

## 一. 实验任务

1. C 语言字符串
2. 顺序串
3. 串的模式匹配算法

## 二. 实验上机时间

## 三. 知识点

1. C 语言字符串  
各种输入输出方法
2. 顺序串
  - (1) 根据串常量 chars 生成串 T
  - (2) 求子串
  - (3) 销毁串 S
  - (4) 清空串 S
  - (5) 把串 S 的内容拷贝到串 T
  - (6) 子串定位
  - (7) 求串长
  - (8) 把串 S 中符合 T 的子串替换
  - (9) 比较串 S 和 T
  - (10) 插入子串
  - (11) 联接串
  - (12) 判断串是否空
  - (13) 打印串
3. 串的模式匹配算法
  - (1) 朴素模式匹配算法 BF
  - (2) KMP 算法

## 四. 源代码及结果截屏

### 1. C 语言字符串

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>

int main()
{
    char a[10];
    printf("请输入长度为10的字符串: ");
    scanf("%s", a);
    char b[] = "CHINA";
    char* xx = NULL;
    xx = (char*)malloc(10);
    strcpy(xx, "china");
    printf("%s\n", a);
    printf("%s\n", b);
    printf("%s\n", xx);
}
```

```

printf("%d\n", strlen(xx));

system("pause");
return 0;
}

```

CS E:\大二上\数据结构\代码保存\String\Debug\String.exe

```

请输入长度为10的字符串: ABCDEFGHIJ
ABCDEFGHIJ
CHINA
china
5
请按任意键继续. . .

```

## 2. 顺序串

### (1) String.h

```

#ifndef STRING_H
#define STRING_H

#include <stdlib.h>
#include "Status.h"

#define MAXSTRLEN 255//用户预定义最大串长
typedef unsigned char SString[MAXSTRLEN + 1]; //0号单元存放串的长度

Status StrAssign(SString T, char* chars); //根据串常量chars生成串T
Status SubString(SString Sub, SString S, int pos, int len); //求子串
void DestroyString(SString S); //销毁串S
Status ClearString(SString S); //清空串S
Status StrCopy(SString T, SString S); //把串S的内容拷贝到串T
int Index(SString S, SString T, int pos); //子串定位
int StrLength(SString S); //求串长
Status Replace(SString S, SString T, SString V); //把串S中符合T的子串替换
int StrCompare(SString S, SString T); //比较串S和T
Status StrInsert(SString S, int pos, SString T); //插入子串
Status Concat(SString T, SString S1, SString S2); //联接串
Status StrEmpty(SString S); //判断串是否空
void StrPrint(SString T); //打印串

#endif

```

### (2) String.c

```

#include "String.h"
#include "Status.h"
#include <stdio.h>

```

```

#include <malloc.h>

Status StrAssign(SString T, char* chars)
{
    //生成一个其值等于chars的串T
    int i = 0;
    if (strlen(chars) > MAXSTRLEN)
        return ERROR;
    else
    {
        T[0] = strlen(chars);
        for (i = 1; i <= T[0]; i++)
            T[i] = *(chars + i - 1);
        return OK;
    }
}

Status SubString(SString Sub, SString S, int pos, int len)
{
    //用Sub返回串S的第pos个字符起长度为len的子串
    int i;
    if (pos<1 || pos>S[0] || len<0 || len>S[0] - pos + 1)
        return ERROR;
    for (i = 1; i <= len; i++)
        Sub[i] = S[pos + i - 1];
    Sub[0] = len;
    return OK;
}

void DestroyString(SString S)
{
    //由于SString是定长类型,无法销毁
}

Status ClearString(SString S)
{
    //初始条件:串S存在。操作结果:将S清为空串
    S[0] = 0; //令串长为零
    return OK;
}

Status StrCopy(SString T, SString S)
{
    //由串S复制得串T

```

```

    int i;
    for (i = 0; i <= S[0]; i++)
        T[i] = S[i];
    return OK;
}

int Index(SString S, SString T, int pos)
{
    //返回子串T在主串S中第pos个字符之后的位置。若不存在,则函数值为0
    //其中,T非空,1≤pos≤StrLength(S)
    int i, j;
    if (1 <= pos && pos <= S[0])
    {
        i = pos;
        j = 1;
        while (i <= S[0] && j <= T[0])
            if (S[i] == T[j]) //继续比较后继字符
            {
                ++i;
                ++j;
            }
            else //指针后退重新开始匹配
            {
                i = i - j + 2;
                j = 1;
            }
        if (j > T[0])
            return i - T[0];
        else
            return 0;
    }
    else
        return 0;
}

int StrLength(SString S)
{
    //返回串的元素个数
    return S[0];
}

Status Replace(SString S, SString T, SString V)
{
    //初始条件: 串S,T和V存在,T是非空串(此函数与串的存储结构无关)

```

