

编译原理

2025-Fall

主讲人：薛博桓

邮箱：bxue@m.scnu.edu.cn

办公室：学院综合楼 712

答疑时间：周一上午1-2节课



- 这是我们本儿即可用它来写编译器的编程语言。
- Cool是一种用于课堂教学的面向对象语言(Classroom Object Orientated Language)
- Cool的独特设计要求：他的编译器必须在较短时间内编写完成(我们只有一个学期)
- Cool必须易于实现
- 实际上世界上Cool编译器比Cool的项目多的多.....(可能世界上cool项目只有几十几百个..)

Cool

- 相对比Cool编写陈鼓型，让编译器更容易编写更重要！
- 所以Cool语言有一些特殊之处，都是为了方便实现而故意设计的
- 基本上Cool拥有现代语言的一切，包括抽象、静态类型、继承、重用以及自动内存管理等
- 我们的目的是把Cool语言编译为MIPS汇编语言
- MIPS是指令集，适用于20世纪80年代设计的电脑(我们的电脑可以用模拟器)

Cool

- 我们拥有5个作业
 - 写一个Cool程序
 - 写词法分析
 - 写语法分析
 - 写语义分析
 - 写代码生成
-
- 可以自由对上述模块进行组合，这样你的代码凉了，可以方便debug

- 在学术领域，尤其是在学习编译原理的学生和教师中，COOL语言有相当的知名度。许多大学的编译器课程都曾使用它作为教学项目。然而，在工业界和商业软件开发中，COOL语言并非一种常见或众所周知的语言。它的主要设计目的是教学，通过在一个学期内实现一个完整的编译器，让学生掌握编译器构造的理论和实践。
- COOL语言包含了现代编程语言的许多核心特性，例如：
 - 面向对象：支持类、对象和继承
 - 静态类型：在编译时进行类型检查，保证了程序的类型安全。
 - 自动内存管理：通过垃圾回收机制自动管理内存。
 - 表达式语言：大多数语言结构都是表达式，并且拥有返回值。
- 尽管具备这些特性，但为了简化编译器实现的任务，COOL语言也缺少了许多生产语言中的功能，比如数组、浮点数运算、接口和异常处理等。

- 下载iso文件(已经会的同学, 肯定不用了)
- <https://mirrors.aliyun.com/ubuntu-releases/22.04.5/ubuntu-22.04.5-desktop-amd64.iso>
- <https://www.virtualbox.org/wiki/Downloads> 或者是vmware(更推荐, 但是无所谓了)

- 一个可以工作的debian系系统最好，或是自己能解决环境问题任何unix都可以。docker也可以
- 以ubuntu20.04为例：执行以下内容
- 别忘了设置PATH

```
# enable programmable completion features (you don't need to
# this, if it's already enabled in /etc/bash.bashrc and /etc
# sources /etc/bash.bashrc).
if ! shopt -oq posix; then
  if [ -f /usr/share/bash-completion/bash_completion ]; then
    . /usr/share/bash-completion/bash_completion
  elif [ -f /etc/bash_completion ]; then
    . /etc/bash_completion
  fi
fi

export PATH=/usr/class/bin:$PATH
```

```
cool@cool: ~/workspace
cool@cool: ~/workspace

sudo apt-get install flex bison build-essential csh libxaw7-dev
sudo mkdir /usr/class
sudo chown $USER /usr/class
cd /usr/class put your student-dist.tar.gz here

tar -xf student-dist.tar.gz

dont forget to install this: sudo apt install libc6:i386
```

是用apt还是yum/dnf/snap自己根据自己操作系统决

让我们完成第一个程序

- 1

```
cool@cool: ~/workspace  
class Main{  
    main():Int { 1 };  
};  
~  
~
```

```
cool@cool:~/workspace$ coolc a.cl  
cool@cool:~/workspace$ ls  
a.cl  a.s  
cool@cool:~/workspace$ |
```




