# 定义

# 存储

## 顺序存储

#include<stdio.h>

#include<stdlib.h>

#define MAX\_SIZE 128

typedef char String[MAX\_SIZE];

int StringIndex(String S, String T, int pos){

int i=pos;

int j=1;

while(i<S[0] && j<=TT[0]){

if(S[i]==T[i]){

++i;

++j;

}else{

i=i+1-(j-1);

j=i;

}

}

if(j>T[0])

return i-T[0];

else

return 0;

}

## 链式存储

# 方法

# 应用

## 递归求字符串的长度

思路：

字符串末尾结束表示：\0

编程：Window—DecC++

代码：

#include<stdio.h>

#include<stdlib.h>

int RecursiveStringLength(char \*ch)

{

if(ch[0] == '\0')

return 0;

else

return RecursiveStringLength(ch+1)+1;

}

int main()

{

char ch[] = "L love coding";

printf("char length is :%d\n",RecursiveStringLength(ch));

getchar();

}

**版本2：**

#include <stdio.h>

#include <stdlib.h>

/\*\*

递归的方式求字符串长度

\*/

int RecurLength(char\* str)

{

if(str[0] == '\0')

return 0;

else

return RecurLength(str+1)+1;

}

// 字符串中最后一个单词的长度

int LastWordLen(char\* str,int len)

{

int lastlen = 0;

int index = len-1;

while(index>=0 && str[index--] != ' ')

lastlen++;

return lastlen;

}

int LastWordLen2(char\* str)

{

int lastlen = 0;

int index =0;

while(str[index] !='\0')

{

if(str[index]==' ')

lastlen =0;

else

lastlen++;

index++;

}

return lastlen;

}

/\*

字符串内存的拷贝 实现memmove函数

\*/

char\* my\_memmove(char\* dst,char\* src,int count)

{

char\* ret = dst;

if(dst == NULL || src == NULL)

return NULL;

//如果dst和src区域没有重叠，那么从开始处逐一拷贝

if(dst <= src || dst >= (src+count))

{

while(count--)

{

\*dst = \*src;

dst++;

src++;

}

}

//如果dst和src区域有交叉，那么从尾部开始向起始位置拷贝，这样可

以避免数据重叠

else

{

dst = dst + count-1;

src = src + count-1;

while(count--)

{

\*dst = \*src;

dst--;

src--;

}

}

return ret;

}

int main()

{

char str[]="jike xueyuan";

printf("str len is %d\r\n",RecurLength(str));

// printf("lastword len is %d\r\n",LastWordLen(str,12));

printf("lastword len is %d\r\n",LastWordLen2(str));

return 0;

}

## 字符串最后一个单词长度

思路：

整个字符串包含多个单词

每个单词中间使用空格隔开

字符串中只有单词，没有别的字符

代码：

## 实现memmove()

## 字符串查找

代码：

#include <stdio.h>

#include <stdlib.h>

// 查找字符串中各个字符出现的次数

void FindTimes(char\* str)

{

int hash[256]={0};

int index =0;

while(str[index]!='\0')

{

hash[str[index]]++;

index++;

}

for(index=0;index<256;index++)

printf("%d ",hash[index]);

printf("\r\n");

}

// 如何翻转一个句子，将整个句子以单词为单位进行翻转

/\*\*

首先完成整个句子的翻转

然后完成内部单词的二次翻转

\*/

//对字符串中固定位置的区域进行翻转

void Reverse(char\* str,int begin,int end)

{

char tmp;

while(begin < end)

{

tmp = str[begin];

str[begin] = str[end];

str[end] = tmp;

begin++;

end--;

}

}

/\*

字符串内部单词的翻转 从头部开始 每确定一个单词就进行翻转

\*/

void Reverse\_second(char\* str,int len)

{

int begin,i,j;

for(i=0;i<len;)

{

begin =i;

for(j=i;j<len;j++)

if(str[j]==' ')

break;

Reverse(str,begin,j-1);

i = j+1;

}

}

int main()

{

char str[]="jike xue yuan huan ying ni";

// FindTimes(str);

Reverse(str,0,25);

Reverse\_second(str,26);

printf("%s \r\n",str);

return 0;

