实现顺序串的各种模式匹配

姓名：凌强 班级：21计算机二班 学号：2021413579

实验目的：

1. 掌握顺序串的基本运算。
2. 掌握串的模式匹配算法即BF(Brute Force)和[KMP](https://so.csdn.net/so/search?q=KMP&spm=1001.2101.3001.7020" \t "_blank)(克努特-莫里斯-普拉特)算法设计
3. 掌握顺序串的概念，学会对顺序存储数据结构进行操作。
4. 加深对顺序存储数据结构的理解，逐步培养解决实际问题的编程能力。

实验内容：

1. 初始化顺序串，接收输入的数据，将以顺序串方式存储。
2. 通过输入的数字序号选择对应的功能以完成顺序串的BF、KMP模式匹配等操作。

# **需求分析**

1. 完成对顺序串的创建、初始化、BF、KMP模式匹配等基本操作。
2. 演示程序中，规定 ElemType 为 char[]型。
3. 演示程序以用户和计算机的对话方式执行，在运行框中出现提示信息之后，由用户在键盘上输入完成相关命令的执行；相应结果显示在其后。
4. 程序执行的命令包括：
   1. 输出用户界面。
   2. 读取用户输入的数据，创建顺序串，并求长度赋值给length。
   3. 输入数字选择对应的功能。
   4. BF模式匹配算法1。
   5. BF模式匹配算法2。
   6. KMP模式匹配算法（nextval）。
   7. 销毁顺序串。
   8. 退出程序。
5. 测试数据：见（[六、测试结果](#_测试结果)）。

# **概要设计**

1. 顺序串的抽象数据定义：

typedef struct

{

    char str[MaxSize];

    int length;

} SqString;

1. 基本操作：

*//求长度并记录*

void StrLength(SqString \**S*)

*//BF算法1*

int index2(SqString *str*, SqString *substr*)

*//BF算法2*

int index(SqString *S*, SqString *T*)

void GetNextval(SqString *t*, int *nextval*[])

*//KMP算法*

int KMPIndex(SqString *s*, SqString *t*)

*//主界面*

void welcome()

*//BF测试1*

void BF()

*//BF测试2*

void BF2()

*//KMP测试1*

void KMP()

1. 程序包含两个模块：
2. 主函数模块：

用于与测试用户交换数据。

1. 操作函数模块：

用于指引测试用户下一步操作，操作链表。

# **详细设计**

## 顺序串的定义

*#include* <iostream>

using namespace std;

*#define* MaxSize 200

typedef struct

{

    char str[MaxSize];

    int length;

} SqString;

## 操作函数

*//求长度并记录*

void StrLength(SqString \**S*)

{

*S*->length = 0;

*while* (*S*->str[*S*->length++] != '\0')

        ;

*S*->length--;

}

*//BF算法1*

int index2(SqString *str*, SqString *substr*)

{

    int i, j, k;

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

    {

*for* (j = i, k = 0; *str*.str[j] == *substr*.str[k];

            j++, k++);

*// cout<<i<<" "<<k<<endl;*

*if* (k == *substr*.length+1)

*//注意j每次从i开始,有回溯*

*return*  i+1;

    }

*return* -1;

}

*//BF算法2*

int index(SqString *S*, SqString *T*)

{

    int i = 0, j = 0;

*while* (i < *S*.length && j < *T*.length)

    {

*if* (*S*.str[i] == *T*.str[j]) *//继续匹配下一个字符*

        {

            i++;

            j++;

        }

*else //主串、子串指针回溯重新开始下一次匹配*

        {

            i = i - j + 1; *//主串从下一个位置开始匹配*

            j = 0; *//子串从头开始匹配*

        }

    }

*if* (j >= *T*.length)

    {

*return* (i - *T*.length+1); *//返回匹配的第一个字符的下标*

    }

*else //模式匹配不成功*

    {

*return* (-1);

    }

}

void GetNextval(SqString *t*, int *nextval*[])

{

    int j = 0, k = -1;

*nextval*[0] = -1;

