定义编译器功能 & 汇编编程

计算机科学与技术专业　　1711436　　皮春莹

摘要

在这次实验中，总结 VC 支持的主要的 C++语言特性，在此基础上，定义了简化版 C++支持的基本语言特性，主要包括标识符、常量、变量声明、表达式、基本语句、函数定义等方面，并用上下文无关文法描述该语言子集。

针对预备作业一中的阶乘和斐波那契数列的 C++程序，分别编写等价的汇编程序，并用 MASM32 生成可执行程序，调试运行。

最后，对编译器程序的数据结构和算法设计进行了思考。

关键字： C++语言特性　上下文无关文法　汇编程序　阶乘　斐波那契数列

引言

编程语言都具备一组公共的语法特征，如标识符、常量、表达式、基本语句、函数定义等，不同的语言仅在特征的细节上有所区别。上下文无关文法是描述语言语法结构的一组形式规则。本文针对常用的 VC 编译器，总结它所支持的主要的 C++语言特性，定义了简化版 C++支持的基本语言特性，用上下文无关文法描述该语言子集。并编写阶乘和斐波那契数列的汇编程序，用 MASM32 生成可执行程序，调试运行。另外，初步了解了编译器的数据结构和算法设计。

实验内容

1. 定义编译器功能

1.1 VC 支持的主要的 C++语言特性

C++的语言特性主要包括：

(1) 支持的数据类型（int，float，char，bool，string 等）。

(2) 支持变量声明、赋值语句、复合语句、if/switch 分支语句，以及 while-do/for/do-while 循环。

(3) 支持函数、数组指针等。

(4) 支持算术运算（加减乘除按位与或等）、逻辑运算（逻辑与或等）、关系运算（不等等于大于小于等），下图是 C++的运算优先级[1]：

优先权	运算符	说明	结合性
1	()	括号	由左至右
2	!、-、++、--	逻辑运算符NOT、算术运算符负号、递增、递减	由右至左
3	*、/、%	算术运算符的乘法、除法、余数	由左至右
4	+、-	算术运算符加法、减法	由左至右
5	<<、>>、>>>	位操作子左移、右移、无符号右移	由左至右
6	>、>=、<、<=	关系运算符大于、大于等于、小于、小于等于	由左至右
7	==、!=	关系运算符等于、不等于	由左至右
8	&	位操作子AND	由左至右
9	^	位操作子XOR	由左至右
10	|	位操作子OR	由左至右
11	&&	逻辑运算符AND	由左至右
12	||	逻辑运算符OR	由左至右
13	?:	条件控制运算符	由右至左
14	=、op=	指定运算符	由右至左

图 1 C++的运算优先级

1.2 上下文无关文法

上下文无关文法[2]（context-free grammar，CFG）是一种用于描述程序设计语言语法的表示方式，一个上下文无关文法由四个元素组成：终结符号、非终结符号、一个开始符号和一组产生式。

(1) 终结符号是组成串的基本符号。

(2) 非终结符号是表示串的集合的语法变量。非终结符号表示的串集合用于定义由文法生成的语言。非终结符号给出了语言的层次结构，而这种层次结构是语法分析和翻译的关键。

(3) 在一个文法中，某个非终结符号被指定为开始符号，这个符号表示的串集合就是这个文法生成的语言。按照惯例，首先列出开始符号的产生式。

(4) 一个文法的产生式描述了将终结符号和非终结符号组合成串的方法。每个产生式由下列元素组成:

a) 一个被称为产生式头或左部的非终结符号。这个产生式定义了这个头所代表的串集合的一部分。

b) 符号→。

c) 一个由零个或多个终结符号与非终结符号组成的产生式体或右部。产生式体中的成分描述了产生式头上的非终结符号所对应的串的某种构造方法。

1.3 CGF 描述 C++主要语言特性

1.3.1 标识符

$$
\begin{aligned}
ndigit \quad &\rightarrow \quad \_ \text{ (下划线)} \\
&| \quad a \mid b \mid \ldots \mid z \text{ (所有小写字母)} \\
&| \quad A \mid B \mid \ldots \mid Z \text{ (所有大写字母)}
\end{aligned}
$$

$$
digit \quad \rightarrow \quad 0 \mid nozero\_digit
$$

$$
\begin{aligned}
id \quad &\rightarrow \quad ndigit \\
&| \quad id\ digit \\
&| \quad id\ ndigit
\end{aligned}
$$

