编译原理与技术

LL(1)语法分析实验

目录

一．实验内容与要求 1

二． 程序设计与实现 1

1． 主要步骤 1

2． 主要算法 1

3． 程序实现 2

4． 样例结果 4

5． 实验总结 6

6． 源码附件 7

2017-11-20

[裴子祥 计科七班 学号 2015211921]

[指导老师：刘辰]

一．实验内容与要求

题目：语法分析程序的设计与实现。

实验内容：编写语法分析程序，实现对算术表达式的语法分析。产生式如下：

E->E+T | E-T | T

T->T*F | T/F | F

F-> id | (E) | num

实验要求：在对输入的算数表达式进行分析的过程中，依次输出所采用的产生式。编写 LL(1)语法分析程序：

1. 编程实现算法 4.2，为给定文法自动构造预测分析表。
2. 编程实现算法 4.1，构造 LL(1)预测分析程序。并进行错误处理。

实验环境：

MICSOFT WINDOWS 10

Visual Studio 2015

C/C++作为实现语言

二． 程序设计与实现

1． 主要步骤

(1) 消除文法的左递归
(2) 提取文法的左公因子
(3) 构造文法的 FIRST 集
(4) 构造文法的 FOLLOW 集
(5) 构造预测分析表
(6) 构造预测分析程序

2． 主要算法

算法 4.2----------->构造预测分析表

输入：文法 G

输出：文法 G 的预测分析表 M

方法：

```
for (文法 G 的每个产生式 A->α) {
    for (每个终结符号 a∈FIRST(α))
      把 A->α放入 M[A,a]中;
    if (ε∈FIRST(α)
    for (任何 b∈OLLOW(A))
        把 A->阿尔法放入 M[A,b]中;
} ;
for (所有无定义的 M[A,a])  标上错误标志;
```

算法 4.1----------->构造预测分析程序
输入：输入符号串 ， 文法 G 的一张预测分析表 M。
输出：若 在 L(G)中，则输出 的最左推导，否则报告错误。
方法：分析开始时，$在栈底，文法开始符号 S 在栈顶， $在输入缓冲区中

```
置 ip 指向 $ 的第一个符号；
do {
  令 X 是栈顶符号，a 是 ip 所指向的符号；
  if (X 是终结符号或$) {
      if (X==a) {
          从栈顶弹出 X; ip 前移一个位置;
      };
      else error();
  else /* X 是非终结符号 */
  if (M[X,a]=X->Y1Y2…Yk ) {
     从栈顶弹出 X;
     把 Yk、Yk-1、…、Y2、Y1 压入栈，Y1 在栈顶;
      输出产生式 X Y1Y2…Yk ;
  };
  else error();
}while(X!=$) /* 栈不空，继续 */
```

3． 程序实现

(1) 输入

待分析表达式字符串
如*180*62*+3
10+(1+2)*3+(22/8)

(2) 输出文件形式部分举例

对符号串的分析过程

(3) 文法改写

消除左递归：
E->TA
A->+TA | -TA | ε
T->FB
B->*FB | /FB | ε
F->(E) | n

提取左公因式
没有左公因式可以提取

(4) FIRST 集与 FOLLOW 集

	FIRST	FOLLOW
E	(n	$)
A	+-ε	$)
T	(n	$+-)
B	*/ε	$+-)
F	(n	$*/)+-

(5) 全局变量

```
const int VnSize = 5;                          //非终结符个数
const int VtSize = 8;                          //终结符个数
const int ProductionSize = 10;                 //产生式个数

char Vn[VnSize] = { 'E','A','T','B','F' };     //非终结符
char Vt[VtSize] = { 'n','+','-','*','/','(',')','$' };//终结符
vector<char> FIRST[VnSize];                    //FIRST集
vector<char> FOLLOW[VtSize];                       //FOLLOW集
stack<char> AnalyseStack;                          //分析栈
string AnalyseTable[VnSize][VtSize];               //预测分析表
string Production[ProductionSize];                 //文法产生式
string inputBuffer;                            //输入缓冲区
int ip;                                        //输入缓冲区指针
```

