

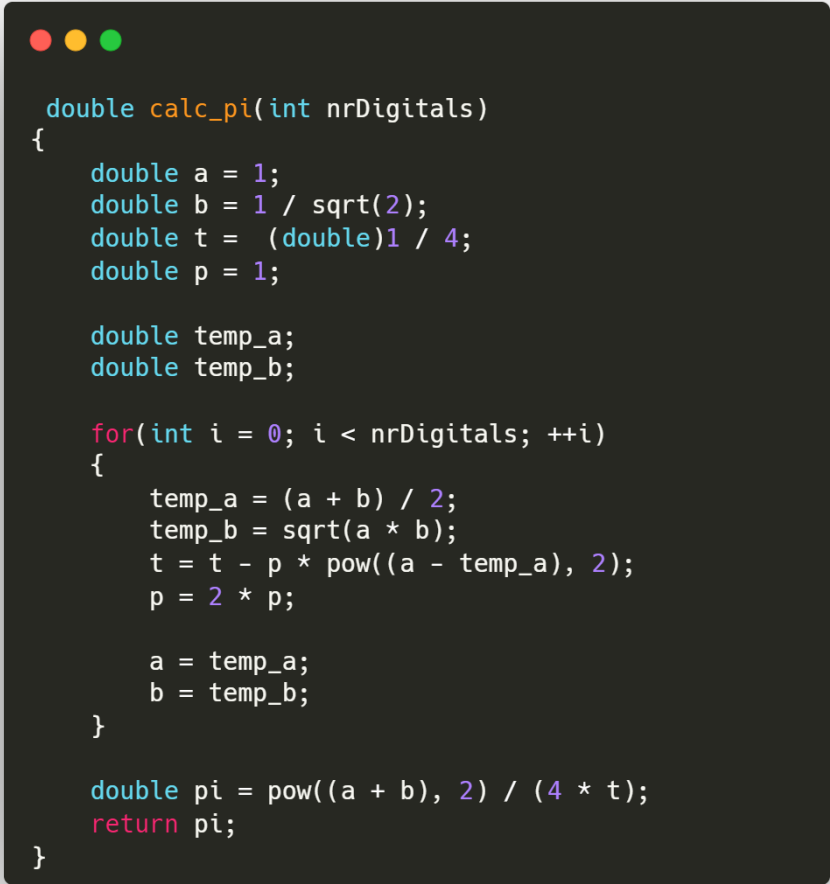
# 《数据结构与算法》实验报告

实验名称	KMP 算法的实现和应用				
姓名	叶鹏	学号	20020007095	日期	2022/4/15
实验内容	<ol style="list-style-type: none"><li>1. 调通圆周率生成算法，生成尽可能长的圆周率序列。</li><li>2. 利用自己实现的 KMP 算法比对圆周率序列中是否存在某个特殊序列并给出位置序号，比如本课程 ID，02003048；自己的生日，YYYYMMDD。</li></ol>				
实验目的	生成函数生成圆周率，并在此基础上实现 KMP 算法查找自己生日				

实验步骤	<p>1. 圆周率的生成</p> <p>在数学上圆周率的生成有很多种算法，如</p> <p>快速收敛级数：</p> $\frac{1}{\pi} = \frac{12}{640320^{3/2}} \sum_{k=0}^{\infty} \frac{(6k)!(13591409 + 545140134k)}{(3k)!(k!)^3(-640320)^{3k}}.$ <p>蒙特卡洛方法：</p> $\pi \approx \frac{2n\ell}{mt}$ <p>贝利-波尔温-普劳夫公式：</p> $\pi = \sum_{k=0}^{\infty} \frac{1}{16^k} \left( \frac{4}{8k+1} - \frac{2}{8k+4} - \frac{1}{8k+5} - \frac{1}{8k+6} \right)$ <p>诸如此类，在普通计算机编程中很少用到很高精度的PI，不过还是有办法计算其近似值</p> <ul style="list-style-type: none"><li>➤ 使用 <code>acos()</code>函数 <code>double pi = 2*acos(0.0);</code></li><li>➤ 使用 <code>asin()</code>函数 <code>double pi = 2*asin(1.0);</code></li><li>➤ 使用内建常量 <code>M_PI</code>（）（c++20）</li><li>➤ 其他数学公式</li></ul>
------	---



```
#include<iostream>
using namespace std;
int main()
{
    int c=0;
    double pi=0;
    float a=200000;
    for(a=1;a<=200000;a=a+2)
    {
        if(c==0)
        {
            pi=pi+4/a;
            c++;
        }
        else
        {
            pi=pi-4/a;
            c--;
        }
    }
    cout<<"PI's Value = "<<pi<<endl;
    return 0;
}
```



```
double calc_pi(int nrDigitals)
{
    double a = 1;
    double b = 1 / sqrt(2);
    double t = (double)1 / 4;
    double p = 1;

    double temp_a;
    double temp_b;

    for(int i = 0; i < nrDigitals; ++i)
    {
        temp_a = (a + b) / 2;
        temp_b = sqrt(a * b);
        t = t - p * pow((a - temp_a), 2);
        p = 2 * p;

        a = temp_a;
        b = temp_b;
    }

    double pi = pow((a + b), 2) / (4 * t);
    return pi;
}
```

