南开大学 编译系统原理课程报告之预备工作二

C语言子集文法描述 & 汇编语言编程练习

学号：1811463 姓名：赵梓杰 专业：信息安全 年级：2018级

【摘要】 本篇报告延续上一篇报告对于编译器的研究，但是由于个人时间问题，没有和上一篇一样采用Latex进行编写，而是采用了Typora的Markdown形式进行编写，主要介绍了GCC/G++编译器所支持的C/C++语言特性，并且以C++语言为例，列举其通用文法，以此为基础，使用上下文无关文法，描述出自己设计的简单C语言编译器所支持的部分语子集。

【关键字】 GCC编译器，C++语言，上下文无关文法

1. GCC支持的C语言标准
    1. C语言标准

        原始的ANSI C标准于1989年批准，1990年出版，该标准于1990年下半年被批准为ISO标准。如果要在GCC中使用这个标准的话，可以使用-ansi -std = C90。

        C标准的第四版，被称为C11，于2011年发布为ISO/IEC 9899：2011。在开发过程中，此标准版本的草案被称为C1X。GCC已经完全支持这个标准，即支持-std = C11或者-std = iso9899：2011。

        默认情况下，GCC为C语言提供了相当多的一些扩展，但是这些扩展只有在极少的情况下才会和C标准发生冲突，如果这些扩展与我们选定的C标准的某个版本发生冲突，使用-std 列出扩展名就可以禁用这些扩展。

    2. C++语言标准

        GCC支持1998年发布的原始ISO C++标准以及2011和2014版本，原始ISO C++标准作为ISO标准出版，并由2003年出版的技术勘误进行了修订，这些标准分别被称为C++ 98和C++ 03。

        GCC实现了大多数C++ 98和C++ 03中的扩展，要在GCC中选择此标准，可以使用-ansi，-std = C++ 98 / C++ 03，要获得标准所需要的所有诊断信息，可以指定-pedantic。

        修订后的ISO C++标准于2011年作为ISO / IEC 14882：2011发布，被称为C++ 11，在发布之前通常被称为C++ 0x。C++ 11包含对C++语言的一些更改，如果需要选择此标准，类似C++ 98，改成C++ 11即可。

2. C语言的通用特性
    1. 关键字

        C语言保留关键字

char	short	int	unsigned	long	float	double	struct
union	void	enum	signed	const	volatile	typedef	auto
register	static	extern	break	case	continue	default	do
else	for	goto	if	return	switch	while	sizeof

        C99新增关键字

_Bool	_Complex	_Imaginary
inline	restrict	

C11新增关键字

_Alignas	_Alignof	_Atomic
_Generic	_Noreturn	_Static_assert

2. 数据类型

    1. 基础数据类型

    C语言在标准头文件limits.h和float.h中说明了基础数据类型的长度，这些数据范围也曾在IEEE 754标准中被提及。

关键字	位长 (字节)	取值范围	格式化字符串
char	1 bytes	-128..127 或 0..255	%c
unsigned char	1 bytes	0..255	%c, %hhu
signed char	1 bytes	-128..127	%c, %hhd, %hhi
int	2 bytes 或 4 bytes	-32768..32767 或 -2147483648..2147483647	%i, %d
unsigned int	2 bytes 或 4 bytes	0..65535 或 0..4294967295	%u
signed int	2 bytes 或 4 bytes	-32768..32767 或 -2147483648..2147483 647	%i, %d
short int	2 bytes	-32768..32767	%hi, % hd
unsigned short	2 bytes	0..65535	%hu
signed short	2 bytes	-32768..32767	%hi, % hd
long int	4 bytes 或 8 bytes	-2147483648..2147483647 或 -9223372036854775808 .. 9223372036854775807	%li, % ld
unsigned long	4 bytes 或 8 bytes	0..4294967295 或 0..18446744073709551615	%lu
signed long	4 bytes 或 8 bytes	-2147483648..2147483647 或 -9223372036854775808 .. 9223372036854775807	%li, % ld
long long	8 bytes	9223372036854775808 .. 9223372036854775807	%lli, %lld
unsigned long long	8 bytes	0..18446744073709551615	%llu
float	4 bytes	2.939E- 38..3.403E38	%f, %e , %g
double	8 bytes	5.563E- 309..1.798E308	%lf, % e, %g
long double	10bytes 或 16bytes	7.065E-9865..1.415E9864	%Lf, % Le, %Lg