```

//操作结果：用V替换主串S中出现的所有与T相等的不重叠的子串 */
int i = 1; //从串S的第一个字符起查找串T
if (StrEmpty(T)) //T是空串
    return ERROR;
do
{
    i = Index(S, T, i); //结果i为从上一个i之后找到的子串T的位置
    if (i) //串S中存在串T
    {
        StrDelete(S, i, StrLength(T)); //删除该串T
        StrInsert(S, i, V); //在原串T的位置插入串V
        i += StrLength(V); //在插入的串V后面继续查找串T
    }
} while (i);
return OK;
}

int StrCompare(SString S, SString T)
{
    //初始条件：串S和T存在
    //操作结果：若S>T,则返回值>0;若S=T,则返回值=0;若S<T,则返回值<0
    int i;
    for (i = 1; i <= S[0] && i <= T[0]; ++i)
    {
        if (S[i] != T[i])
            return S[i] - T[i];
    }
    return S[0] - T[0];
}

Status StrInsert(SString S, int pos, SString T)
{
    //初始条件：串S和T存在, 1≤pos≤StrLength(S)+1
    //操作结果：在串S的第pos个字符之前插入串T。完全插入返回TRUE, 部分插入返回FALSE
    int i;
    if (pos<1 || pos>S[0] + 1)
        return ERROR;
    if (S[0] + T[0] <= MAXSTRLEN)
    {
        //完全插入
        for (i = S[0]; i >= pos; i--)
            S[i + T[0]] = S[i];
        for (i = pos; i < pos + T[0]; i++)

```

```

        S[i] = T[i - pos + 1];
    S[0] = S[0] + T[0];
    return TRUE;
}
else
{
    //部分插入
    for (i = MAXSTRLEN; i <= pos; i--)
        S[i] = S[i - T[0]];
    for (i = pos; i < pos + T[0]; i++)
        S[i] = T[i - pos + 1];
    S[0] = MAXSTRLEN;
    return FALSE;
}
}

Status Concat(SString T, SString S1, SString S2)
{
    //用T返回联接而成的新串，若未截断，则返回TRUE, 否则FALSE
    int i = 0;
    int uncut = 0;
    if (S1[0] + S2[0] <= MAXSTRLEN) //未截断
    {
        for (i = 1; i <= S1[0]; i++)
            T[i] = S1[i];
        for (i = 1; i <= S2[0]; i++)
            T[i+S1[0]] = S2[i];
        T[0] = S1[0] + S2[0];
        uncut = TRUE;
    }
    else if (S1[0] < MAXSTRLEN) //截断
    {
        for (i = 1; i <= S1[0]; i++)
            T[i] = S1[i];
        for (i = S1[0] + 1; i <= MAXSTRLEN; i++)
            T[i] = S2[i - S1[0]];
        T[0] = MAXSTRLEN;
        uncut = FALSE;
    }
    else //截断
    {
        for (i = 0; i < +MAXSTRLEN; i++)
            T[i] = S1[i];
        uncut = FALSE;
    }
}

```

```

    }
}

Status StrDelete(SString S, int pos, int len)
{
    //初始条件：串S存在,  $1 \leq \text{pos} \leq \text{StrLength}(S) - \text{len} + 1$ 
    //操作结果：从串S中删除第pos个字符起长度为len的子串
    int i;
    if (pos < 1 || pos > S[0] - len + 1 || len < 0)
        return ERROR;
    for (i = pos + len; i <= S[0]; i++)
        S[i - len] = S[i];
    S[0] -= len;
    return OK;
}

Status StrEmpty(SString S)
{
    //若S为空串, 则返回TRUE, 否则返回FALSE
    if (S[0] == 0)
        return TRUE;
    else
        return FALSE;
}

void StrPrint(SString T)
{
    //输出字符串T
    int i;
    for (i = 1; i <= T[0]; i++)
        printf("%c", T[i]);
    printf("\n");
}

```

### (3) Test.c

```

#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include "String.h"

int main()
{
    int i, j, *p;
    SString S1, S2, T;

```

```

//StrAssign & StrPrint test
StrAssign(S1, "acabaabaabcacacabc");
printf("S1串为: ");
StrPrint(S1);
StrAssign(S2, "abaabcac");
printf("S2串为: ");
StrPrint(S2);
//SubString test
SString Sub;
SubString(&Sub, S1, 3, 5);
printf("串S1的第3个字符起长度为5的子串为:");
StrPrint(Sub);
//StrCopy test
SString A;
StrCopy(&A, S2); //把串S的内容拷贝到串T
printf("拷贝S2后A串为: ");
StrPrint(A);
//Index test
int pos = 1;
printf("子串S2在主串S1中第%d个字符之后的位置\n", Index(S1, S2, pos));
//StrLength test
printf("子串S2长度为: %d\n", StrLength(S2));
//Replace test(含StrDelete & StrInsert)
SString V;
StrAssign(T, "cac");
StrAssign(V, "ded");
Replace(&S1, T, V);
printf("替换后S1串为:");
StrPrint(S1);
//StrCompare test
if (StrCompare(S1, S2) > 0)
    printf("S1更长\n");
else if (StrCompare(S1, S2) == 0)
    printf("S1S2一样长\n");
else
    printf("S2更长\n");
//Concat test
Concat(&T, S1, S2);
printf("concat后T串为: ");
StrPrint(T);