华南师范大学
SOUTH CHINA NORMAL UNIVERSITY

看看a.s的内容

- 都是汇编代码

```
cool@cool: ~/workspace
.data
.align 2
.globl class_nameTab
.globl Main_protObj
.globl Int_protObj
.globl String_protObj
.globl bool_const0
.globl bool_const1
.globl _int_tag
.globl _bool_tag
.globl _string_tag
_int_tag:
.word 2
_bool_tag:
.word 3
_string_tag:
.word 4
.globl _MemMgr_INITIALIZER
_MemMgr_INITIALIZER:
.word _NoGC_Init
.globl _MemMgr_COLLECTOR
_MemMgr_COLLECTOR:
.word _NoGC_Collect
.globl _MemMgr_TEST
_MemMgr_TEST:
.word 0
.word -1
str_const8:
.word 4
.word 5
.word String_dispTab
.word int_const1
.byte 0
.align 2
.word -1
str_const7:
.word 4
.word 6
.word String_dispTab
.word int_const2
```



执行汇编代码

- 就输出一些东西，但是没输出正经东西，因为我们没要求程序产生任何输出。

```
cool@cool:~/workspace$ spim a.s
SPIM Version 6.5 of January 4, 2003
Copyright 1990-2003 by James R. Larus (larus@cs.wisc.edu).
All Rights Reserved.
See the file README for a full copyright notice.
Loaded: ../lib/trap.handler
COOL program successfully executed
cool@cool:~/workspace$ |
```



测试更多的语法

```
1 class Main{
2     i : IO <- new Ip;
3     main():Int{
4         {
5             i.out_string("Hello World!") ;
6             1;
7         }
8     };
9 };
```

```
cool@cool:~/workspace$ coolc 2.cl
cool@cool:~/workspace$ spim 2.s
SPIM Version 6.5 of January 4, 2003
Copyright 1990-2003 by James R. Larus (larus@cs.wisc.edu).
All Rights Reserved.
See the file README for a full copyright notice.
Loaded: ../lib/trap.handler
Hello World!COOL program successfully executed
cool@cool:~/workspace$ ls
2.cl 2.s a.cl a.s
cool@cool:~/workspace$ |
```

- 这里的I可以堪称是一个field name类似于JAVA里的。在COOL里叫做属性。
- 类既可以包含属性或field name，也可以包括执行计算的方法

- 这样的话编译就报错了,
- 原因是IO类型不是Int类型

```
1 class Main{
2     i : IO <- new IO;
3     main():Int{
4         {
5             i.out_string("Hello World!\n") ;
6         }
7     };
8 };
```

```
cool@cool:~/workspace$ coolc 2.cl
2.cl:3: Inferred return type IO of method main does not conform to declared return type Int
Compilation halted due to static semantic errors.
cool@cool:~/workspace$ |
```

```
1 class Main{
2     i : IO <- new IO;
3     main():Int{
4         i.out_string("Hello World!\n")
5     };
6 };
```

```
cool@cool:~/workspace$ coolc 2.cl
2.cl:3: Inferred return type IO of method main does not conform to declared return type Int.
Compilation halted due to static semantic errors.
cool@cool:~/workspace$ |
```

```
1 class Main{
2     i : IO <- new IO;
3     main():IO{
4         i.out_string("Hello World!\n")
5     };
6 };
```

```
cool@cool:~/workspace$ coolc 2.cl
cool@cool:~/workspace$ spim 2.s
SPIM Version 6.5 of January 4, 2003
Copyright 1990-2003 by James R. Larus (larus@cs.wisc.edu).
All Rights Reserved.
See the file README for a full copyright notice.
Loaded: ../lib/trap.handler
Hello World!
COOL program successfully executed
cool@cool:~/workspace$ |
```

- 所有类型都属于Object

```
1 class Main{
2     i : IO <- new IO;
3     main():Object{
4         i.out_string("Hello World!\n")
5     };
6 };
```

```
cool@cool:~/workspace$ coolc 2.cl
cool@cool:~/workspace$ spim 2.s
SPIM Version 6.5 of January 4, 2003
Copyright 1990-2003 by James R. Larus (larus@cs.wisc.edu).
All Rights Reserved.
See the file README for a full copyright notice.
Loaded: ../lib/trap.handler
Hello World!
COOL program successfully executed
cool@cool:~/workspace$
```

- 似乎i并不是必要的

```
1 class Main{
2     main():Object{
3         (new IO).out_string("Hello World\n")
4     };
5 };
```

```
cool@cool:~/workspace$ coolc 2.cl
cool@cool:~/workspace$ spim 2.s
SPIM Version 6.5 of January 4, 2003
Copyright 1990-2003 by James R. Larus (larus@cs.wisc.edu).
All Rights Reserved.
See the file README for a full copyright notice.
Loaded: ../lib/trap.handler
Hello World
COOL program successfully executed
cool@cool:~/workspace$
```