实验步骤

用来计算  $\pi$  的公式很多，但是基本上都是计算近似值，有精度限制，提高精度需要提高数据量，对于本实验要求在确定数值中寻找字符情况不是很适合，因此我们可以使用别人已经造好的轮子，用先辈们已经计算出的数据来完成我们的实验。

2. 准备  $\pi$  数值文件

Pi\_billion.txt 属性

常规安全详细信息以前的版本

Pi\_billion.txt

文件类型:

文本文档 (.txt)

打开方式:

记事本

更改(C)...

位置:

C:\Users\ASUS\Documents\DataStructure\laborato

大小:

953 MB (1,000,000,002 字节)

占用空间:

953 MB (1,000,001,536 字节)

创建时间:

2022年4月18日, 16:04:06

修改时间:

2016年4月8日, 23:49:08

访问时间:

2022年4月22日, 10:33:18

属性:

☐ 只读(R)

☐ 隐藏(H)

高级(D)...

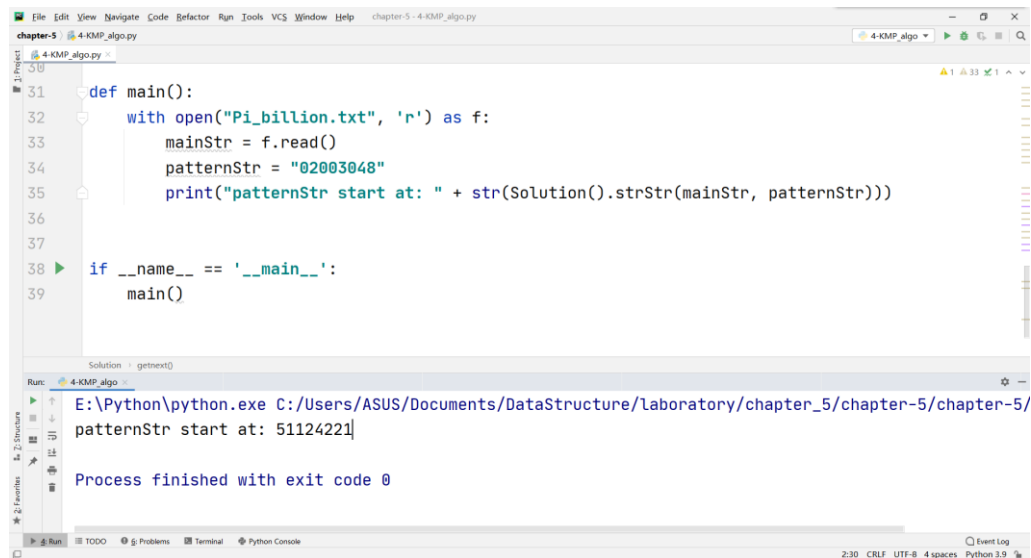
确定

取消

应用(A)