其中，ndigit 表示字母和下划线，digit 表示数字，id 表示标识符，标识符由字母 a~z，A~Z 或下划线开头，后面跟任意（可为 0）个字母、下划线或数字。

1.3.2 常量

$$
nozero\_digit \quad \rightarrow \quad 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9
$$

$$
\begin{aligned}
const\_int &\rightarrow nozero\_digit \\
&\mid const\_int\ digit
\end{aligned}
$$

nozero_digit 表示非零数字，const_int 表示整型常量。

$$
\begin{aligned}
digit\_seq &\rightarrow digit \\
&\mid digit\_seq\ digit
\end{aligned}
$$

$$
\begin{aligned}
signed\_digit\_seq &\rightarrow digit\_seq \\
&\mid +digit\_seq \\
&\mid -digit\_seq
\end{aligned}
$$

$$
\begin{aligned}
const\_frac &\rightarrow . digit\_seq \\
&\mid digit\_seq\ . \\
&\mid digit\_seq\ .\ digit\_seq
\end{aligned}
$$

$$
\begin{aligned}
exponent &\rightarrow e\ signed\_digit\_seq \\
&\mid E\ signed\_digit\_seq
\end{aligned}
$$

$$
\begin{aligned}
const\_float &\rightarrow const\_frac \\
&\mid const\_frac\ exponent \\
&\mid digit\_seq\ exponent
\end{aligned}
$$

digit_seq 表示数字序列，数字序列的开头可以是 0，signed_digit_seq 是带符号的数字序列，const_frac 是小数部分，3.、.2、0.05 都是符合规范的小数，exponent 表示指数部分，指数部分由 e 或 E 开头，后面跟带符号的数字序列，const_float 表示浮点数常量。

$const\_char \rightarrow$ 除了单引号、双引号、反斜线之外的所有字符

$$
\begin{aligned}
str &= "\ str\ const\_char\ " \\
&= "\ const\_char\ "
\end{aligned}
$$

const_char 表示字符型常量，str 表示字符串常量。

$$
\begin{aligned}
constant &\rightarrow const\_int \\
&\mid const\_float \\
&\mid const\_char \\
&\mid str
\end{aligned}
$$

constant 表示常量，由整数型常量、浮点数常量、字符型常量和字符串常量构成。

1.3.3 变量声明

$$
\begin{aligned}
store &\rightarrow \boldsymbol{auto} \mid \boldsymbol{register} \mid \boldsymbol{static} \mid \boldsymbol{extern} \mid \varepsilon \\
type &\rightarrow \boldsymbol{int} \mid \boldsymbol{float} \mid \boldsymbol{char} \mid \boldsymbol{bool} \mid \boldsymbol{string} \\
idlist &\rightarrow id \mid idlist\ ,\ id \\
decl &\rightarrow store\ type\ idlist\ ;
\end{aligned}
$$

其中 id 表示标识符，type 表示变量类型，idlist 表示标识符列表，decl 表示变量声明语句。

1.3.4 常量声明

(1) const 关键字

$$decl \rightarrow \boldsymbol{const}\ type\ id = constant\ ;$$

(2) #define 预处理

$$decl \rightarrow \#\boldsymbol{define}\ id\ constant$$

其中 decl 表示常量声明语句，const 是关键字，id表示标识符，constant 是上面说明过的常量。

1.3.5 运算符

查找 C++的运算优先级以及结合性，从中选了常用的运算符，按照优先级从低到高的顺序，用上下文无关文法表达出来：

第 15 级，逗号表达式，用 expr 表示，结合性从左到右：

$$
\begin{aligned}
expr &\rightarrow \text{assign\_}expr \\
&\mid expr\ ,\text{assign\_}expr
\end{aligned}
$$