*while* (j < *t*.length)

    {

*if* (k == -1 || *t*.str[j] == *t*.str[k])

        {

            j++;

            k++;

*if* (*t*.str[j] != *t*.str[k])

*nextval*[j] = k;

*else*

*nextval*[j] = *nextval*[k];

        }

*else*

            k = *nextval*[k];

    }

}

*//KMP算法*

int KMPIndex(SqString *s*, SqString *t*)

{

    int nextval[MaxSize], i = 0, j = 0;

    GetNextval(*t*, nextval);

*while* (i < *s*.length && j < *t*.length)

    {

*if* (j == -1 || *s*.str[i] == *t*.str[j])

        {

            i++;

            j++;

        }

*else*

            j = nextval[j];

    }

*if* (j >= *t*.length)

*return* (i - *t*.length+1);

*else*

*return* (-1);

}

*//主界面*

void welcome()

{

    system("cls");

    printf("〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓\n");

    printf("〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓\n");

    printf("〓〓〓〓〓〓〓〓〓〓  ☆      顺序串的模式匹配      ☆  〓〓〓〓〓〓〓〓〓〓\n");

    printf("〓〓〓〓〓〓★★★★★         ★★★★★★★         ★★★★★〓〓〓〓〓〓\n");

    printf("〓〓〓〓〓〓★★★★★         ★★★★★★★         ★★★★★〓〓〓〓〓〓\n");

    printf("〓〓〓〓〓〓〓〓〓★  ☆          1.BF算法          ☆  ★〓〓〓〓〓〓〓〓〓\n");

    printf("〓〓〓〓〓〓〓〓〓★  ☆          2.BF算法2         ☆  ★〓〓〓〓〓〓〓〓〓\n");

    printf("〓〓〓〓〓〓〓〓〓★  ☆          3.KMF算法         ☆  ★〓〓〓〓〓〓〓〓〓\n");

    printf("〓〓〓〓〓〓〓〓〓★  ☆          0.退出系统        ☆  ★〓〓〓〓〓〓〓〓〓\n");

    printf("〓〓〓〓〓〓〓〓〓★  ☆                              ☆  ★〓〓〓〓〓〓〓〓\n");

    printf("〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓\n");

    cout << "初始化成功！" << endl;

}

*//BF测试1*

void BF()

{

    cout << "请输入要测试的原字符串和模板字符串:" << endl;

    SqString S, T;

    cin >> S.str >> T.str;

    StrLength(&S);

    StrLength(&T);

*if* (index(S, T) != -1)

    {

        cout << "匹配成功！" << endl;

        cout << "对应下标为:" << index(S, T)<<endl;

*return*;

    }

    cout << "匹配失败！" << endl;

*return*;

}

*//BF测试2*

void BF2()

{

    cout << "请输入要测试的原字符串和模板字符串:" << endl;

    SqString S, T;

    cin >> S.str >> T.str;

    StrLength(&S);

    StrLength(&T);

*if* (index2(S, T) != -1)

    {

        cout << "匹配成功！" << endl;

        cout << "对应下标为:" << index2(S, T)<<endl;

*return*;

    }

    cout << "匹配失败！" << endl;

*return*;

}

*//KMP测试1*

void KMP(){

    cout << "请输入要测试的原字符串和模板字符串:" << endl;

    SqString S, T;

    cin >> S.str >> T.str;

    StrLength(&S);

    StrLength(&T);

*if* (KMPIndex(S, T) != -1)

    {

        cout << "匹配成功！" << endl;

        cout << "对应下标为:" << KMPIndex(S, T)<<endl;

*return*;

    }

    cout << "匹配失败！" << endl;

*return*;

}

## 主函数完善

int main()

{

    welcome();

*while* (1)

    {

        int choice;

        cout << "请输入需要实现的功能序号:";

        cin >> choice;

*switch* (choice)

        {

*case* 1:

            BF();

*continue*;

*case* 2:

            BF2();

*continue*;

*case* 3:

            KMP();

*continue*;