picture 1 十亿位的  $\pi$  数值文件

考虑到可能存在数据溢出的问题(KMP 算法的 cpp 实现中数组长度类型为 int)，我们可以先使用 Python 作为工具提前检测一下所会用到的数据规模



```
def main():
    with open("Pi_billion.txt", 'r') as f:
        mainStr = f.read()
        patternStr = "02003048"
        print("patternStr start at: " + str(Solution().strStr(mainStr, patternStr)))

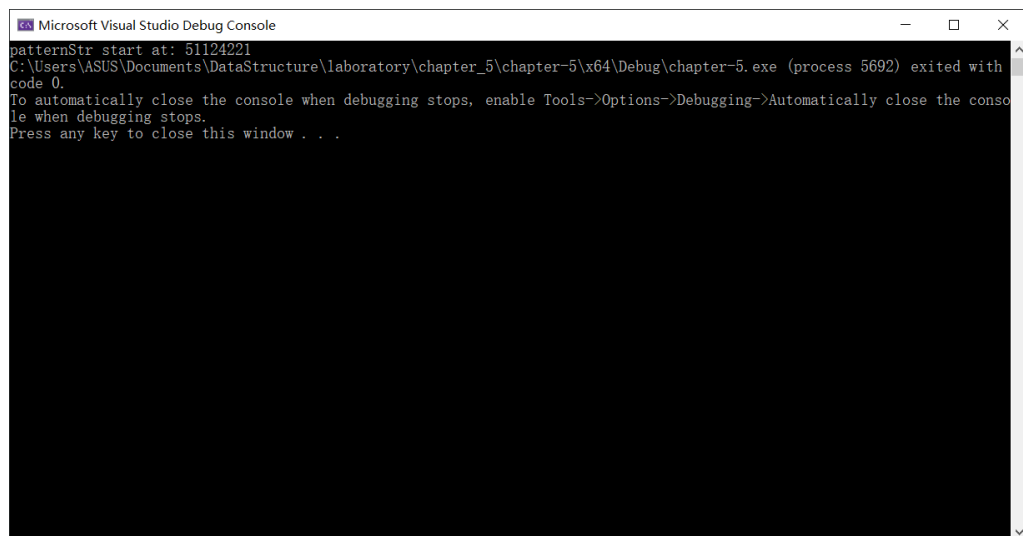
if __name__ == '__main__':
    main()
```

Run: E:\Python\python.exe C:/Users/ASUS/Documents/DataStructure/Laboratory/chapter\_5/chapter-5/chapter-5/patternStr start at: 51124221

Process finished with exit code 0

picture 2 Python 实现 KMP

可以看到我们的课程号(02003048)出现在 Pi 中第 51124221 处位置，因此确定不会溢出，在 C++中使用数据规模为 1 亿的 Pi 值即可

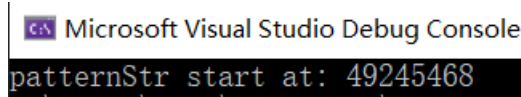


```
patternStr start at: 51124221
C:\Users\ASUS\Documents\DataStructure\laboratory\chapter_5\chapter-5\x64\Debug\chapter-5.exe (process 5692) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .
```

picture 3 在 C++中找到相同结果


### 3. 寻找其他可能的数值位置

- 8 位生日 ID (20011012)




```
patternStr start at: 49245468
```

- 国庆日 (1001)

 Microsoft Visual Studio Debug Console


```
patternStr start at: 15762
```

- 莫名其妙的数值（114514）

 Microsoft Visual Studio Debug Console

```
patternStr start at: 132112
```

- 长者生日（19260817）

 Microsoft Visual Studio Debug Console

```
patternStr start at: 69943589
```

#### 4. 实验中用到的 KMP 算法源代码

- C++

```

#pragma once
#include <iostream>
#include <vector>
using namespace std;

vector<int> getNext(string pattern)
{
    int j = 0, m = pattern.length();
    vector<int> next(m);
    int t = -1;
    next[0] = -1;
    while (j < m - 1)
    {
        if (t < 0 || pattern[t] == pattern[j])
        {
            t++;
            j++;
            next[j] = t;
        }
        else
        {
            t = next[t];
        }
    }
    return next;
}

int kmp_algo(string mainString, string pattern)
{
    int n = mainString.length();
    int m = pattern.length();
    vector<int> next = getNext(pattern);
    int i = 0, j = 0;
    while (i < n && j < m)
    {
        if (j < 0 || mainString[i] == pattern[j])
        {
            i++;
            j++;
            if (j == m)
                return i - j;
        }
        else
        {
            j = next[j];
        }
    }
    return -1;
}

```



➤ Python

```
class Solution:
    def strStr(self, haystack: str, needle: str) ->
    int:
        a=len(needle)
        b=len(haystack)
        if a==0:
            return 0
        next=self.getNext(a,needle)
        p=-1
        for j in range(b):
            while p>=0 and needle[p+1]!=haystack[j]:
                p=next[p]
            if needle[p+1]==haystack[j]:
                p+=1
            if p==a-1:
                return j-a+1
        return -1

    def getNext(self,a,needle):
        next=['' for i in range(a)]
        k=-1
        next[0]=k
        for i in range(1,len(needle)):
            while (k>-1 and needle[k+1]!=needle[i]):
                k=next[k]
            if needle[k+1]==needle[i]:
                k+=1
            next[i]=k
        return next
```

实验总结

本次实验非常有趣，在理解 KMP 算法的基础上完成了一个有趣的小程序，体会到了字符串搜索算法的实际用处，加深了对算法的理解与印象，并通过实际调用感受到其妙处。