Linux实验报告

1. 实验目的

了解并学习Linux系统的相关操作。

1. 实验设备

PC机（Linux系统）

1. 实验内容

通过数据结构实现B+树，红黑树，堆排序等操作。

1. 实验源代码

/\*

顺序表代码实现

\*/

#include "SqList.h"

int Init\_SqList(SqList \*S, int DataType)

{

void \*tPtr = NULL;//数据区临时指针

long i;

S->DataType = DataType;

switch (DataType)

{

case DATATYPE\_CHAR:

if (!(tPtr = (char\*)malloc(20 \* sizeof(char))))

return ERROR;

break;

case DATATYPE\_LONG:

if (!(tPtr = (long\*)malloc(20 \* sizeof(long))))

return ERROR;

break;

case DATATYPE\_FLOAT:

if (!(tPtr = (float\*)malloc(20 \* sizeof(float))))

return ERROR;

break;

case DATATYPE\_DOUBLE:

if (!(tPtr = (double\*)malloc(20 \* sizeof(double))))

return ERROR;

break;

case DATATYPE\_CSTRING:

if (!(tPtr = (char\*\*)malloc(20 \* sizeof(char\*))))

return ERROR;

for (i = 0; i < 20; i++)

((char\*\*)tPtr)[i] = NULL;//C字符串需要预先将指针清零

break;

default:

S->DataType = DATATYPE\_INT;

case DATATYPE\_INT:

if (!(tPtr = (int\*)malloc(20 \* sizeof(int))))

return ERROR;

break;

}

S->DataList = tPtr;

S->size = 20;

S->length = 0;

return NO\_ERROR;

}

int Free\_SqList(SqList \*S)

{

Clear\_SqList(S);//清空数据区

S->size = 0;

free(S->DataList);

S->DataList = NULL;

return NO\_ERROR;

}

int Clear\_SqList(SqList \*S)

{

long i;

if (S->DataType == DATATYPE\_CSTRING)

{//需要连带清空每个C字符串占有空间

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

if (((char\*\*)(S->DataList))[i])

free(((char\*\*)(S->DataList))[i]);

}

S->length = 0;

return NO\_ERROR;

}

int IsEmpty\_SqList(SqList \*S)

{

return !(S->length);

}

long GetLength\_SqList(SqList \*S)

{

return S->length;

}

void\* GetData\_SqList(SqList \*S, long pos)

{

if (pos < 0 || S->length >= pos)

return ERROR;//越界报错

switch (S->DataType)

{

case DATATYPE\_CHAR:

return (((char\*)(S->DataList)) + pos);

break;

case DATATYPE\_LONG:

return (((long\*)(S->DataList)) + pos);

break;

case DATATYPE\_FLOAT:

return (((float\*)(S->DataList)) + pos);

break;

case DATATYPE\_DOUBLE:

return (((double\*)(S->DataList)) + pos);

break;

case DATATYPE\_CSTRING:

return (((char\*\*)(S->DataList)) + pos);

break;

default:

case DATATYPE\_INT:

return (((int\*)(S->DataList)) + pos);

break;

}

}

long FindDataPos\_SqList(SqList \*S, void \*DataPtr)

{

long i;

for (i = 0; i < S->length; i++)

{

switch (S->DataType)

{

case DATATYPE\_CHAR:

if (((char\*)S->DataList)[i] == \*(char\*)DataPtr)

return i;

break;

case DATATYPE\_LONG:

if (((long\*)S->DataList)[i] == \*(long\*)DataPtr)

return i;

break;

case DATATYPE\_FLOAT:

if (((float\*)S->DataList)[i] == \*(float\*)DataPtr)

return i;

break;

case DATATYPE\_DOUBLE:

if (((double\*)S->DataList)[i] == \*(double\*)DataPtr)

return i;

break;

case DATATYPE\_CSTRING:

if (!strcmp(((char\*\*)S->DataList)[i], (char\*)DataPtr))

return i;

break;

default:

case DATATYPE\_INT:

if (((int\*)S->DataList)[i] == \*(int\*)DataPtr)

return i;

break;

}

}

if (i == S->length)

return ERROR;//没有这个数据

return i;

}

void\* GetNextData\_SqList(SqList \*S, long pos)

{

return GetData\_SqList(S, pos + 1);

}

void\* GetPrioData\_SqList(SqList \*S, long pos)

{

return GetData\_SqList(S, pos - 1);

}

long InsertData\_SqList(SqList \*S, long pos, void \*DataPtr)

{

long i;

void \*tPtr = NULL;//数据区临时指针

char \*tCStr = NULL;//用于插入的C字符串

if (pos < 0 || pos > S->length)

return ERROR;//越界报错

if (S->DataType == DATATYPE\_CSTRING)