- 继承的玩法（参考JAVA或者C++）
- Main继承了IO，就自己有IO的功能了
- Main继承了IO类的所有属性和方法(函数)
认是self自身

有的语言self叫做this(比如C/C++) 不写self就默

```
cool@cool: ~/workspace
1 class Main inherits IO{
2     main():Object{
3         self.out_string("Hello World\n")
4     };
5 };
cool@cool:~/workspace$ coolc 2.cl
cool@cool:~/workspace$ spim 2.s
SPIM Version 6.5 of January 4, 2003
Copyright 1990-2003 by James R. Larus (larus@cs.wisc.edu).
All Rights Reserved.
See the file README for a full copyright notice.
Loaded: ../lib/trap.handler
Hello World
COOL program successfully executed
cool@cool:~/workspace$
```

```
1 class Main inherits IO{
2     main():Object{
3         out_string("Hello World\n")
4     };
5 };
cool@cool:~/workspace$ coolc 2.cl
cool@cool:~/workspace$ spim 2.s
SPIM Version 6.5 of January 4, 2003
Copyright 1990-2003 by James R. Larus (larus@cs.wisc.edu).
All Rights Reserved.
See the file README for a full copyright notice.
Loaded: ../lib/trap.handler
Hello World
COOL program successfully executed
cool@cool:~/workspace$
```


更复杂的cool的例子——N!函数

```
cool@cool:~/workspace$ coolc 2.cl
cool@cool:~/workspace$ spim 2.s
SPIM Version 6.5 of January 4, 2003
Copyright 1990-2003 by James R. Larus (larus@cs.wisc.edu).
All Rights Reserved.
See the file README for a full copyright notice.
Loaded: ../lib/trap.handler
123
123
COOL program successfully executed
cool@cool:~/workspace$ spim 2.s
SPIM Version 6.5 of January 4, 2003
Copyright 1990-2003 by James R. Larus (larus@cs.wisc.edu).
All Rights Reserved.
See the file README for a full copyright notice.
Loaded: ../lib/trap.handler
3334
3334
COOL program successfully executed
cool@cool:~/workspace$ |
```

```
cool@cool: ~/workspace  cool@cool: ~/workspace
1 class Main {
2     main():Object{
3         (new IO).out_string((new IO).in_string().concat("\n"))
4     };
5 };
6 |
```

更复杂的cool的例子——N!函数

- 有一个现成的cool实现的，可以给字符串转化为整数的库

```
cool@cool: ~/workspace  x  cool@cool: ~/workspace  x  +  v
1 class Main inherits A2I{
2     main():Object{
3         (new IO).out_string(i2a((a2i((new IO).in_string()+1)).concat("\n")))
4     };
5 };
6
```

```
cool@cool:~/workspace$ coolc 2.cl atoi.cl
cool@cool:~/workspace$ spim 2.s
SPIM Version 6.5 of January 4, 2003
Copyright 1990-2003 by James R. Larus (larus@cs.wisc.edu)
All Rights Reserved.
See the file README for a full copyright notice.
Loaded: ../lib/trap.handler
884
885
COOL program successfully executed
cool@cool:~/workspace$ |
```



N!的完整例子

```
1 class Main inherits A2I{
2     main():Object{
3         (new IO).out_string(i2a(fact(a2i((new IO).in_string()))).concat("\n"))
4     };
5
6     fact(i : Int) : Int {
7         if (i = 0) then 1 else i * fact(i-1) fi
8     };
9 };
10
```

```
cool@cool:~/workspace$ spim 2.s
SPIM Version 6.5 of January 4, 2003
Copyright 1990-2003 by James R. Larus (larus@cs.wisc.edu).
All Rights Reserved.
See the file README for a full copyright notice.
Loaded: ../lib/trap.handler
3
6
COOL program successfully executed
cool@cool:~/workspace$ spim 2.s
SPIM Version 6.5 of January 4, 2003
Copyright 1990-2003 by James R. Larus (larus@cs.wisc.edu).
All Rights Reserved.
See the file README for a full copyright notice.
Loaded: ../lib/trap.handler
5
120
COOL program successfully executed
cool@cool:~/workspace$ |
```

```
cool@cool: ~/workspace
1 class Main inherits A2I{
2     main():Object{
3         (new IO).out_string(i2a(fact(a2i((new IO).in_string()))).concat("\n"))
4     };
5
6     fact(i : Int) : Int {
7         let fact: Int <- 1 in {
8             while (not (i=0)) loop
9                 {
10                     fact <- fact * i;
11                     i <- i - 1;
12                 }
13             pool;
14             fact;
15         };
16     };
17 };
18
~
```

```
cool@cool:~/workspace$ coolc 2.cl atoi.cl
cool@cool:~/workspace$ spim 2.s
SPIM Version 6.5 of January 4, 2003
Copyright 1990-2003 by James R. Larus (larus@cs.wisc.edu).
All Rights Reserved.
See the file README for a full copyright notice.
Loaded: ../lib/trap.handler
4
24
COOL program successfully executed
cool@cool:~/workspace$ |
```

