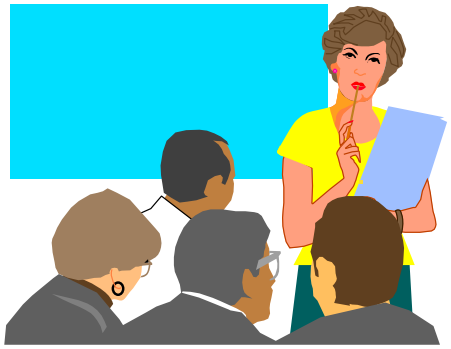
编译技术



胡春明
hucm@buaa.edu.cn

2019.9-2019.12

编译过程

编译过程是指将高级语言程序翻译为等价的目标程序的过程。

习惯上是将编译过程划分为5个基本阶段：

前两节课（符号表+运行时存储）：为代码生成做“需求分析”

上一节课（语法制导翻译）：为代码生成做“需求分析”

符号表
（编译时数据结构）

符号表

	name	kind	type	lev	other inf
1	x	var	real	0	
2	y	var	real	0	
3	i	var	int	0	
4	k	var	int	0	
5	name	var	array	0	
6	M_1	proc		0	
7	ind	para	int	1	
8	x	var	int	1	
9	P_2	proc		1	
10	j	para	real	2	
11	P_3	proc		2	
12	f	var	array	3	
13	test1	var	boolean	3	

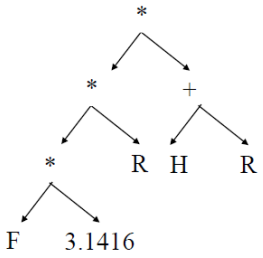
文法

E::= E+T | T
T ::= T*F | F
F ::= (E) | i

```
int a;
int b;
int c;
int d;

a = b + c + d;
b = a * a + b * b;
```

源程序

Parsing
语法制导翻译（SDT）



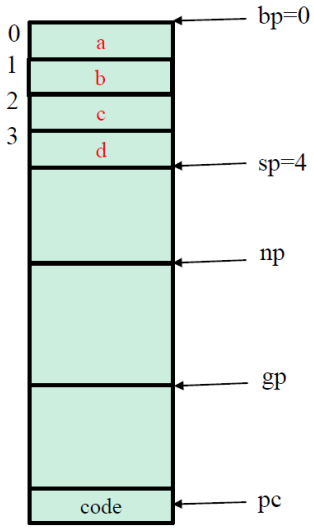
抽象语法树

```
LDA 1
LDA 2
ADI
LDA 3
ADI
STO 0
LDA 0
LDA 0
MULT
LDA 1
LDA 1
MULT
ADI
STO 1
```

中间代码

确保执行时符合编译器对运行组织的构想

编译器
对中间语言抽象机及运行时存储组织的设想



第十章 语义分析和代码生成

- **声明的处理** 符号表操作
- **表达式的处理** 变量的引用/运行时语义检查
- **赋值语句的处理** 写内存指令
- **控制语句的处理** 控制流处理（标号/比较/跳转）
- **过程调用和返回** 参数传递、运行栈操作

过程调用与返回

10.7 过程调用和返回

10.7.1 参数传递的基本形式

1.传值（call by value) — 值调用

实现：

调用段（过程语句的目标程序段）：

计算实参值 => 操作数栈栈顶

被调用段（过程说明的目标程序段）：

从栈顶取得值 => 形参单元

过程体中对形参的处理：

对形参的访问等于对相应实参的访问

特点：

数据传递是单向的

如C语言，
Ada语言的in参数，
Pascal 的值参数。

2.传地址（call by reference）— 引用调用

实现：

调用段：

计算实参地址 => 操作数栈栈顶

被调用段：

从栈顶取得地址 => 形参单元

如：FORTRAN, Pascal 的变量形参。

过程体中对形参的处理：

通过对形参的间接访问来访问相应的实参

特点:

结果随时送回调用段

3. 传名（call by name）

又称“名字调用”。即把实参名字传给形参。这样在过程体中引用形参时，都相当于对当时实参变量的引用。

当实参变量为下标变量时，传名和传地址调用的效果可能会完全不同。

传名参数传递方式，实现比较复杂，其目标程序运行效率较低, 现已很少采用。

```
begin
   integer I;
   array A[1:10] integer;
   procedure P(x);
      integer x;
        begin
           ……
           I := I + 1;
           x := x+ 5;
           ……
        end;
   begin
      ……
      I := 1;
      P( A[I] );
      ……
   end;
end;
```