第 14 级，赋值与三元条件，assign_op 表示赋值相关的运算符，用 assign_expr 表示这一级的表达式，结合性从右到左：

$$
\begin{aligned}
\text{assign\_}op &\rightarrow = \mid +=\mid -= \\
&\mid *= \mid /= \mid \%= \\
&\mid <<= \mid >>= \mid \&= \\
&\mid \ \hat{}\ =
\end{aligned}
$$

$$
\begin{aligned}
\text{assign\_}expr &\rightarrow \text{logic\_or\_}expr \\
&\mid \text{logic\_or\_}expr\ \text{assign\_}op\ \text{assign\_}expr \\
&\mid \text{logic\_or\_}expr\ ?\ \text{logic\_or\_}expr\ :\ \text{logic\_or\_}expr
\end{aligned}
$$

第 13 级，逻辑或，用 logic_or_expr 表示，结合性从左到右：

$$\begin{aligned} \text{logic\_or\_}expr &\rightarrow \text{logic\_and\_}expr \\ &\mid \text{logic\_or\_}expr \;\&\&\; \text{logic\_and\_}expr \end{aligned}$$

第 12 级，逻辑与，用 logic_and_expr 表示，结合性从左到右：

$$\begin{aligned} \text{logic\_and\_}expr &\rightarrow \text{or\_}expr \\ &\mid \text{logic\_and\_}expr \;||\; \text{or\_}expr \end{aligned}$$

第 11 级，逐位或，用 or_expr 表示，结合性从左到右：

$$\begin{aligned} \text{or\_}expr &\rightarrow \text{xor\_}expr \\ &\mid \text{or\_}expr \;|\; \text{xor\_}expr \end{aligned}$$

第 10 级，逐位异或，用 xor_expr 表示，结合性从左到右：

$$\begin{aligned} \text{xor\_}expr &\rightarrow \text{and\_}expr \\ &\mid \text{xor\_}expr \;\hat{}\; \text{and\_}expr \end{aligned}$$

第 9 级，逐位与，用 and_expr 表示，结合性从左到右：

$$\begin{aligned} \text{and\_}expr &\rightarrow \text{equal\_}expr \\ &\mid \text{and\_}expr \;\&\; \text{equal\_}expr \end{aligned}$$

第 8 级，=和≠关系运算，用 equal_expr 表示，结合性从左到右：

$$\begin{aligned} \text{equal\_}expr &\rightarrow \text{relate\_}expr \\ &\mid \text{equal\_}expr \;==\; \text{relate\_}expr \\ &\mid \text{equal\_}expr \;!=\; \text{relate\_}expr \end{aligned}$$

第 7 级，＞，≥，＜，≤关系运算，用 relate_expr 表示，结合性从左到右：

$$\begin{aligned} \text{relate\_}expr &\rightarrow \text{shift\_}expr \\ &\mid \text{relate\_}expr \;>=\; \text{shift\_}expr \\ &\mid \text{relate\_}expr \;<=\; \text{shift\_}expr \\ &\mid \text{relate\_}expr \;>\; \text{shift\_}expr \\ &\mid \text{relate\_}expr \;<\; \text{shift\_}expr \end{aligned}$$

第 6 级，左移、右移，用 shift_expr 表示，结合性从左到右：

$$\begin{aligned} \text{shift\_}expr &\rightarrow add\_expr \\ &\mid \text{shift\_}expr \;\ll\; add\_expr \\ &\mid \text{shift\_}expr \;\gg\; add\_expr \end{aligned}$$

第 5 级，加减运算，用 add_expr 表示，结合性从左到右：

$$
\begin{aligned}
add\_expr &\rightarrow mul\_expr \\
&\mid add\_expr + mul\_expr \\
&\mid add\_expr - mul\_expr
\end{aligned}
$$