- let 表达式是 Cool 语言中用于引入局部变量的核心语法。它与许多其他语言中的变量声明不同，因为它本身是一个 表达式，而不仅仅是一个 语句。

- let 的基本结构如下：

- let <id1> : <type1> <- <expr1>, <id2> : <type2> <- <expr2>, ... in <expr_body>

let: 关键字，表示开始一个局部变量绑定。

<id> : <type> <- <expr>: 这是一个变量绑定。

<id> 是变量名。

<type> 是变量的类型。

<- 是赋值符号。

<expr> 是用于初始化该变量的表达式。

初始化部分 <- <expr> 是可选的。如果省略，变量会被初始化为其类型的默认值（例如 Int 是 0，Bool 是 false，其他对象是 void）。

,: 如果需要在 let 中定义多个变量，用逗号分隔。

in: 关键字，标志着变量定义部分的结束和 let 表达式主体部分的开始。

<expr_body>: 这是 let 表达式的主体。它可以是一个简单的表达式，也可以是一个由 {} 包围的复杂表达式块。
在 in 之后定义的变量（如 <id1>, <id2>）只在 <expr_body> 这个作用域内有效。

- 核心特性：let 是一个表达式
- 最重要的一点是，整个 let ... in ... 结构本身是一个表达式，它有自己的值。它的值就是其主体 `<expr_body>` 的值。
- 为什么有的 {} 后面不需要分号 ;?
- 这个问题触及了 Cool 语言的另一个核心设计理念：一切皆为表达式，以及分号 ; 的真正作用。
- 在 Cool 中，分号 ; 不是语句结束符（不像 C++ 或 Java），而是 表达式分隔符。它用于在一个序列中分隔多个表达式。
- 由花括号 {} 包围起来的一系列表达式被称为“块表达式”。它的语法是：{ `<expr1>; <expr2>; ...; <exprN>;` } 这个块本身也是一个表达式，它的值等于块中最后一个表达式 `<exprN>` 的值。前面的表达式 `<expr1>` 到 `<exprN-1>` 会被依次求值，但它们的结果会被丢弃（当然，它们的副作用，如赋值操作，会保留下来）。

现在我们来分析不同情况下的 {}:

- 情况 1: 需要分号分隔表达式的 {}

- 9 {
- 10 fact <- fact * i;
- 11 i <- i - 1;
- 12 }

- 这个 {} 位于 while 循环体内。
- 它包含两个独立的表达式: fact <- fact * i 和 i <- i - 1。
- 分号 ; 在这里是必须的, 因为它用来分隔这两个表达式。
- 这个块表达式的值是 i <- i - 1 的值, 但 while 循环会忽略其循环体的返回值, 所以这里的返回值没有实际意义。

情况 2：定义方法/类主体，结尾需要分号的 {}

- 6 `fact(i : Int) : Int { ... };` -- 注意这里的分号

和

- 1 `class Main inherits A2I{ ... };` -- 注意这里的分号

- 在这里，{} 的作用不是创建一个块表达式，而是用来定义一个类（class）或方法（method）的主体。
- 语法规则，一个完整的类定义或方法定义必须以分号；结尾。
- 所以，这里的 {...}; 不是一个块表达式后面跟了一个分号，而是 `class Name { body };` 和 `methodName(...) : Type { body };` 这个整体语法结构的一部分。这个分号是用来结束定义的。