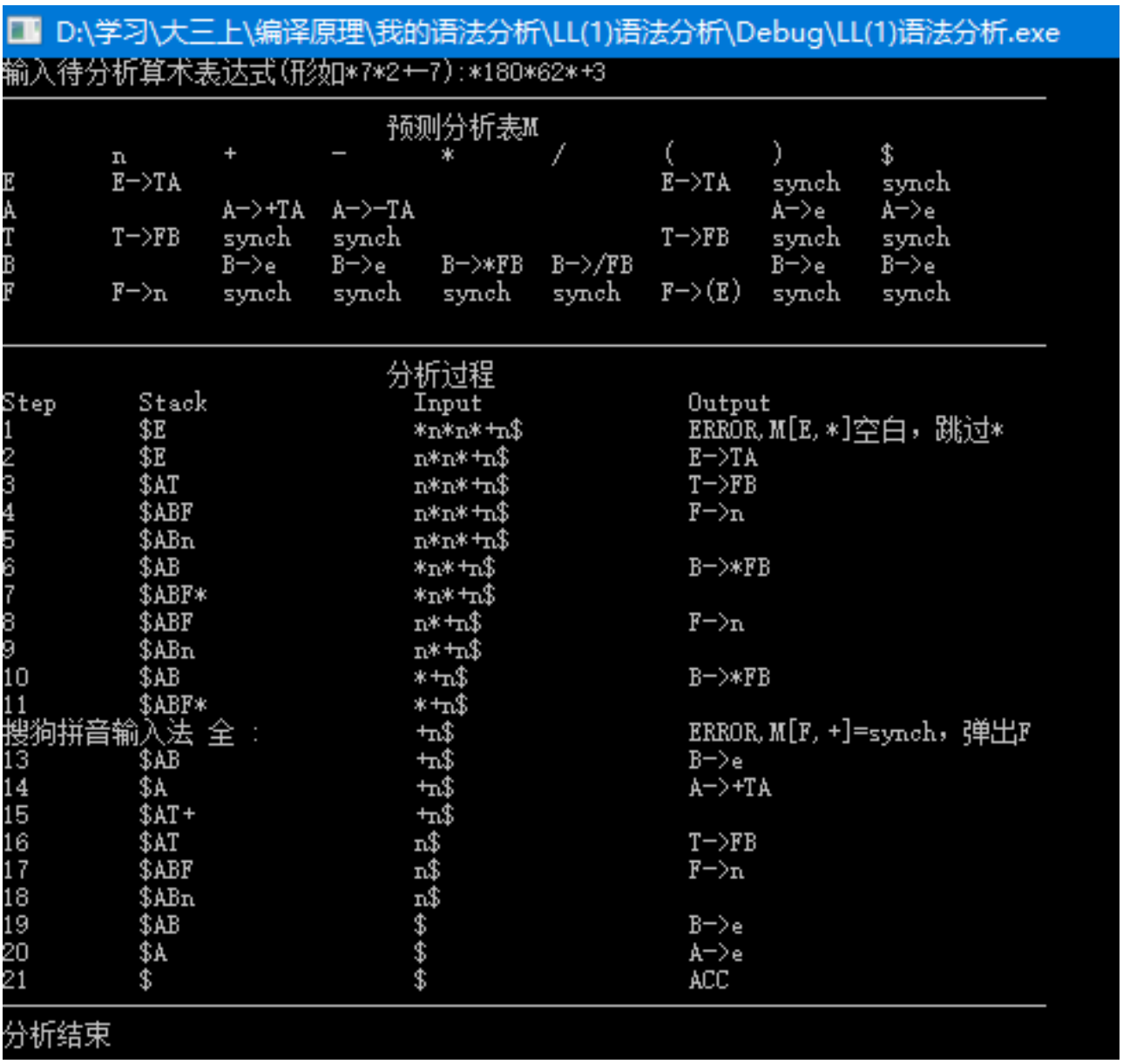
(6) 函数与过程

```
int mapVn(char c);                             //非终结符与数字对应

int mapVt(char c);                             //终结符与数字对应

void Initial();                                //产生式、FIRST、FOLLOW初始化

void createAnalyseTable();                         //创建预测分析表，算法4.2

string outStack();                             //输出栈内容字符串

void error1();                                 //第一种错误，栈顶终结符与当前输入符不匹配
void error2(char X, char a);                       //第二种错误，表项空白
void error3(char X, char a);                       //第三种错误，表项为"synch"

void LL1_analyse(string w);                    //算法4.1,非递归预测分析方法

string inStringChange(string w);                   //输入数字转化为非终结符'n'
```

```
void TablePrint();                          //输出预测分析表
```