第 4 级，乘除取模运算，用 mul_expr 表示，结合性从左到右：

$$
\begin{aligned}
mul\_expr &\rightarrow unary\_expr \\
&\mid mul\_expr * unary\_expr \\
&\mid mul\_expr / unary\_expr \\
&\mid mul\_expr \,\%\, unary\_expr
\end{aligned}
$$

第 3 级。单目运算、前缀自增自减、强制类型转换、指针求值、间接取址，用 unary_expr 表示，结合性从右到左，unary_op 表示这一级中的运算符：

$$
\begin{aligned}
unary\_op &\rightarrow \; ++ \mid -- \mid + \\
&\mid \; -- \mid ! \mid \sim \\
&\mid \; (type) \mid * \mid \&
\end{aligned}
$$

$$
\begin{aligned}
\text{unary}\_expr &\rightarrow postfix\_expr \\
&\mid \text{unary}\_op \; \text{unary}\_expr
\end{aligned}
$$

第 2 级，后缀自增自减、**函数调用、数组下标、成员访问**，用 postfix_expr 表示，结合性从左到右：

$$
\begin{aligned}
args &\rightarrow expr \\
&\mid \epsilon
\end{aligned}
$$

$$
\begin{aligned}
postfix\_expr &\rightarrow primary\_expr \\
&\mid postfix\_expr ++ \\
&\mid postfix\_expr -- \\
&\mid postfix\_expr[expr] \\
&\mid postfix\_expr(args) \\
&\mid postfix\_expr.id \\
&\mid postfix\_expr \rightarrow id
\end{aligned}
$$

其中，postfix_expr[expr]表示数组下标访问，args 表示函数的参数列表，args 为空或者由一个逗号表达式 expr 组成，postfix_expr(args)表示函数调用，postfix_expr.id 是类的成员访问，postfix_expr→id 是指针成员访问。

第 1 级，基本表达式 primary_expr：

$$
\begin{aligned}
primary\_expr &\rightarrow id \\
&\mid constant \\
&\mid (expr)
\end{aligned}
$$

1.3.6 循环及分支语句

if 语句：

$$\begin{aligned} stmt &\rightarrow \boldsymbol{if}\ (expr)\ stmt\ \boldsymbol{else}\ stmt \\ &\mid \boldsymbol{if}\ (expr)\ stmt \end{aligned}$$

switch-case 语句（这里我不是很确定）：

$$\begin{aligned} single\_case &\rightarrow \boldsymbol{case}(constant): stmt \\ &\mid \boldsymbol{case}(constant): stmt\ \boldsymbol{break}; \end{aligned}$$

$$\begin{aligned} cases &\rightarrow single\_case \\ &\mid cases\ single\_case \\ &\mid \epsilon \end{aligned}$$

$$\begin{aligned} stmt &\rightarrow \boldsymbol{switch}(expr)\ cases\ \boldsymbol{default}: stmt \\ &\mid \boldsymbol{switch}(expr)\ cases \end{aligned}$$

for 循环：

$$\begin{aligned} optexpr &\rightarrow expr \\ &\mid \epsilon \end{aligned}$$

$$stmt \rightarrow \boldsymbol{for}\ (optexpr\ ;\ optexpr\ ;\ optexpr)\ stmt$$

while 语句和 do-while 语句：

$$stmt \rightarrow \boldsymbol{while}\ (expr)\ stmt$$

$$stmt \rightarrow \boldsymbol{do}\ stmt\ \boldsymbol{while}\ (expr)\ ;$$

其中，stmt 为语句，expr 为表达式。

1.3.7 函数

(1) 函数定义

$$\begin{aligned} funcdef &\rightarrow type\ funcname(paralist)\ stmt \\ paralist &\rightarrow paralist,\ paradef \mid paradef \mid \epsilon \\ paradef &\rightarrow type\ id \end{aligned}$$

(2) 函数调用

$$funcall \rightarrow funcname(paralist);$$

其中，paralist 表示参数列表，paradef 表示参数声明,funcname 表示函数名称，funcdef 表示函数声明语句，funcall 表示函数调用语句，id 是在上面已经表示过的标识符。