system("pause");
return 0;
}

```



E:\大二上\数据结构\代码保存\String\Debug\String.exe

```
S1串为: acabaabaabcacaabc
S2串为: abaabcac
串S1的第3个字符起长度为5的子串为: abaab
拷贝S2后A串为: abaabcac
子串S2在主串S1中第6个字符之后的位置
子串S2长度为: 8
替换后S1串为: acabaabaabdedaabc
S1更长
concat后T串为: acabaabaabdedaabcabaabcac
请按任意键继续. . .
```

### 3. 串的模式匹配算法

- (1) 朴素模式匹配算法 BF
- (2) KMP 算法

#### A. StringIndex.h

```
#ifndef STRINGINDEX_H
#define STRINGINDEX_H

#include <stdlib.h>
#include "Status.h"

#define MAXSTRLEN 255//用户预定义最大串长
typedef unsigned char SString[MAXSTRLEN + 1]; //0号单元存放串的长度

int Index(SString S, SString T, int pos);
void get_next(SString T, int *next[]);
int Index_KMP(SString S, SString T, int pos);
Status StrAssign(SString T, char* chars);
void StrPrint(SString T);

#endif
```

#### B. StringIndex.c

```
#include "Stringindex.h"
#include "Status.h"
#include <stdio.h>
#include <malloc.h>

int Index(SString S, SString T, int pos)
```

```

{
    //返回子串T在主串S中第pos个字符之后的位置。若不存在,则函数值为0
    //其中,T非空,  $1 \leq \text{pos} \leq \text{StrLength}(S)$ 
    int i, j;
    if (1 <= pos && pos <= S[0])
    {
        i = pos;
        j = 1;
        while (i <= S[0] && j <= T[0])
            if (S[i] == T[j]) //继续比较后继字符
            {
                ++i;
                ++j;
            }
            else //指针后退重新开始匹配
            {
                i = i - j + 2;
                j = 1;
            }
        if (j > T[0])
            return i - T[0];
        else
            return 0;
    }
    else
        return 0;
}

```

```

void get_next(SString T, int next[])
{
    int i = 1;
    next[1] = 0;
    int j = 0;
    while (i < T[0])
    {
        if (j == 0 || T[i] == T[j])
        {
            ++i;
            ++j;
            next[i] = j;
        }
        else
            j = next[j];
    }
}

```

```

}

int Index_KMP(SString S, SString T, int pos)
{
    //利用算法模式串T的next函数求T在主串S中第pos个字符之后的位置的KMP算法
    //其中, T非空, 1<=pos<=StrLength(S)
    int next[255] = { "0"};
    int i = pos;
    int j = 1;//j用于子串T中当前位置下标值
    get_next(T, next);
    while (i <= S[0] && j <= T[0])
    {
        if (j == 0 || S[i] == T[j])
        {
            ++i;
            ++j;
        }
        else
            j = next[j];
    }
    if (j > T[0])
        return i - T[0];
    else
        return 0;
}

Status StrAssign(SString T, char* chars)
{
    //生成一个其值等于chars的串T
    int i = 0;
    if (strlen(chars) > MAXSTRLEN)
        return ERROR;
    else
    {
        T[0] = strlen(chars);
        for (i = 1; i <= T[0]; i++)
            T[i] = *(chars + i - 1);
        return OK;
    }
}

void StrPrint(SString T)
{
    //输出字符串T

```

```

    int i;
    for (i = 1; i <= T[0]; i++)
        printf("%c", T[i]);
    printf("\n");
}

```

### C. test.c

```

#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include "Stringindex.h"
#include <windows.h>
#include <time.h>

int main()
{
    int i, j, * p;
    SString S1, S2, T;
    StrAssign(S1, "acabaabaabcacaabc\0");
    printf("S1串为: ");
    StrPrint(S1);
    StrAssign(S2, "abaabcac\0");
    printf("S2串为: ");
    StrPrint(S2);

    int pos = 1;
    //Index test
    long op, ed; //定义开始时间和结束时间
    op = clock(); //开始计时
    for (int i = 0; i < 10000; i++)
    {
        Index(S1, S2, pos);
    }
    Sleep(1000); //静态方法，控制当前正在运行的线程
    ed = clock(); //结束计时
    printf("运行时间为%ldms\n", ed - op); //运行时间

    printf("子串S2在主串S1中第%d个字符之后的位置\n", Index(S1, S2, pos));

    pos = 1;
    //Index_KMP test
    op = clock(); //开始计时
    for (int i = 0; i < 10000; i++)

```