假定：A[1] = 1 A[2] = 2

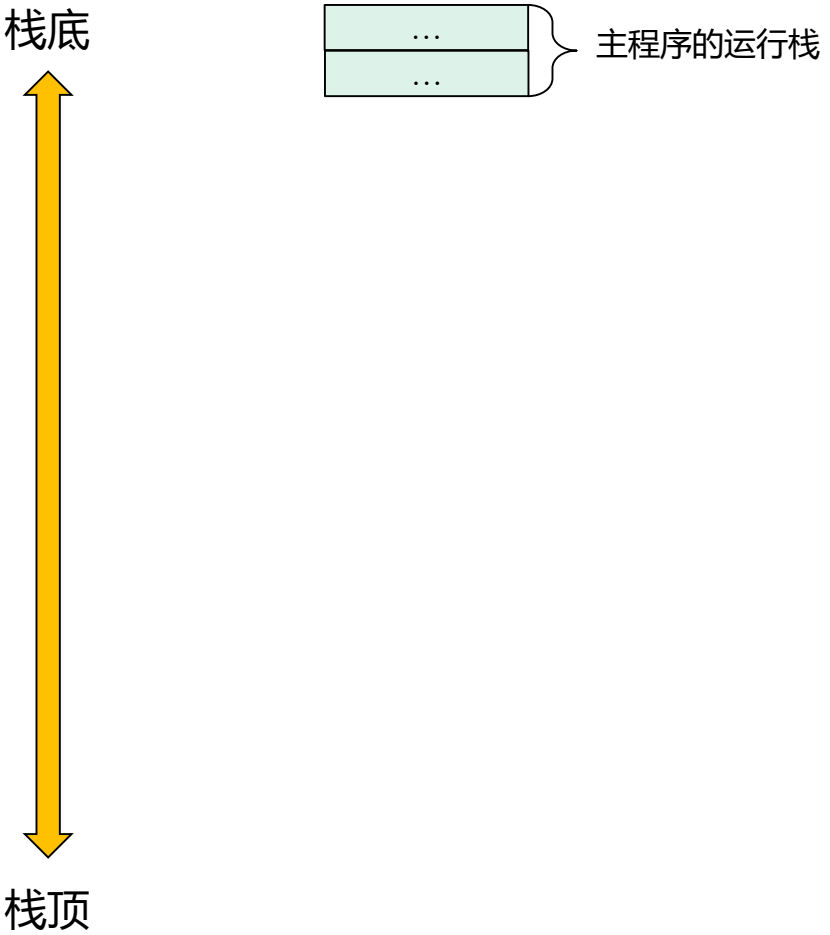
传值：	传地址：	传名：
I: 2 x: 6 A[1]: 1	I: 2 x: A[1] A[1]: 6	I: 2 x: =A[I] A[I]:= A[I]+5
A[1] = 1 A[2] = 2	A[1] = 6 A[2] = 2	A[1] = 1 A[2] = 7