2. 汇编编程

2.1 阶乘

2.1.1 C++代码

```
void main(){
    int  n;
    int i = 2, f = 1;
    cout << "请输入阶乘最大值：";
    cin >> n;
    if(n > 1){
        while (i <= n){
        f = f * i;
        i = i + 1;
        }
    }
    cout <<"阶乘结果为："<< f << endl;
}
```

2.1.2 汇编代码

```
.486 ;使用 486 处理器
.model flat,stdcall
include \masm32\include\msvcrt.inc
includelib \masm32\lib\msvcrt.lib

.data
result dd 1;计算结果，dd 定义双字类型变量，占 4 个字节
tmp dd 2
inputdata dd ?;输入的阶乘的最大值
strtype db '%d',0;0 作为字符串结束的标记，编译软件没有自动加零的功能。
data1 db '请输入阶乘最大值：',0;db 的 b 是 byte 的缩写，占 1 个字节
data2 db '阶乘结果为：',0

.code
start:

invoke crt_printf,addr data1;打印字符串
```

```
18. invoke crt_scanf,addr strtype,addr inputdata;strtype 表示输入的格式，inputdata
    存储输入的值
19. push eax;保存现场
20. push ecx
21. mov eax,result
22. mov ecx,result
23.
24. cmp ecx,inputdata
25. jNB $2
26.
27. mov ecx,tmp
28. $1:
29. mul ecx;乘积存于 AX 中
30. inc ecx
31. cmp ecx,inputdata
32. jNG $1;不大于则跳转
33. $2:
34. mov result,eax
35. pop ecx;还原现场
36. pop eax
37.
38.
39. invoke crt_printf,addr data2
40. invoke crt_printf,addr strtype,result
41. INVOKE crt__getch
42. ret
43.
44. END start
```

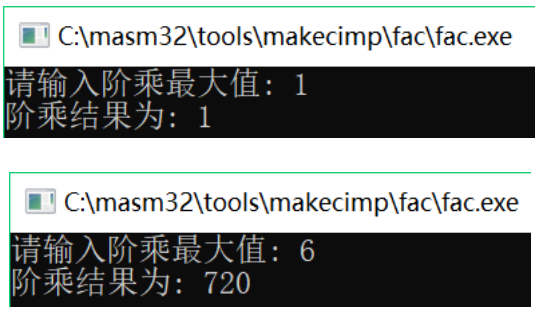

图 2 阶乘程序计算结果

2.2 斐波那契数列

2.2.1 C++代码

```
int fib(int i)
{
    if (i == 1 || i == 2)
        return 1;
    else
        return fib(i - 2) + fib(i - 1);
}
void main(){
    int  n;
    cout << "请输入斐波那契数列个数：";
    cin >> n;
    cout <<"阶乘结果为："<< fib(n) << endl;
}
```

2.2.2 汇编代码

```
.486 ;使用 486 处理器
.model flat,stdcall
include \masm32\include\msvcrt.inc
includelib \masm32\lib\msvcrt.lib

.data
result0 dd 0;存储最终结果的倒数第一个
result1 dd 1;存储最终结果的倒数第二个
input dd ?;输入的斐波那契数列的个数
type0 db '%d',0
data1 db '请输入斐波那契数列个数: ',0
data2 db '结果为: ',0
data3 db ' ',0

.code
start:
invoke crt_printf,addr data1
invoke crt_scanf,addr type0,addr input
invoke crt_printf,addr data2
mov ecx,1

$1:
mov eax,result0
```

```
24. mov ebx,result1
25. mov result0,ebx
26. add ebx,eax
27. mov result1,ebx
28.
29. inc ecx
30. cmp ecx,input
31. jL $1;小于则跳转
32.
33. invoke crt_printf,addr type0,result1
34.
35. INVOKE crt__getch
36. ret
37.
38. END start
```