    2. 结构数据类型

    结构数据类型允许编程人员（我）构造多个由基础数据类型组合而形成的复杂数据结构，struct是一个关键字，表明结构类型定义的开始，结构类型的说明符可以是基础类型的各种数据类型，例如下面：

```
struct student{
    long long num = 1811463 ; //学号
    char name[10] = "赵梓杰" ;//姓名
    int age = 20; //年龄
    char class[20] = "编译原理" ;//代表课程
    float score = 60; //代表分数
}
```

在C中struct和class可能还有一定程度的区分，但是在CPP中，struct几乎和class完全一致没有什么区别了。

3. 数组

   数组的话一般是变量名字后面跟着中括号，a[100]类似的声明就是数组声明，字符串其实也是一个数组，字符串默认以ASCII的串尾符作为数组的结束，最开始学习C++的时候，总是会忽视数组的索引值是从0开始算起的。

4. 指针

   指针作为大一的时候让我头皮发麻，当场自闭的一大难点，现在看看还是感觉很难，指针的定义就是在变量声明的时候使用*号，那么这个变量就是指针型变量，简而言之，就是这个变量存储了一个地址，而*并非是乘法中的乘号，而是单目运算符*，为取内容操作符，意思是取这个内存地址中所存储的内容。

5. 字符串

   除了char型数字，C标准库中还包含了用于对字符串进行操作的函数，使得他们看起来就像是我们人类所理解的字符串而不是数组，使用这些函数需要引用头文件string.h，而此时的数据类型被定义为String类型。

3. 基本运算符号

运算符	名称
() 、[] 、-> 、. 、++、--	后缀运算符
++ 、- 、* 、& 、 、! 、+ 、- 、sizeof、(cast)	单目运算符
* 、/ 、%、==	算术运算符
+ 、-	算术运算符
« 、»	位运算符
< 、<= 、> 、>=	关系运算符
== 、!=	关系运算符号
&	位与
^	位异或
|	位或
&&	逻辑与
||	逻辑或
? :	条件运算符
= 、+= 、-= 、*= 、/= 、%= 、&= 、|= 、 ^=	赋值运算符
,	顺序运算符

4. 逻辑结构语句

   1. 条件语句

      C语言有两种形式的条件语句，分别是：if分支和switch分支语句。

```
//if语句格式
if (boolean-expression){
    statement(s);
}else{
    statement(s);
}
//switch语句格式
switch(expression){
```

```
    case expression1:
        statement(s);
    case expression2:
        statement(s);
    ...
    default:
        statement(s);
}
```

2. 循环语句

   C语言有三种形式的循环语句：while循环、do-while循环和for循环。

```
//while语句格式
while(condition){
    statement(s);
}
//do-while语句格式
do{
    statement(s);
}while(condition);
//for语句格式
for(init;condition;increament){
    statement(s);
}
```

3. 跳转语句

   C语言中支持四种跳转（其中goto跳转现在不常用）：goto跳转、continue跳转、break跳转和return跳转

5. 函数

   函数是一组一起执行一个任务的语句，每个C++都至少有main()函数这一个函数，所有简单的程序都可以定义其他额外的函数，函数可以帮助我们实现代码的划分，通过一些特定的任务对代码进行区分和识别，同时C++标准库中提供了大量的程序可以调用的内置函数比如memcpy()等方便我们去使用。

6. 基本输入输出

   C++标准库提供了相当多相当多的输入输出功能，在大一下C++的最后一章节用一个章节取描述C++的输入输出，我在本文中只简单介绍最基本的I/O操作。

   C++的I/O发生在流中，流是一个字节序列，如果字节流是从设备流向内存，这叫做输入操作，如果字节流是从内存流向设备，叫做输出操作。

   预定义的对象cout是iostream类的一个实例，结合流插入运算符<<结合使用，输出到显示屏。

   预定义的对象cin是iostream类的一个实例，结合流提取运算符>>使用，cin对象附属到输入设备键盘。

3. C语言子集上下文无关文法的描述及其应用

   1. 标识符

      标识符只能是由数字、字母和下划线组成的字符串，并且第一个字符必须为字母和下划线不能为数字

      identifier -> non-figure-idchar | identifier non-figure-idchar | identifier figure

      figure -> 0|1|2|3|4|5|6|7|8|9

      non_figure-idchar -> _ | a-z | A-Z

2. 常量

   常量包含数值型常量和和字符串常量

   constant -> numeric-constant | character-constant

   数值型常量包括整数型常量与浮点型常量

   numeric-constant -> integer-constant | floating-constant

   整数型常量只支持十进制的整数，可以识别出先导0并且认为0无效，即0003识别为3

   integer-constant -> figure | integer-constant figure

3. 赋值表达式

   一个赋值表达式一般主要由三个部分组成：

   - 左值。左值的要求是终结符，即作为左值的变量有唯一一个确定的地址，从而可以将赋值表达式右值将值赋给这个地址上的变量。
   - 等号。一个赋值表达式，只能有一个等号，一般在循环语句中会默认判断flag=来描述隐式的左值和等号，通过布尔型变量flag决定分支和循环的进行。
   - 右值。右值可以是由终结符号、非终结符号及经过各种逻辑运算的组合构成，只要这种组合满足逻辑运算符的定义规范，将右值进行逻辑运算最终会得到一个特定的值，那么这个值会被赋值给表达式的左值。