*default*:

*break*;

        }

    }

    system("pause");

*return* 0;

}

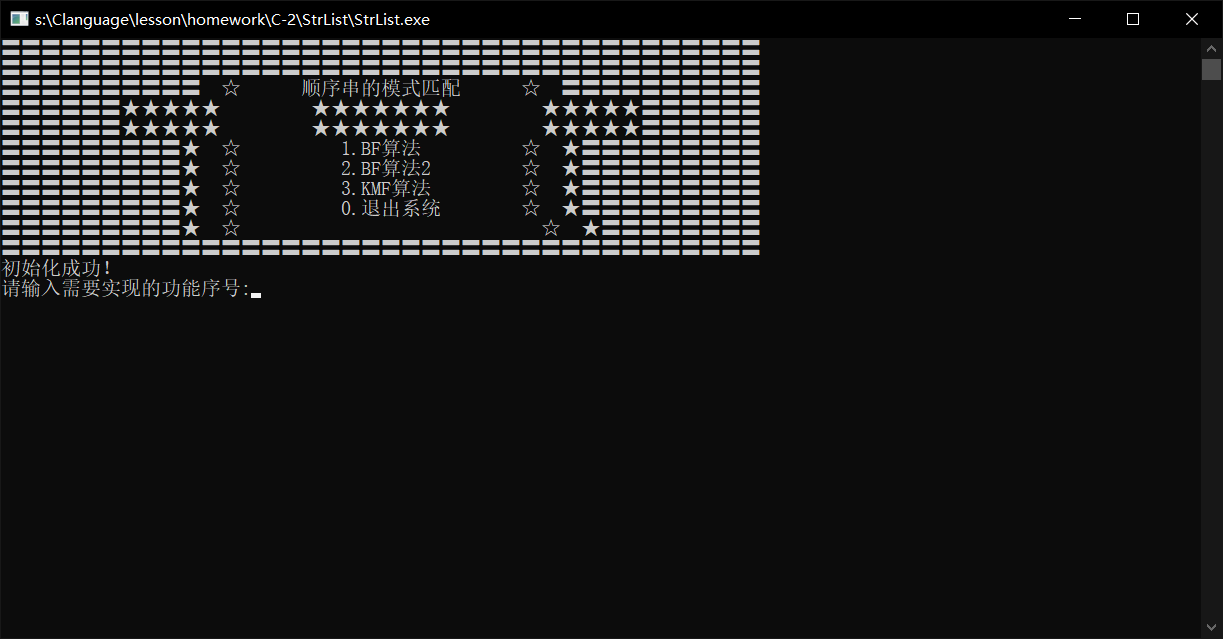
# **调试分析**

1. 模块划分较合理。
2. BF采用回溯法记录i值
3. BF算法最坏情况下时间复杂度为O(n\*m)，最好为O(m)
4. KMP算法中求next数组的时间复杂度为O(m),总的时间复杂度为O(n+m)
5. 本次实验报告完成了对顺序串的各种模式匹配表示，是一次良好的程序设计训练。

# **用户使用说明**

程序运行环境为vscode,执行文件为StrList.cpp

用户界面：



先选择功能，再根据算法输入原字符串和模板字符串，最后输出结果。

# **测试结果**

## 顺序串



# **实验体会**

通过这次实验，学习了顺序串定义和各种基本操作，并且了解到了其应用领域，大致总结一下吧：

在BF算法中，当主串和子串不匹配的时候，主串和子串你的指针都需要回溯，所以导致了该算法时间复杂度比较高为 O(nm) ，空间复杂度为 O(1) 注：虽然其时间复杂度为 O(nm) 但是在一般应用下执行，其执行时间近似 O(n+m) 所以仍被使用

KMP算法，利用子串的结构相似性，设计next数组，在此之上达到了主串不回溯的效果，大大减少了比较次数，但是相对应的却牺牲了存储空间，KMP算法 时间复杂度为 O(n+m) 空间复杂度为 O(n)

# **附录**

源程序：

StrList.cpp