函数调用的整个场景

主调用函数

```
process_symb(symb,
cursor,replacestr);
```

1. 检查该过程名是否已定义（过程名和函数名不能用错） **实参和形参在类型、顺序、个数上是否一致。(查符号表)**



函数调用的整个场景

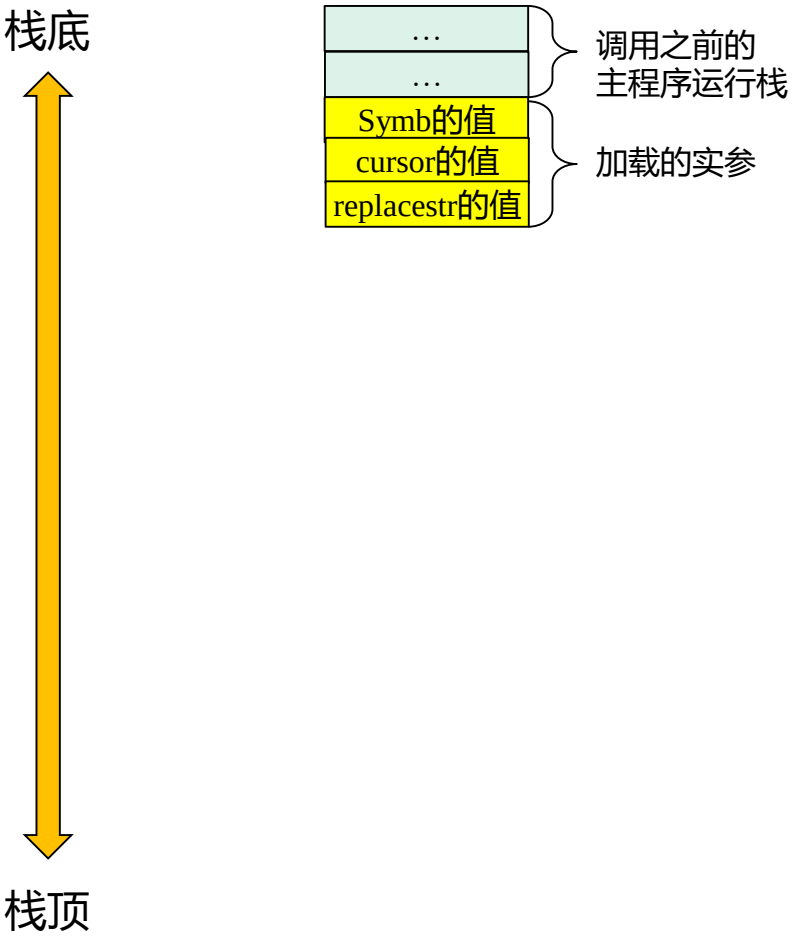
主调用函数

```
process_symb(symb,
cursor,replacestr);
```

1. **检查该过程名是否已定义**（过程名和函数名不能用错） **实参和形参在类型、顺序、个数上是否一致。(查符号表)**

2. **加载实参（值或地址）**

```
LOD, (addr of symb )
LOD, (addr of cursor )
LOD, (addr of replacestr)
```



函数调用的整个场景

主调用函数

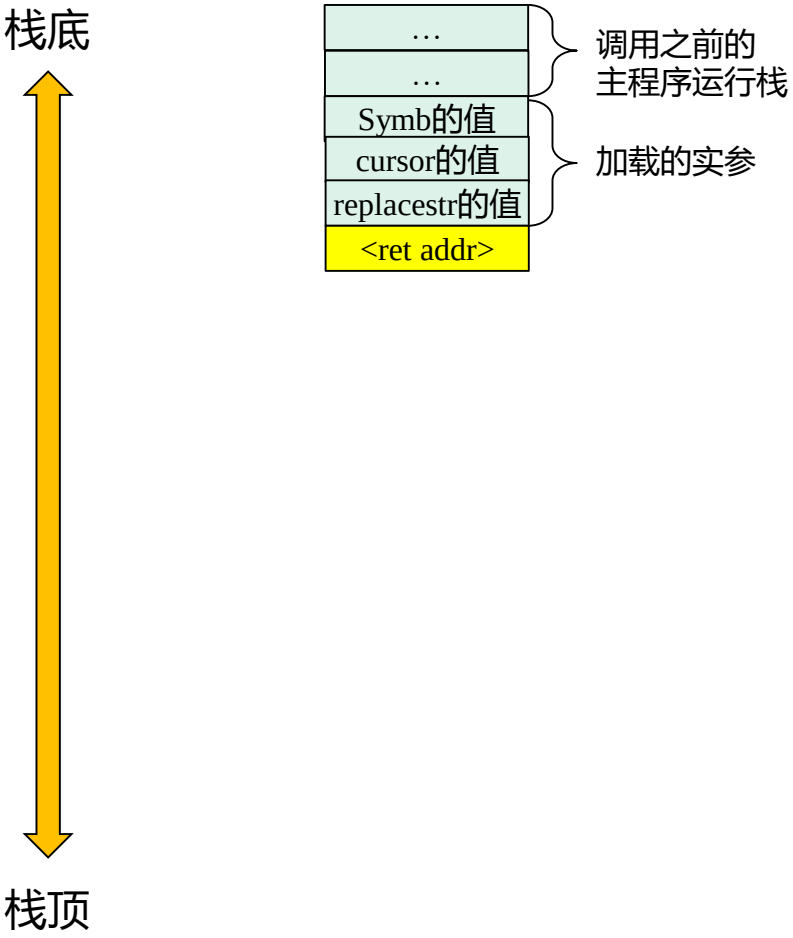
```
process_symb(symb,
cursor,replacestr);
```