4. 上下文无关文法的产生式描述

   根据上面的一些描述和C语言的一些规则可以得到以下的产生式。

```
type -> int | char | float | double | bool | ....
l-value -> id | id[num] | *(pointer)
r-value  num | id | &id | r-value + r-value |
           r-value - r-value  | r-value * r-value |
           r-value / r-value | r-value%integer |
           r-value== r-value| r-value< r-value |
           r-value> r-value | r-value<= r-value |
           r-value>= r-value | r-value!= r-value |
           function
assign-expr  l-value=r-value; | assign-expr \n assign-expr |
              id++; | id--; |if(r-value){assign-expr} |
              if(r-value){assign-expr}else{assign-expr} |
              if(r-value){assign-expr}else if(r-value){assign-
              expr} |
              for(type id=r-value ;r-value ;assign-expr)
              {assign-expr} | for(;;){assign-expr}
              while(r-value){assign-expr}
              type array_id={content1,content2,......} |
              char array_id="content" |
                 return r-value;
function  type func_name(type para1,type para2,.....){
               assign-expr
            }
structdef  struct new_type{type id1;type id2;......}
uniondef  union new_type{type id1;type id2;......}
enumdef  enum new_type{id1,id2,id3......} |
           enum new_type{id1=num ,id2,id3......} |
           enum new_type{id1=num1 ,id2=num2 ,id3=num3 ......}
```

4. 汇编语言Intel X86

   1. 斐波那契数列程序的汇编语言实现

```
    .bss
    .align  4
n:
    .zero   4

    .align  4
a:
    .zero   4

    .align  4
b:
    .zero   4

    .section    .rodata
STR0:
    .string "%d"
STR1:
    .string "The result is %d\n"
STR2:
    .string "Too small!\n" # < 1
STR3:
    .string "Too big!\n"  # > 46

    .text
    .globl  main
    .type   main,@function
main:
start:
    pushl   $n
    pushl   $STR0
    call    scanf
    addl    $8,%esp
    movl    $0,a
    movl    $1,b
    movl    $1,%eax
    movl    n,%ebx
    cmpl    %eax,%ebx
    jl      L3
    movl    $46,%eax
    movl    n,%ebx
    cmpl    %ebx,%eax
    jl      L4
    jmp     L1
L0:
    movl    b,%eax
    movl    a,%ebx
    addl    %ebx,%eax
    movl    %eax,b
    movl    b,%eax
    movl    a,%ebx
    subl    %ebx,%eax
    movl    %eax,a
    movl    n,%eax
    subl    $1,%eax
    movl    %eax,n
    jmp     L1
L1:
    movl    $0,%eax
```

```
    movl    n,%ebx
    subl    $1,%ebx
    cmpl    %ebx,%eax
    jl      L0
L2:
    pushl   b
    pushl   $STR1
    call    printf
    addl    $8,%esp
    movl    $0,%eax
    ret
L3:
    pushl   $STR2
    call    printf
    addl    $4,%esp
    jmp     start
L4:
    pushl   $STR3
    call    printf
    addl    $4,%esp
    jmp     start
    .section    .note.GNU-stack,"",@progbits
```

```
bcyx@LAPTOP-RNG7N73P:/mnt/f/Data/3-1/3-1/bianyi/homework/2/lab2$ make fib
gcc fib.s -m32 -o fib.out
qemu-i386 fib.out
0
Too small!
100
Too big!
40
The result is 102334155
bcyx@LAPTOP-RNG7N73P:/mnt/f/Data/3-1/3-1/bianyi/homework/2/lab2$
```