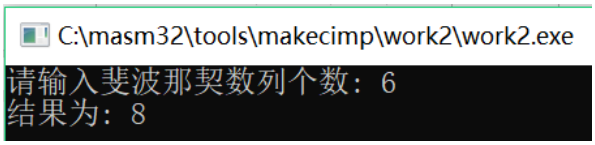

图 3 斐波那契数列计算结果

2.3 代码说明

借助 Masm32 包对 msvcrt 动态链接库的支持，调用 scanf、printf 库函数，分别实现了计算阶乘和斐波那契数列的功能，代码中已给出详细的注释，.486 表示使用 486 处理器，.model flat,stdcal 表示使用平坦内存模式和 stdcall 调用习惯，之后引用所需的头文件，.data 是变量定义部分，dd 表示双字变量，db 表示单个字节，最后是代码段，使用 inc 进行自增 1 操作，使用 cmp 对变量进行比较，使用 JL、JNG 等指令进行条件跳转。通过 Masm32 Editor 中 Project→Console Assemble & Link 完成汇编和链接，点击 Project→Run Program 执行 exe 文件。

最开始在写斐波那契数列的代码时，遇到了一些问题，最后查出原因是调用 invoke 的时候会改变寄存器 ecx 的值，所以在调用之前先 push，然后调用完之后 pop。

3. 编译器数据结构

如果要实现一个计算机程序（编译器）来将 C++程序转换为汇编程序，需要对程序的数据结构和算法进行精心的设计。这一节中我初步了解了编译器的主要数据结构及核心算法，在以后会有更深入的学习。

编译器的核心算法有：词法分析、语法分析、语义分析、代码生成、代码优化。

编译器的数据结构主要有[3]：

(1) 语法树：基于指针的数据结构，在进行分析时动态分配该结构，整棵树可作为一个指向根节点的单个变量保存。每一个节点都是一个记录，它的域表示由分析程序和之后的语义分析程序收集的信息。

(2) 符号表：与标识符有关（函数、变量、常量以及数据类型）。扫描程序、分析程序将标识符输入到表格中的语义分析程序；语义分析程序增加数据类型和其他信息；优化阶段和代码生成阶段也将利用由符号表提供的信息选出恰当的代码。因为对符号表的访问十分频繁，所以插入、删除和访问操作都必须比常规操作更有效，杂凑表是达到这一要求的标准数据结构。

(3) 常数表：存放程序中用到的常量和字符串，常数表需要快速插入和查找，但却无需删除，因为它的数据全程应用于程序而且常量或字符串在该表中只出现一次。代码生成器中也需要常数表来构造用于常数和在目标代码文件中输入数据定义的符号地址。

(4) 中间代码：根据中间代码的类型和优化的类型，该代码可以是文本串的数组、临时文本文件或是结构的连接列表。

结论

编译器要有能力翻译所有合法的 C++程序。但穷举所有 C++程序是不可能的，因此我们需要提炼语言特性，用符号化的方法进行翻译。

在这次实验中，总结了 VC 支持的主要的 C++语言特性，由于 C++语言的内容很多，我的总结并不够完善。定义简化版 C++支持的基本语言特性，主要包括标识符、常量、变量声明、表达式、基本语句、函数定义等，并用上下文无关文法描述该语言子集，部分描述不够简练，后面在编写编译器程序时需要继续做修改。

另外，分别编写阶乘和斐波那契数列的 C++程序的汇编程序，并用 MASM32 生成可执行程序，调试运行。这次编写的程序比较简单，主要是对跳转语句的使用，没有过多地考虑寄存器分配、数据溢出等问题，实际的汇编编程中会有更多要考虑的因素。

最后，初步了解编译器程序的数据结构和算法设计，方便后面的深入学习。

参考文献

[1] https://zh.cppreference.com/w/cpp/language/operator_precedence

[2] Alfred V.Aho 等.编译原理[M]. 北京:机械工业出版社, 2014. 114-115.

[3] https://www.cnblogs.com/x-police/p/10859240.html