1. **检查该过程名是否已定义**（过程名和函数名不能用错） **实参和形参在类型、顺序、个数上是否一致。(查符号表)**

2. **加载实参（值或地址）**

```
LOD, (addr of symb )
LOD, (addr of cursor )
LOD, (addr of replacestr)
```

3. **加载返回地址**



函数调用的整个场景

被调用函数

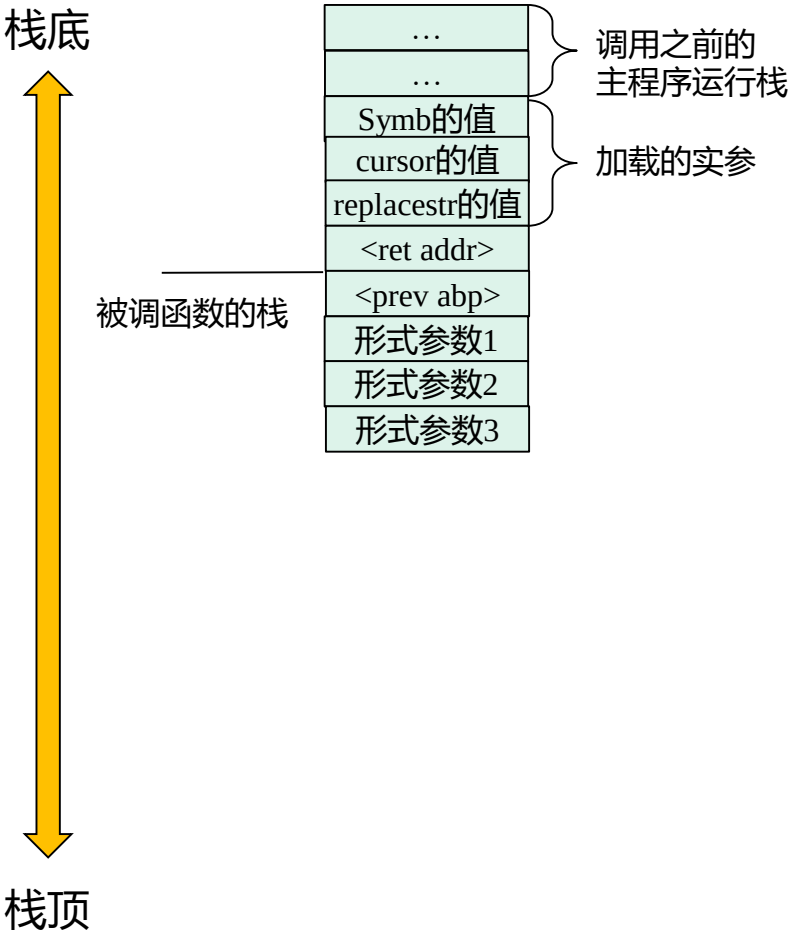
```
procedure process_symb
```

1. 准备活动记录、参数区

display区： n-1个指针，取决于当前层数n

隐式参数区： prev abp

显式参数区： 依赖于参数个数



10.7.2 过程调用处理

```
process_symb(symb, cursor,replacestr);
```

与调用有关的动作如下：

1. **检查该过程名是否已定义**（过程名和函数名不能用错）**实参和形参在类型、顺序、个数上是否一致。(查符号表)**

2. **加载实参（值或地址）**

3. **加载返回地址**

4. **转入过程体入口地址**

传值调用

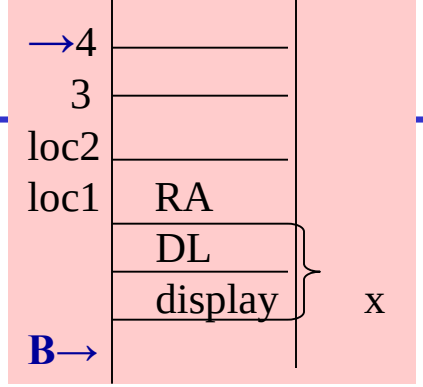
调用该过程生成的目标代码为：

```
 LOD, (addr of symb )
 LOD, (addr of cursor )
 LOD, (addr of replacestr)
 JSR, (addr of
process_symb)
<retaddr>:….
```

若实参并非上例中所示变量，而是表达式，则应生成相应计算实参表达式值的指令序列。

JSR指令先把返回地址压入操作数栈，然后转到被调过程入口地址。

```
procedure  process_symb
          (string:symbal, int: cur, string:
```