- 作为表达式分隔符：当 `{ }` 内部有多个表达式需要按顺序执行时，它们之间用 `;` 分隔。
- 例如：`{ expr1; expr2; expr3 }`
- 作为定义结束符：当 `{ }` 用于定义一个类或方法的主体时，整个定义（包括 `{ }`）的末尾需要一个 `;` 来结束这个定义。
- 例如：`class T { ... };`
- 例如：`method() : T { ... };`

更更更复杂的COOL的例子：list类实现

```
cool@cool: ~/workspace
1 class Main inherits IO{
2     main():Object{
3         let hello : String <- "Hello",
4           world : String <- "world",
5           newline : String <- "\n"
6         in
7           out_string(hello.concat(world.concat(newline)))
8     };
9
10 };
11
```

```
cool@cool:~/workspace$ coolc 2.cl atoi.cl
cool@cool:~/workspace$ spim 2.s
SPIM Version 6.5 of January 4, 2003
Copyright 1990-2003 by James R. Larus (larus@cs.wisc.edu)
All Rights Reserved.
See the file README for a full copyright notice.
Loaded: ../lib/trap.handler
Helloworld
COOL program successfully executed
cool@cool:~/workspace$
```

```
cool@cool: ~/workspace
#include <iostream>
using namespace std;

int main(){
    for (string a="hello",b="world",st="d";
         st[0]!='50';
         std::cout<<a+b<<std::endl);

    return 0;
}
```

```
cool@cool:~/workspace$ g++ -o a a.cpp
cool@cool:~/workspace$ ./a
helloworld
cool@cool:~/workspace$
```

```
cool@cool: ~/workspace
1 class List{
2     item: String;
3     next: List;
4     init(i: String, n: List):List{
5         {
6             item <- i;
7             next <- n;
8             self;
9         }
10    };
11
12    flatten(): String{
13        if (isvoid next) then
14            item
15        else
16            item.concat(next.flatten())
17        fi
18    };
19 };
20
21 class Main inherits IO{
22     main():Object{
23         let hello : String <- "Hello| " ,
24             world : String <- "World",
25             newline : String <- "\n",
26             nil : List,
27             list : List <- (new List).init(hello,
28                                     (new List).init(world,
29                                     (new List).init(newline, nil)))
30         in
31             out_string(list.flatten())
32     };
33
34 };
35
```

```
cool@cool:~/workspace$ coolc 2.cl
cool@cool:~/workspace$ spim 2.s
SPIM Version 6.5 of January 4, 2003
Copyright 1990-2003 by James R. Larus (larus@cs.wisc.edu).
All Rights Reserved.
See the file README for a full copyright notice.
Loaded: ../lib/trap.handler
Hello World
COOL program successfully executed
cool@cool:~/workspace$ |
```



改为任意对象，而非字符串

```
cool@cool:~/workspace$ coolc 2.cl atoi.cl
cool@cool:~/workspace$ spim 2.s
SPIM Version 6.5 of January 4, 2003
Copyright 1990-2003 by James R. Larus (larus@cs.wisc.edu).
All Rights Reserved.
See the file README for a full copyright notice.
Loaded: ../lib/trap.handler
Hello World 42
COOL program successfully executed
cool@cool:~/workspace$
```

```
class List inherits A2I{
  item: Object;
  next: List;
  init(i: Object, n: List):List{
    {
      item <- i;
      next <- n;
      self;
    }
  };

  flatten(): String{
    let string : String <-
      case item of
        i : Int => i2a(i);
        s : String => s;
        o: Object => { abort(); ""; };
      esac

    in if (isvoid next) then
      string
    else
      string.concat(next.flatten())
    fi
  };
};

class Main inherits IO{
  main():Object{
    let hello : String <- "Hello ",
        world : String <- "World ",
        i : Int <- 42,
        newline : String <- "\n",
        nil : List,
        list : List <- (new List).init(hello,
          (new List).init(world,
            (new List).init(i,
              (new List).init(newline, nil))))

    in
      out_string(list.flatten())
  };
};
```



作业1 - 9月30日前

- 使用COOL语言完成二叉树的构建。因为这是我们这节课的核心要实现的语言，还是需要简单熟悉下的。请选择 其中一个 进行实现。每个数据结构后面都列出了建议实现的最小功能集。

