: 175. printf("没有你需要的功能\n"); 176. **break**; 177. } 178. } 179. **return** 0; 180. }   方法二：   1. #include<stdio.h> 2. #include<stdlib.h> 3. #define OK 1 4. #define  ERROR 0 5. #define LIST\_SIZE 3  //开辟LIST\_SIZE四个字符空间 6. **typedef** **int** Status; 7. **typedef** **float** ElemType; 8. **typedef** **struct** 9. { 10. ElemType  e[3]; 11. }Triplet; 13. Status initTriplet(Triplet &T,ElemType v0,ElemType v1,ElemType v2) 14. { 15. T.e[0]=v0; 16. T.e[1]=v1; 17. T.e[2]=v2; 18. } 19. Status getELem(Triplet T,**int** i,ElemType &e) 20. { 21. **if** (i < 1 || i > LIST\_SIZE)   //先判断是否超出了给定的空间 22. **return** ERROR; 23. **else** e=T.e[i-1]; 24. **return**  OK; 25. } 27. Status putElem(Triplet &T,**int** i,ElemType z) 28. { 29. **if** (i < 1 || i > 3) 30. **return** ERROR; 31. **else** T.e[i-1]=z; 32. **return** OK; 33. } 35. Status getMax(Triplet T,ElemType &max) 36. { 37. **int** i; 38. max=T.e[0]; 39. **for**(i=1;i<LIST\_SIZE;i++) 40. **if**(T.e[i]>max) max=T.e[i]; 41. **return** max; 42. } 44. Status getMin(Triplet T,ElemType &min) 45. { 46. **int** i; 47. min=T.e[0]; 48. **for**(i=1;i<LIST\_SIZE;i++) 49. **if**(T.e[i]<min) min=T.e[i]; 50. **return** min; 51. } 53. Status isAscending(Triplet T) 54. { 55. **return** (T.e[0]<=T.e[1])&&(T.e[1]<=T.e[2]); 56. } 57. Status isDscending(Triplet T) 58. { 59. **return** (T.e[0]>=T.e[1])&&(T.e[1]>=T.e[2]); 60. } 61. **void** add(Triplet T,Triplet &T2) 62. { 63. **int** i; 64. **for**(i=0;i<LIST\_SIZE;i++) 65. T2.e[i]+=T.e[i]; 66. } 67. **void** mulCoef(Triplet &T,**float** coef) 68. { 69. **int** i; 70. **for**(i=0;i<LIST\_SIZE;i++) 71. T.e[i]\*=coef; 72. } 74. **int** main() 75. { 76. Triplet \*T,T2;     //开辟动态存储单位 77. T=(Triplet \*)malloc(LIST\_SIZE\***sizeof**(Triplet)); 78. ElemType a,b,c,v[LIST\_SIZE],e; 79. ElemType x,y,flag,coef; 80. **int** i; 81. //输入一个三元组 82. scanf("%f %f %f",&a,&b,&c); 83. initTriplet(\*T,a,b,c); 84. printf("三元组是：%f,%f,%f\n",T->e[0],T->e[1],T->e[2]); 85. //输入另一个三元组，并与原来的三元组相加 86. **for**(i=0;i<LIST\_SIZE;i++) 87. scanf("%f",&v[i]); 88. initTriplet(T2,v[0],v[1],v[2]); 89. add(\*T,T2);  //add函数将相加后的值返回到T2中 90. printf("相加之后的N(N=%d)元组为：\n",LIST\_SIZE) ; 91. printf("相加后的三元组是：%f,%f,%f\n",T2.e[0],T2.e[1],T2.e[2]); 92. //输入另一个三元组，并与原来的三元组相乘法 93. printf("请输入所乘系数\n"); 94. scanf("%f", &coef); 95. mulCoef(\*T, coef); 96. printf("每项乘系数后的N元组为;\n"); 97. printf("相乘后的三元组是：%f,%f,%f\n",T->e[0],T->e[1],T->e[2]); 98. //用e获取T的第2个元素的值 99. getELem(\*T, 2, e); 100. printf("第2个元素的值为%.1f\n",e); 101. //置T的第2元的值为 59 102. putElem(\*T, 2, 59); 103. printf("新的三元组是：%f,%f,%f\n",T->e[0],T->e[1],T->e[2]); 104. //判断是否升序 105. flag = isAscending(\*T); 106. **if** (flag == 1) printf("升序排序\n"); 107. **else** printf("非升序排列\n"); 108. //取最小值 109. getMax(\*T, e); 110. printf("最大值为%.1f\n", e); 111. //取最大值 112. getMin(\*T, e); 113. printf("最小值为%.1f\n", e); 114. //释放三元组T 115. free(T); 116. **return**  0; 117. } | | | |
| 6.实验结论及心得  开辟动态存储空间有两种方式，一种为在主函数开辟；   1. **int** main() 2. { 3. Triplet \*T; 4. T=(Triplet \*)malloc(N\***sizeof**(Triplet)); 5. }   另一种为在函数内部开辟：   1. Status initTriplet(Triplet &T,ElemType v[]) 2. { 3. //动态内存分配 4. T=(ElemType \*)malloc(**sizeof** (ElemType) \* N); 5. **if**(!T) exit(-2); 6. **int** i; 7. **for**(i=0;i<N;i++) 8. {     T[i]=v[i];  } 9. }   这两中写法，第二个（方法一）开辟了结构体里成员指针空间，如果free，不能free(T)，只能free(T.e);  同时，源代码中的菜单目录，可以更加清晰的反映不同模块的功能。  静态的内存使用的是栈空间内存,不用程序员自己来分配.动态内存由程序员根据需要来自己分配并收回.两者最大的区别在于动态的内存分配时候会用malloc或calloc函数。之所以要程序员自己来分配内存是由于有时候不能确定程序要使用多少内存,比如要通过用户或者文件或者数据库中的查询结果来确定使用多少数据,这时候程序员无法在程序的编写的时候就把内存给固定分配出来.这时候必须得让程序在运行的时候自己来为自己找到可用的内存,就一定要用动态的方式来分配内存. | | | |