过程体头部应生成指令，存储返回地址和形参的值。

x: display+DL

```
ALC, 4 + x     /* x为定长项空间 */
STO, <actrec loc1>   /* 保存返回地址 */
STO, <actrec loc4>   /* 保存replacestr */
STO, <actrec loc3>   /* 保存cursor */
STO, <actrec loc2>   /* 保存symb */
```

过程调用时，实参加载指令是把实参变量内容（或地址）送入操作数栈顶，过程声明处理时，应先生成把操作数栈顶的实参送运行栈AR中形参单元的指令。

将操作数栈顶单元内容存入运行栈（动态存储分配的数据区）当前活动记录的形式参数单元。

可认为此时运行栈和操作数栈不是一个栈（分两个栈处理）

过程调用的ATG文法：

```
<proc call>→ <call head>↑i , z  @initm↑m<args> ↓i , z  @genjsr↓i
<call head>↑i , z → <id>↑n  @lookupproc↓n↑i, z
<args>↓i,z → @chklength↓i , z  | (<arg list>↓i, z)
<arg list>↓i , z → <expr>↑t  @chktype↓t, i, m, z↑z  <exprs>↓i , z
<exprs>↓i , z → @chklength↓i , z
```

@chklenth 应检验z最后值为0。否则表示实参数目小于形参数目。

@genjsr 生成JSR指令。该指令转移地址为 symbtbl [i] .addr

```
procedure  chktype(t, i, m, z);
    string t; integer m, i, z;
    if z < 1
    then begin
        error( '实参数大于形参数' , symtbl [i].name, statno);
        return ( z);
     end
     m := m+1;              /* 实参计数 */
     if  t ≠symtbl [i+m].type
      then error('实参和形参类型不匹配' , symtbl [i+m].name, statno);
      z := z-1;               /* 减去已匹配的形参数 */
      return (z);           /* 剩下待匹配的形参数 */
end;
```

过程调用的ATG文法：

```
<proc call>→ <call head>↑i , z  @initm↑m<args> ↓i , z  @genjsr↓i
<call head>↑i , z → <id>↑n  @lookupproc↓n↑i, z
<args>↓i,z → @chklength↓i , z   | (<arg list>↓i, z)
<arg list>↓i , z → <expr>↑t  @chktype↓t, i, m, z↑z   <exprs>↓i , z
<exprs>↓i , z → @chklength↓i , z
                              |   , <expr>↑t  @chktype↓t, i, m, z↑z
<exprs>↓i , z
```

```
procedure  lookupproc(n);
    string n; integer i, z;
    i := lookup(n);  /*查符号表*/
    if   i < 1
    then begin
          error( '过程' , n ,'未定义' , statno);
          errorrecovery( panic );   /*应急处理过程 */
          return ( i := 0, z:= 0);
         end
    else  return( i , z:= symtbl [i].dim);      /* z为形参数目*/
end;
```

形参个数计数，j初值为0

过程说明（定义）的ATG文法如下：

形参名填表

K := j ++

K := j

l := j

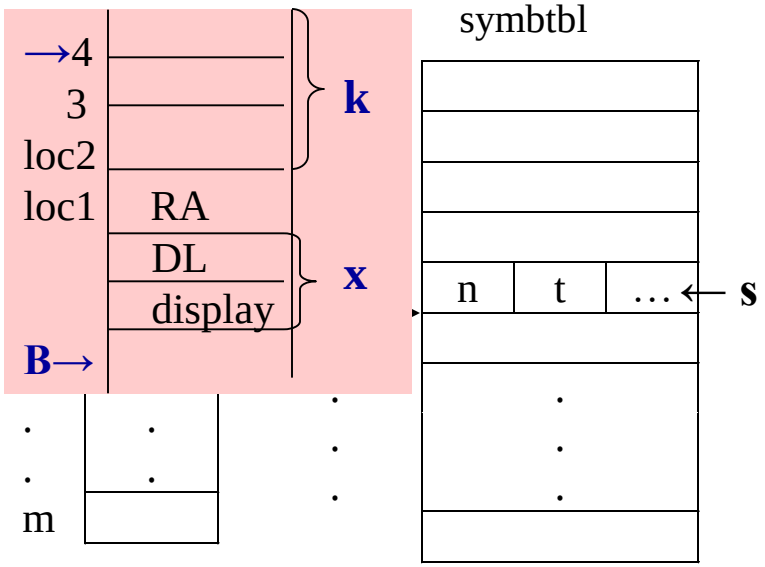
```
<proc defn>→<proc defn head>  @initcnt↑j
                              <parameters>↓j↑k  @emitstores↓l
<proc defn head>→procedure↑t <id>↑n @tblinsert↓t, n
<parameters>↓j↑k→@echo↓j↑k | (<parm list>↓j↑k)
<parm list>↓j↑l→<type>↑t : <id>↑n @tblinsert↓t, n

@upcnt↓j↑k <parms>↓j↑l
<parms>↓j↑l→@e↓j↑l | , <type>↑t : <id>↑n @tblinsert↓t, n

@upcnt↓j <parms>↓k↑l
```