4． 样例结果

（1） 示例输入 1，*180*62*+3



结果分析

通过示例 1，能够看出错误处理能够识别出三种错误：

一种是 X 为终结符却不和当前输入符号 a 匹配，此时弹出栈顶终结符

第二种是预测分析表项为空，此时跳过该输入符号

第三种是预测分析表项为同步信息 synch,此时弹出栈顶非终结符。

该程序能够很好的还原书本上的预测分析步骤，结果也是正确的。

（2） 示例输入 2，10+(1+2)*3+(22/8)

D:\学习\大三上\编译原理\我的语法分析\LL(1)语法分析\Debug\LL(1)语法分析.exe

```
输入待分析算术表达式(形如*7*2+-7)：10+(1+2)*3+(22/8)

                        预测分析表M
        n       +       -       *       /       (       )       $
E       E->TA                                   E->TA   synch   synch
A               A->+TA  A->-TA                          A->e    A->e
T       T->FB   synch   synch                   T->FB   synch   synch
B               B->e    B->e    B->*FB  B->/FB          B->e    B->e
F       F->n    synch   synch   synch   synch   F->(E)  synch   synch

                        分析过程
Step    Stack           Input           Output
1       $E              n+(n+n)*n+(n/n)$        E->TA
2       $AT             n+(n+n)*n+(n/n)$        T->FB
3       $ABF            n+(n+n)*n+(n/n)$        F->n
4       $ABn            n+(n+n)*n+(n/n)$
5       $AB             +(n+n)*n+(n/n)$         B->e
6       $A              +(n+n)*n+(n/n)$         A->+TA
7       $AT+            +(n+n)*n+(n/n)$
8       $AT             (n+n)*n+(n/n)$          T->FB
9       $ABF            (n+n)*n+(n/n)$          F->(E)
10      $AB)E(          (n+n)*n+(n/n)$
11      $AB)E           n+n)*n+(n/n)$           E->TA
12      $AB)AT          n+n)*n+(n/n)$           T->FB
13      $AB)ABF         n+n)*n+(n/n)$           F->n
14      $AB)ABn         n+n)*n+(n/n)$
15      $AB)AB          +n)*n+(n/n)$            B->e
16      $AB)A           +n)*n+(n/n)$            A->+TA
17      $AB)AT+         +n)*n+(n/n)$
18      $AB)AT          n)*n+(n/n)$             T->FB
19      $AB)ABF         n)*n+(n/n)$             F->n
20      $AB)ABn         n)*n+(n/n)$
21      $AB)AB          )*n+(n/n)$              B->e
22      $AB)A           )*n+(n/n)$              A->e
23      $AB)            )*n+(n/n)$
24      $AB             *n+(n/n)$               B->*FB
25      $ABF*           *n+(n/n)$
26      $ABF            n+(n/n)$                F->n
27      $ABn            n+(n/n)$
28      $AB             +(n/n)$                 B->e
29      $A              +(n/n)$                 A->+TA
30      $AT+            +(n/n)$
31      $AT             (n/n)$                  T->FB
32      $ABF            (n/n)$                  F->(E)
33      $AB)E(          (n/n)$
34      $AB)E           n/n)$                   E->TA
35      $AB)AT          n/n)$                   T->FB
36      $AB)ABF         n/n)$                   F->n
37      $AB)ABn         n/n)$
38      $AB)AB          /n)$                    B->/FB
39      $AB)ABF/        /n)$
40      $AB)ABF         n)$                     F->n
41      $AB)ABn         n)$
42      $AB)AB          )$                      B->e
43      $AB)A           )$                      A->e
44      $AB)            )$
45      $AB             $                       B->e
46      $A              $                       A->e
47      $               $                       ACC

分析结束
```