```

{
    Index_KMP(S1, S2, pos);
}

Sleep(1000); //静态方法，控制当前正在运行的线程
ed = clock(); //结束计时
printf("KMP运行时间为%ldms\n", ed - op); //运行时间
printf("子串S2在主串S1中第%d个字符之后的位置\n", Index_KMP(S1, S2, pos));

system("pause");
return 0;
}

```

CA E:\大二上\数据结构\代码保存\Stringindex\Debug\Stringindex.exe

```

S1串为: acabaabaabcacaabc
S2串为: abaabcac
运行时间为1008ms
子串S2在主串S1中第6个字符之后的位置
KMP运行时间为1006ms
子串S2在主串S1中第6个字符之后的位置
请按任意键继续. . .


```

## 五 . 实验总结

1. 0x7BA2EF8C (ucrtbased.dll)处(位于 String.exe 中)引发的异常: 0xC0000005: 写入位置 0x00350000 时发生访问冲突。

vs2019 下 scanf()函数报错

在 vs2019 环境下直接使用 scanf()函数，程序运行报错。

 C4996 'scanf': This function or variable may be unsafe. Consider using scanf\_s instead. To disable deprecation, use \_CRT\_SECURE\_NO\_WARNINGS. See online help for details.

### 解决方法 1

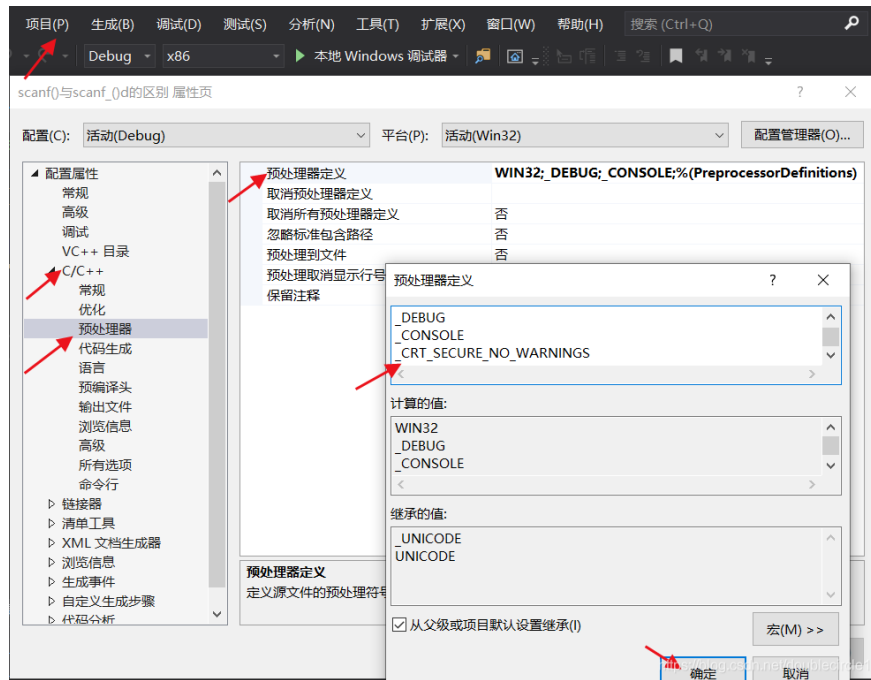
使用 scanf\_s()函数来代替 scanf()函数

在调用该函数时，必须提供一个数字以表明最多读取多少位字符。

比如：scanf("%s",a,sizeof(a))

### 解决方法 2

打开项目→属性→C/C++→预处理器→预处理器定义→添加\_CRT\_SECURE\_NO\_WARNINGS



解决方法 3

加入宏定义: `#define _CRT_SECURE_NO_WARNINGS`

2. 学习函数 `strlen(str)` `strcpy(str,"china")` 添加头文件 `#include <string.h>`  
区别 C 语言和 C++ 语言: C 语言中没有 `string`

3. 关于字符串的赋值

- (1) `char c[10];`  
`c="hello";` ✗ (不可以给 `c` 直接进行赋值)
- (2) `char c[10] = "hello";` ✓
- (3) `char *c;`  
`c=(char*) malloc (10 * sizeof (char));`  
`c="hello";` ✓
- (4) `char *c = NULL;`  
`c="hello";` ✓
- (5) `char *c;`  
`c=(char*) malloc (10 * sizeof (char));`  
`strcpy (c,"hello");` ✓
- (6) `char c[10];`  
`strcpy (c,"hello");` ✓
- (7) `char c[10];`  
`scanf("%s",c);` ✓

- 4.