栈 (Stack - LIFO)

push(item): 将一个元素压入栈顶。
pop(): 移除并返回栈顶元素。如果栈为空，应妥善处理（例如，中止程序并报错或返回void）。
peek(): 返回栈顶元素，但不移除它。
isEmpty(): 检查栈是否为空，返回 Bool。
print(): 打印栈内所有元素，用于调试。

二叉搜索树 (Binary Search Tree) (挑战项)

insert(value: Int): 插入一个整数值。
注意：二叉搜索树不允许重复值。
search(value: Int): 查找一个值是否存在，返回 Bool。
printInOrder(): 以中序遍历（左-根-右）的方式打印树中所有节点的值。
这将得到一个有序序列。

队列 (Queue - FIFO)

enqueue(item): 将一个元素加入队尾。
dequeue(): 移除并返回队首元素。如果队列为空，应妥善处理。
front(): 返回队首元素，但不移除它。
isEmpty(): 检查队列是否为空。
print(): 打印队列内所有元素。

单向链表 (Singly Linked List)

insert(item): 在链表头部或尾部插入一个元素。
delete(item): 删除链表中第一个匹配的元素。
search(item): 查找元素是否存在于链表中，返回 Bool。
isEmpty(): 检查链表是否为空。
print(): 打印链表中所有元素

哈希表 (Hash Table) (挑战项)

你需要实现一个简单的哈希函数（例如，对整数使用取模运算）。
你需要选择一种冲突解决方法，建议使用“拉链法”（Chaining），即在冲突的位置挂一个链表。
put(key: Int, value: String): 插入一个键值对。
get(key: Int): 根据键查找对应的值。如果键不存在，应妥善处理。
remove(key: Int): 删除一个键值对。

- 你需要提交一份 PDF 格式的实验报告，报告中必须包含一个公开的 GitHub 仓库链接。

GitHub 仓库要求：

仓库必须为 公开 (Public) 状态。

根目录下必须包含一个 README.md 文件，简要说明你的项目是什么以及如何运行你的 COOL 代码（例如，`coolc your_file.cl && spim your_file.s`）。