5． 实验总结

这是编译原理与技术第二次程序设计实验，程序设计极大地参考并实现了课本上所给出的算法 4.1 与算法 4.2，还有输入输出形式，采用了较为规范的编程形式，使程序具有良好的可读性，在不断地改进与 debug 后能够实现所给出的算术表达式的分析，这是令人欢欣鼓舞的！！

但是程序也有不足之处，程序中的文法改写，如消除左递归，提取左公因式，得出 FIRST、FOLLOW 集，都是通过人力运算得出，若能实现程序求出，将会变得更为智能。这是能够改进的。

这次实验获益匪浅，加深了对语法分析 LL(1)文法的理解，也极大地激励了学习编译原理的热情，对课本内容的不断理解与探索，对 debug 的一丝不苟，都是对后面的学习将是极大的鼓励。

6． 源码附件

```
/**
*LL1_Praser.cpp 文件内容
*作者:裴子祥
*时间:2017.11.20
*/

#include<iostream>
#include<vector>
#include<string>
#include<stack>
#include<iomanip>

using namespace std;

const int VnSize = 5;                          //非终结符个数
const int VtSize = 8;                          //终结符个数
const int ProductionSize = 10;                 //产生式个数

char Vn[VnSize] = { 'E','A','T','B','F' };     //非终结符
char Vt[VtSize] = { 'n','+','-','*','/','(',')','$' };//终结符
vector<char> FIRST[VnSize];                        //FIRST集
vector<char> FOLLOW[VtSize];                   //FOLLOW集
stack<char> AnalyseStack;                      //分析栈
string AnalyseTable[VnSize][VtSize];           //预测分析表
string Production[ProductionSize];             //文法产生式
string inputBuffer;                                //输入缓冲区
int ip;                                            //输入缓冲区指针


int mapVn(char c);                                 //非终结符与数字对应

int mapVt(char c);                                 //终结符与数字对应

void Initial();                                    //产生式、FIRST、FOLLOW初始化

void createAnalyseTable();                     //创建预测分析表，算法4.2

string outStack();                                 //输出栈内容字符串

void error1();                                     //第一种错误，栈顶终结符与当前输入符不匹配
void error2(char X, char a);                   //第二种错误，表项空白
void error3(char X, char a);                   //第三种错误，表项为"synch"
```

```
void LL1_analyse(string w);                          //算法4.1,非递归预测分析方法

string inStringChange(string w);                  //输入数字转化为非终结符'n'

void TablePrint();                                   //输出预测分析表


int main()//主函数
{
    string s;
    cout << "输入待分析算术表达式(形如*7*2+-7):";
    cin >> s;
    string w = inStringChange(s);
    Initial();
    createAnalyseTable();
    TablePrint();
    LL1_analyse(w);

    system("pause");
    return 0;
}

int mapVn(char c)//非终结符与数字对应
{
    switch (c)
    {
    case 'E':
        return 0;
        break;
    case 'A':
        return 1;
        break;
    case 'T':
        return 2;
        break;
    case 'B':
        return 3;
        break;
    case 'F':
        return 4;
        break;
    default:
        return -1;
        break;
    }
```

```
}
int mapVt(char c)//终结符与数字对应
{
    switch (c)
    {
    case 'n':
        return 0;
        break;
    case '+':
        return 1;
        break;
    case '-':
        return 2;
        break;
    case '*':
        return 3;
        break;
    case '/':
        return 4;
        break;
    case '(':
        return 5;
        break;
    case ')':
        return 6;
        break;
    case '$':
        return 7;
        break;
    default:
        return -1;
        break;
    }
}


void Initial()
{
    //消除左递归、提取左公因式后的产生式
    Production[0] = "E->TA";
    Production[1] = "A->+TA";
    Production[2] = "A->-TA";
    Production[3] = "A->e";
    Production[4] = "T->FB";
    Production[5] = "B->*FB";
```

```
    Production[6] = "B->/FB";
    Production[7] = "B->e";
    Production[8] = "F->(E)";
    Production[9] = "F->n";

    //FIRST集
    FIRST[0].push_back('(');
    FIRST[0].push_back('n');
    FIRST[1].push_back('+');
    FIRST[1].push_back('-');
    FIRST[1].push_back('e');