}

## 字符串排序

题目要求：一个字符串中包含着大小写的英文字符，排序后使得大写字符在前，小写字符在后。

如果需要保持原始小写字符的相对位置该怎么办？

代码：

#include <stdio.h>

#include <stdlib.h>

// 排序字符串 使得字符串中大写字符在前 小写字符在后

void StrSort(char\* str)

{

int index =-1;

int pos =0;

char temp;

for(;str[pos]!='\0';pos++)

{

if(str[pos] >= 'A' && str[pos]<='Z')

{

index++;

temp = str[index];

str[index] = str[pos];

str[pos]= temp;

}

}

}

// 同样的要求 但是使得小写字符的相对顺序不改变

void StrSort2(char\* str)

{

int len =0;

while(str[len]!='\0')

len++;

int index =len;

int pos = len-1;

char temp;

for(;pos>=0;pos--)

{

if(str[pos] >= 'a' && str[pos] <= 'z')

{

index--;

temp = str[index];

str[index]=str[pos];

str[pos] = temp;

}

}

}

int main()

{

char str[] ="bdAFfdDAfdAfaFAfaGSdDFfdsaGd";

printf("%s\n",str);

StrSort2(str);

printf("%s\n",str);

return 0;

}

## 字符串去重

题目要求：在一个已经排序好的字符串中，经过处理，得到重复的字符保留一个。

代码：

#include <stdio.h>

#include <stdlib.h>

// 一个已经排序好的字符串 将重复的字符串保留一个

void Remove(char\* str)

{

int index =0;

int pos =1;

for(;str[pos]!='\0';pos++)

{

if(str[pos] != str[index])

{

index++;

str[index]= str[pos];

}

}

str[index+1] ='\0';

}

// 一个已经排序好的字符串 将重复的字符保留两个

void Remove2(char\* str)

{

int index =0;

int pos =1;

int flag =0;

for(;str[pos]!='\0';pos++)

{

//如果当前位置的字符和已经保存的字符的最后一个位置的字符不同

if(str[pos] != str[index])

{

index++;

str[index] = str[pos];

flag=0;

}

//如果当前位置的字符和已经保存的字符的最后一个位置的字符相同

else

{

if(flag == 0)

{

index++;

str[index] = str[pos];

flag =1;

}

}

}

str[index+1]='\0';

}

//重复的字符全部删除

void Remove3(char\* str)

{

int flag =0;

int index =-1;

int pos;

//遍历所有字符

for(pos=0;str[pos+1]!='\0';)

{

//如果当前字符和下一个字符相同

if(str[pos] == str[pos+1])

{

flag =1;

pos++;

}

//如果当前字符和下一个字符不同

else

{

if(flag == 1)

{

flag =0;

pos++;

}

else

{

index++;

str[index] = str[pos];

pos++;

}

}

}

if(str[pos]!= str[pos-1])

str[++index] = str[pos];

str[index+1] = '\0';

}

int main()

{

char str[]="AAABCCCDFGGIIIKKLMNNOOPQQQQ";

printf("%s\n",str);

Remove3(str);

printf("%s\n",str);

return 0;

}

拓展：

如何将重复的字符全部删除？

如何将重复的字符都保留两个？

## atoi()实现

代码：

#include <iostream>

#include <vector>

#include <string>

#include <limits>

using namespace std;

/\*

实现atoi函数

需要注意的地方：

1.数值越界

2.正负数

3.字符串结束标志

\*/

int StringToInt(string& str)

{

if(str.length() == 0)

return 0;

int i;

int flag =1;

int result =0;

int digit;

// 清空空格字符

for(i=0;i<str.length();i++)

if(str[i] !=' ')

break;

//查找正负号

if(str[i] == '+')

i++;

if(str[i]== '-')

{

flag =0;

i++;

}

//开始处理字符

for(;i<str.length();i++)

{

if(str[i]<'0' || str[i] > '9')

break;

digit = str[i]-'0';

//判断越界

if(flag && (numeric\_limits<int>::max() - result\*10) <= digit)

return numeric\_limits<int>::max();

else if(!flag && (numeric\_limits<int>::min() + result\*10) >= digit\*-1)

return numeric\_limits<int>::min();

result = result\*10+digit;

}

return flag ==1? result:-result;

}

int main()

{

string str("-4324324878787998797");

cout<<StringToInt(str)<<endl;

return 0;

}