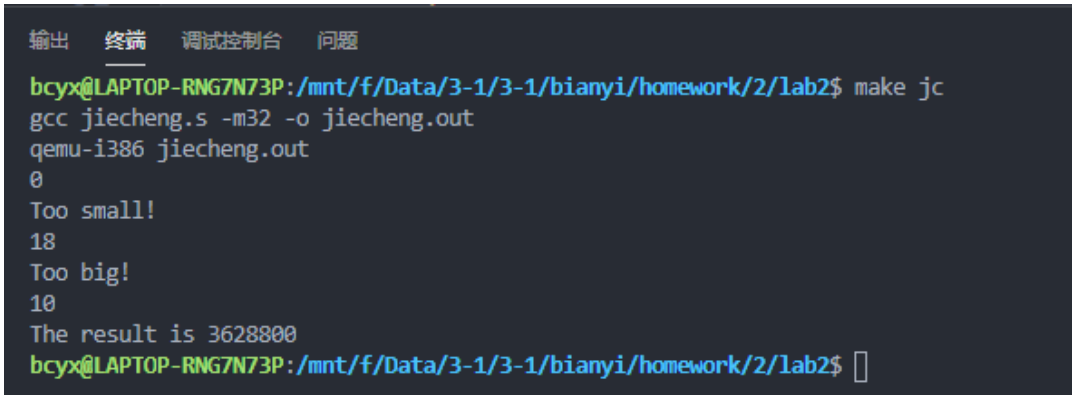
2. 阶乘程序的汇编语言实现

```
    .bss
    .align  4
n:
    .zero   4
t:
    .zero   4

    .section    .rodata
STR0:
    .string "%d"
STR1:
    .string "The result is %d\n"
STR2:
    .string "Too small!\n" # < 1
STR3:
    .string "Too big!\n"  # > 12
    .text
    .globl  main
    .type   main,@function
main:
start:
```

```
 pushl   $n
    pushl   $STR0
    call    scanf
    addl    $8,%esp
    movl    $1,t
    movl    $1,%eax
    movl    n,%ebx
    cmpl    %eax,%ebx
    jl      L3
    movl    $12,%eax
    movl    n,%ebx
    cmpl    %ebx,%eax
    jl      L4
    jmp     L1
L0:
    movl    t,%eax
    movl    n,%ebx
    imull    %ebx,%eax
    movl    %eax,t
    movl    n,%eax
    subl    $1,%eax
    movl    %eax,n
    jmp     L1
L1:
    movl    $1,%eax
    cmpl    n,%eax
    jl      L0
L2:
    pushl   t
    pushl   $STR1
    call    printf
    addl    $8,%esp
    movl    $0,%eax
    ret
L3:
    pushl   $STR2
    call    printf
    addl    $4,%esp
    jmp     start
L4:
    pushl   $STR3
    call    printf
    addl    $4,%esp
    jmp     start

    .section    .note.GNU-stack,"",@progbits
```



5. 思考

   如果不是我们去实现手写“编译”，而是借助一个计算机程序即编译器来实现将C程序转换为汇编程序，应该如何做?

   正向老师在题目中所提到的一样，这个编译器不能只会编译一个源程序，也不能只编译简单的源程序，而应该翻译所有合法的C程序，那么显然，穷举法是不可取的。

   在之前我们了解到了整个编译过程会分为词法分析、语法分析、语义分析、中间代码生成、代码优化和代码生成这几个阶段，而我们整个编译过程的重点我个人认为就在于词法分析和语法分析阶段，词法分析中我们需要把字符流分解成若干个记号，每个记号记录了符号的类型和数值，生成token字，而语法分析主要就是构造一个抽象的语法结构，也就是我们之前所学的抽象语法树结构。

   简单来讲，就是词法分析需要生成token，token就是对合法化实例进行分类表示所得到的结果，语法阶段需要构造出语法分析树，而语法分析树的范式化的被解析方式，让编译器能实现自动编译。

6. 参考文献

   GCC 官方文档 Language Standards Supported by GCC——https://gcc.gnu.org/onlinedocs/gcc-8.2.0/gcc/Standards.html

   C++ 基本的输入输出——http://www.runoob.com/cplusplus/cpp-basic-input-output.html

   C++ 数据类型——http://www.runoob.com/cplusplus/cpp-data-types.html

   C++ 常量——http://www.runoob.com/cplusplus/cpp-constants-literals.html

   C++ 标准库——http://www.runoob.com/cplusplus/cpp-standard-library.html

   王爽. 汇编语言（第三版）. 清华大学出版社. 2013

7. 总结

   这次报告的主要难度在于汇编代码的编写，因为之前没怎么用过printf和scanf的缘故，所以开始错误的以为printf也需要&，因此在汇编的时候一直使用取值，导致能生成可执行但是中间会覆盖edx为0，所以导致地址为空，出现段溢出(借助IDA Pro Debug发现的错误)，也算是给自己提个醒，也确实也感慨高级语言的优势，汇编代码多写了一个$就全都错了，也不会报很细节的错。