    FIRST[2].push_back('(');
    FIRST[2].push_back('n');
    FIRST[3].push_back('*');
    FIRST[3].push_back('/');
    FIRST[3].push_back('e');
    FIRST[4].push_back('(');
    FIRST[4].push_back('n');

    //FOLLOW集
    FOLLOW[0].push_back('$');
    FOLLOW[0].push_back(')');
    FOLLOW[1].push_back('$');
    FOLLOW[1].push_back(')');
    FOLLOW[2].push_back('$');
    FOLLOW[2].push_back('+');
    FOLLOW[2].push_back('-');
    FOLLOW[2].push_back(')');
    FOLLOW[3].push_back('$');
    FOLLOW[3].push_back('+');
    FOLLOW[3].push_back('-');
    FOLLOW[3].push_back(')');
    FOLLOW[4].push_back('$');
    FOLLOW[4].push_back('+');
    FOLLOW[4].push_back('-');
    FOLLOW[4].push_back(')');
    FOLLOW[4].push_back('*');
    FOLLOW[4].push_back('/');

    for (int i = 0; i < VnSize; ++i)
    {
        for (int j = 0; j < VtSize; ++j)
        {
            AnalyseTable[i][j] = "";//预测分析表项置空
        }
```

```
    }
}

void createAnalyseTable()//创建预测分析表，算法4.2
{
    for (int i = 0; i < ProductionSize; ++i)
    {
        int row = -1, col = -1, row2 = -1;//分析表横纵坐标
        char RightHead = Production[i][3];
        row = mapVn(Production[i][0]);
        if (RightHead == 'n' || RightHead == '+' || RightHead == '-'
            || RightHead == '*' || RightHead == '/' || RightHead == '('
            || RightHead == ')' || RightHead == '$')//为终结符时
        {
            col = mapVt(RightHead);
            AnalyseTable[row][col] = Production[i];
        }
        else if (RightHead == 'E' || RightHead == 'A' || RightHead == 'T'
            || RightHead == 'B' || RightHead == 'F')//非终结符
        {
            int row2 = mapVn(RightHead);//非终结符
            for (int j = 0; j < FIRST[row2].size(); ++j)
            {
                if (FIRST[row2][j] != 'e')//不是A->ε
                {
                    col = mapVt(FIRST[row2][j]);
                    AnalyseTable[row][col] = Production[i];
                }
                else
                {
                    for (int k = 0; k < FOLLOW[row].size(); ++k)
                    {
                        col = mapVt(FOLLOW[row][k]);
                        AnalyseTable[row][col] = Production[i];
                    }
                }
            }
        }
        else//A->ε
        {
            for (int k = 0; k < FOLLOW[row].size(); ++k)
            {
                col = mapVt(FOLLOW[row][k]);
                AnalyseTable[row][col] = Production[i];
            }
```

```
        }
    }

    for (int i = 0; i < VnSize; ++i)//对非终结符遍历
    {
        int col = -1;
        for (int j = 0; j < FOLLOW[i].size(); ++j)
        {
            col = mapVt(FOLLOW[i][j]);//b∈FOLLOW(A)
            if (AnalyseTable[i][col].empty())
            {//若M[A,b]为空，填入"synch"
                AnalyseTable[i][col] = "synch";
            }
        }
    }

}

string outStack()//输出站内容
{
    string s = "";
    char ctemp;
    stack<char> stemp;
    while (!AnalyseStack.empty())
    {
        ctemp = AnalyseStack.top();
        AnalyseStack.pop();
        stemp.push(ctemp);
    }
    while (!stemp.empty())
    {
        ctemp = stemp.top();
        s += ctemp;
        AnalyseStack.push(ctemp);
        stemp.pop();
    }
    return s;
}


void error1()//第一种错误，栈顶终结符与当前输入符不匹配
{
    AnalyseStack.pop();//弹出栈顶
}
void error2(char X, char a)//第二种错误，表项空白
```

```
{
    ip++;//空白，错误，跳过a
    cout << "ERROR,M[" << X << ',' << a << "]空白，跳过" << a << endl;
}
void error3(char X, char a)//第三种错误，表项为"synch"
{