@tblinsert 是把过程名和它的形参名填入符号表中：

```
procedure tblinsert( t, n );
  string t, n; integer hloc;
   if lookup ( n ) > 0
   then error( '名字定义重复' , statno);
   else begin
     hloc := hashfctn(n);
/*求散列函数值*/
   hashtbl[hloc]:= s;
    /*s为符号表指针（下标），为全局量*/
    symbtbl [s].name:= n;
    symbtbl [s].type:= t;
    s := s+1;
end;
```

最后得到的形参个数

```
procedure emitstores(k);
  integer  k;
  emitl( 'ALC', k + x +… );
  emitl( 'STO', < ll, x+1 >);
                        /*保存返回地址*/
  for i := k + x+1 down to x+2
                        /*保存参数值*/
       emitl( 'STO', < ll , i > )
  end;
end;
```




```
ALC, 4 + x    /* x为定长项空间 */
STO, <actrec loc1>   /* 保存返回地址 */
STO, <actrec loc4>   /* 保存replacestr */
STO, <actrec loc3>   /* 保存cursor */
STO, <actrec loc2>   /* 保存symb */
```

注：实际ALC指令所分配的空间应在所有局部变量定义处理完以后，并考虑固定空间（前述‘x’）大小，反填回去。

10.7.3 返回语句和过程体结束的处理

其语义动作有：

1)若为函数过程，应将操作数栈（或运行栈）顶的函数结果值送入（存回）函数值结果单元

2)生成无条件转移返回地址的指令（JMP RA)

3)产生删除运行栈中被调用过程活动记录的指令（只要根据DL—活动链，把abp退回去即可）

一个完整的例子

```
const a=45, b=27;
var x,y,g,m;

procedure swap;
  var temp;
  begin
    temp:=x;
    x:=y;
    y:=temp
  end;

procedure mod;
  x:=x-x/y*y;

begin
  x:=a;
  y:=b;
  call mod;
  while x<>0 do
    begin
      call swap;
      call mod;
    end;
  g:=y;
  m:=a*b/g;
  write(g,m)
end;
```

求最大公约数：gcd()，辗转相除法

```
const true=1, false=0;
var x,y,m,n,pf;

procedure prime;
  var i,f;
  procedure mod;
    x:=x-x/y*y;
  begin
    f:=true;
    i:=3;
    while i<m do
      begin
        x:=m;
        y:=i;
        call mod;
        if x=0 then f:=false;
        i:=i+2;
      end;
    if f=true then
      begin
        write(m);
        pf:=true
      end
  end;

begin
  pf:=false;
  read(n);
  while n>=2 do
    begin
      write(2);
      if n=2 then pf:=true;
      m:=3;
      while m<=n do
        begin
          call prime;
          m:=m+2;
        end;
      read(n)
    end;
  if pf=false then write(0);
end;
```

习题课

知识点1：逆波兰式、三元式、间接三元式、四元式（TAC）

表达式 `-a+b*(-c+d)` 的逆波兰式是_____。

A. `ab+-cd+-*`

B. `a-b+c-d+*`

C. `a-b+c-d*+`

D. `a-bc-d+*+`

知识点1：逆波兰式、三元式、间接三元式、四元式（TAC）

写出下列语句的逆波兰表示。

```
if (x+y)*Z=5 then x:=(a+b)^c else y:=a^b^c
```

知识点2：属性翻译文法

终结符具有_____属性。

A. 传递

B. 继承

C. 抽象

D. 综合

知识点2：属性翻译文法

知识点3：属性翻译文法的构造

谢谢！