{

if (!(tCStr = (char\*)malloc((strlen((char\*)DataPtr) + 10) \* sizeof(char))))

return ERROR;//C字符串内存开辟失败

memset(tCStr, 0, strlen((char\*)DataPtr) + 10);//临时C字符串初始化

strcpy(tCStr, (char\*)DataPtr);//复制为新的C字符串

}

if (S->length >= S->size - 2)

{//重新分配内存用于扩增

switch (S->DataType)

{

case DATATYPE\_CHAR:

if (!(tPtr = (char\*)realloc(S->DataList, (20 + S->size) \* sizeof(char))))

return ERROR;

break;

case DATATYPE\_LONG:

if (!(tPtr = (long\*)realloc(S->DataList, (20 + S->size) \* sizeof(long))))

return ERROR;

break;

case DATATYPE\_FLOAT:

if (!(tPtr = (float\*)realloc(S->DataList, (20 + S->size) \* sizeof(float))))

return ERROR;

break;

case DATATYPE\_DOUBLE:

if (!(tPtr = (double\*)realloc(S->DataList, (20 + S->size) \* sizeof(double))))

return ERROR;

break;

case DATATYPE\_CSTRING:

if (!(tPtr = (char\*\*)realloc(S->DataList, (20 + S->size) \* sizeof(char\*))))

return ERROR;

for (i = S->size; i < 20 + S->size; i++)

((char\*\*)tPtr)[i] = NULL;//C字符串需要预先将指针清零

break;

default:

S->DataType = DATATYPE\_INT;

case DATATYPE\_INT:

if (!(tPtr = (int\*)realloc(S->DataList, (20 + S->size) \* sizeof(int))))

return ERROR;

break;

}

S->DataList = tPtr;

S->size += 20;//每次固定扩增20个单位

}

for (i = S->length; i > pos; i--)

{//插入前后移

switch (S->DataType)

{

case DATATYPE\_CHAR:

((char\*)S->DataList)[i] = ((char\*)S->DataList)[i - 1];

break;

case DATATYPE\_LONG:

((long\*)S->DataList)[i] = ((long\*)S->DataList)[i - 1];

break;

case DATATYPE\_FLOAT:

((float\*)S->DataList)[i] = ((float\*)S->DataList)[i - 1];

break;

case DATATYPE\_DOUBLE:

((double\*)S->DataList)[i] = ((double\*)S->DataList)[i - 1];

break;

case DATATYPE\_CSTRING:

((char\*\*)S->DataList)[i] = ((char\*\*)S->DataList)[i - 1];

break;

default:

case DATATYPE\_INT:

((int\*)S->DataList)[i] = ((int\*)S->DataList)[i - 1];

break;

}

}

//插入数据

switch (S->DataType)

{

case DATATYPE\_CHAR:

((char\*)S->DataList)[i] = \*(char\*)DataPtr;

break;

case DATATYPE\_LONG:

((long\*)S->DataList)[i] = \*(long\*)DataPtr;

break;

case DATATYPE\_FLOAT:

((float\*)S->DataList)[i] = \*(float\*)DataPtr;

break;

case DATATYPE\_DOUBLE:

((double\*)S->DataList)[i] = \*(double\*)DataPtr;

break;

case DATATYPE\_CSTRING:

((char\*\*)S->DataList)[i] = tCStr;

break;

default:

case DATATYPE\_INT:

((int\*)S->DataList)[i] = \*(int\*)DataPtr;

break;

}

S->length++;

return pos;

}

int DeleteData\_SqList(SqList \*S, long pos)

{

long i;

if (pos < 0 || pos >= S->length)

return ERROR;//越界错误

if (S->DataType == DATATYPE\_CSTRING)

free(((char\*\*)S->DataList)[pos]);//C字符串型数据需释放字符串所占内存

for (i = pos; i < S->length - 1; i++)

{//前移

switch (S->DataType)

{

case DATATYPE\_CHAR:

((char\*)S->DataList)[i] = ((char\*)S->DataList)[i + 1];

break;

case DATATYPE\_LONG:

((long\*)S->DataList)[i] = ((long\*)S->DataList)[i + 1];

break;

case DATATYPE\_FLOAT:

((float\*)S->DataList)[i] = ((float\*)S->DataList)[i + 1];

break;

case DATATYPE\_DOUBLE:

((double\*)S->DataList)[i] = ((double\*)S->DataList)[i + 1];

break;

case DATATYPE\_CSTRING:

((char\*\*)S->DataList)[i] = ((char\*\*)S->DataList)[i + 1];

break;

default:

case DATATYPE\_INT:

((int\*)S->DataList)[i] = ((int\*)S->DataList)[i + 1];

break;

}

}

if (S->DataType == DATATYPE\_CSTRING)

((char\*\*)S->DataList)[S->length - 1] = NULL;//C字符串需特殊处理，防止中途Free/Clear时报错

S->length--;

return NO\_ERROR;

}

1. 实验小结

通过本次实验，了解并且学习了在Linux系统上的相关操作。