    AnalyseStack.pop();//弹出栈顶
    cout << "ERROR,M[" << X << ',' << a << "]=synch，弹出" << X << endl;
}


void LL1_analyse(string w)//算法4.1,非递归预测分析方法
{
    char X;//指向栈顶文法符号
    char a;//ip所指向的输入符号
    ip = 0;//w$指针，初始指向第一个符号
    int step = 0;//记录步骤
    inputBuffer = w;
    AnalyseStack.push('$');
    AnalyseStack.push('E');
    cout << "                              分析过程" << endl;
    cout << left << setw(10) << "Step" << left << setw(20) << "Stack" << left << setw(20) << "Input" << left
<< setw(20) << "Output" << endl;

    do
    {
        X = AnalyseStack.top();
        a = inputBuffer[ip];
        ++step;
        cout << left << setw(10) << step << left << setw(20) << outStack() << left << setw(20) <<
inputBuffer.substr(ip);

        if (X == 'n' || X == '+' || X == '-' || X == '*'
            || X == '/' || X == '(' || X == ')' || X == '$')//终结符或$
        {
            if (X == a)
            {
                AnalyseStack.pop();
                ip++;
                if (X == '$')
                {
                    cout << "ACC";
                    cout << endl;
                    cout << "--------------------------------------------------------------------------------"
<< endl;
```

```
                cout << "分析结束" << endl;
            }

            cout << endl;
        }
        else
        {
            error1();//第一种错误
        }
    }
    else//非终结符
    {
        int rowVn = mapVn(X);
        int colVt = mapVt(a);
        if (rowVn == -1 || colVt == -1)
        {
            cout << "输入含有错误文法符号" << endl;
            exit(0);
        }
        if (AnalyseTable[rowVn][colVt].empty())
        {
            error2(X, a);//第二种错误,空白
        }
        else if (AnalyseTable[rowVn][colVt] == "synch")
        {
            error3(X, a);//第三种错误synch
        }
        else//正常情况
        {
            if (AnalyseTable[rowVn][colVt][3] == 'e')
            {
                AnalyseStack.pop();
                cout << AnalyseTable[rowVn][colVt] << endl;
            }
            else
            {
                AnalyseStack.pop();
                for (int i = AnalyseTable[rowVn][colVt].length() - 1; i >= 3; --i)
                {
                    AnalyseStack.push(AnalyseTable[rowVn][colVt][i]);
                }
                cout << AnalyseTable[rowVn][colVt] << endl;
            }
        }
    }
```

```
    } while (X != '$');

}


string inStringChange(string w)
{
    string temp = "";
    for (int i = 0; i < w.length(); ++i)
    {
        //cout << w[i] << endl;
        if (w[i] >= '0'&&w[i] <= '9'&&i < w.length())
        {//是数字
            ++i;
            while (w[i] >= '0'&&w[i] <= '9'&&i < w.length())
            {
                ++i;
            }
            temp += 'n';
            --i;
        }
        else
        {
            temp += w[i];
        }
    }
    temp += '$';
    return temp;
}


void TablePrint()//输出预测分析表
{
    cout << "--------------------------------------------------------------------------------" << endl;
    cout << "                              预测分析表M" << endl;
    for (int j = 0; j < VtSize; j++) {
        cout << "\t" << Vt[j];
    }
    cout << endl;
    for (int i = 0; i < VnSize; i++) {
        cout << Vn[i] << "\t";
        for (int j = 0; j < VtSize; j++) {
            cout << AnalyseTable[i][j] << "\t";
        }
```

```
        cout << endl;
    }
    cout << endl;
    cout << "--------------------------------------------------------------------------------" << endl;
}
```