包含你所有的 COOL 源代码文件（.cl 文件）。

我们鼓励你有清晰的 commit 记录，但这不作强制要求。

PDF 报告要求：

报告应包含以下几个部分：

封面：注明你的姓名、学号、所选的数据结构。

摘要：简要介绍你实现了什么数据结构，以及报告的主要内容。

设计与实现：

详细描述你的设计思路。你定义了哪些 COOL 类？每个类的属性和方法是什么？类与类之间是什么关系（如继承、组合）？

解释核心算法的实现逻辑。例如，你是如何实现队列的 `dequeue` 操作或二叉树的 `insert` 操作的？可以附上关键代码片段并加以解释。

讨论你在实现过程中遇到的主要挑战以及解决方案。

测试与结果：

展示你的 Main 类中的测试代码。

粘贴你的程序运行的最终输出结果截图或文本。

解释你的测试结果如何证明了你的实现是正确的。

结论：总结本次作业的收获和体会。

GitHub 链接：在报告的显著位置（例如封面或摘要后）提供你的 GitHub 仓库链接。



五、评分标准

评分项	权重	说明
代码功能正确性	40%	数据结构的核心功能是否全部正确实现；是否能处理边界情况（如空栈、空队列）。
COOL语言应用	20%	是否恰当地使用了类、继承、方法等面向对象特性；代码风格是否清晰、规范。
报告质量	25%	报告结构是否完整、内容是否详实、逻辑是否清晰、文字是否流畅。
代码管理与文档	15%	GitHub仓库是否按要求创建；README.md 是否规范；代码注释是否清晰。
总计	100%	


```
1 (*
2 * stack.cl
3 *
4 * 一个用COOL语言实现的通用栈数据结构。
5 * 这个栈可以存储任何继承自Object的类型的元素，
6 * 但为了演示，我们主要关注Int和String。
7 *
8 *)
9
10 (*
11 * StackNode 类
12 * 代表栈中的一个节点。它包含一个数据项(item)和指向下一个节点的指针(next)。
13 * 这是一个典型的链表节点实现。
14 *)
15 class StackNode inherits Object {
16   item : Object;      -- 节点存储的数据，类型为Object使其通用
17   next : StackNode;   -- 指向栈中的下一个节点
18
19   -- 初始化节点
20   init(i : Object, n : StackNode) : StackNode {
21     {
22       item <- i;
23       next <- n;
24       self;
25     }
26   };
27
28   -- 返回节点的数据
29   getItem() : Object {
30     item
31   };
32
33   -- 返回下一个节点
34   getNext() : StackNode {
35     next
36   };
37 };
38
```

```
40 (*
41 * Stack 类
42 * 实现了栈的核心功能。内部使用StackNode构成的链表来存储数据。
43 * 栈顶由属性'top'表示。
44 *)
45 class Stack inherits IO {
46   top : StackNode; -- 指向栈顶的节点，如果栈为空，则为 void
47
48   -- isEmpty(): 检查栈是否为空
49   -- 如果 top 是 void, 说明栈里没有节点。
50   isEmpty() : Bool {
51     isvoid top
52   };
53
54   -- push(item: Object): 将一个元素压入栈顶
55   -- 创建一个新节点，让它指向旧的栈顶，然后更新栈顶为这个新节点。
56   push(item : Object) : SELF_TYPE {
57     {
58       let new_node : StackNode <- (new StackNode).init(item, top) in
59       top <- new_node;
60       self;
61     }
62   };
63
64   -- peek(): 返回栈顶元素，但不移除它
65   -- 如果栈为空，则中止程序并报错。
66   peek() : Object {
67     if isEmpty() then
68       {
69         out_string("Error: peek from an empty stack.\n");
70         abort();
71         new Object; -- abort()会中止程序，这行是为了让类型检查器满意
72       }
73     else
74       top.getItem()
75     fi
76   };
77
78   -- pop(): 移除并返回栈顶元素
79   -- 如果栈为空，则中止程序并报错。
80   pop() : Object {
81     if isEmpty() then
82       {
83         out_string("Error: pop from an empty stack.\n");
84         abort();
85         new Object; -- 同样，这行是为了通过类型检查
86       }
87     else
88       let item_to_return : Object <- top.getItem() in
89       {
90         top <- top.getNext();
91         item_to_return;
92       }
93     fi
94   };
95
96   -- print(): 打印栈内所有元素，从栈顶到栈底
97   -- 使用一个循环遍历所有节点，并根据类型打印。
98   print() : SELF_TYPE {
99     { -- <--- 这里是修正的关键：添加了起始花括号
100       if isEmpty() then
101         out_string("Stack is empty.\n")
102       else
103         {
104           out_string("---- Top of Stack ----\n");
105           let current : StackNode <- top in
106           -- 使用循环遍历链表
107           while not (isvoid current) loop
108             {
109               -- 使用 case 语句判断元素的具体类型，并调用合
110               case current.getItem() of
111                 s : String => { out_string(s); out_string("\n"); }
112                 i : Int => { out_int(i); out_string("\n"); }
113                 o : Object => out_string("Unprintable Ob");
114               esac;
115               current <- current.getNext();
116             }
117             pool;
118             out_string("---- Bottom of Stack ----\n");
119           fi;
120           self; -- self 作为整个块的返回值，类型是 SELF_TYPE，符合方法
121         } -- <--- 这里是修正的关键：添加了结束花括号
122       };
123     };
124 };
125
126 (*
127 * Main 类
128 * 用于测试我们实现的Stack。
129 *)
130 class Main inherits IO {
131   main() : Object {
132     let my_stack : Stack <- new Stack in
133     {
134       out_string("---- Stack Demo ----\n\n");
135
136       -- 1. 测试初始状态
137       out_string("Is stack empty? ");
138       if my_stack.isEmpty() then out_string("Yes\n") else out_
139       my_stack.print();
140       out_string("\n");
141
142       -- 2. 推入一些元素 (字符串和整数)
143       out_string("Pushing 'Alice', 100, 'Bob'...\n");
144       my_stack.push("Alice");
145       my_stack.push(100);
146       my_stack.push("Bob");
147       my_stack.print();
148       out_string("\n");
149
150       -- 3. 测试 peek
151       out_string("Peeking top element: ");
152       case my_stack.peek() of
153         s : String => out_string(s);
154         i : Int => out_int(i);
155         o : Object => out_string("Object");
156       esac;
157       out_string("\n\n");
158     }
159   }
160 }
```

```
91   item_to_return;
92 }
93 fi
94 };
95
96 -- print(): 打印栈内所有元素，从栈顶到栈底
97 -- 使用一个循环遍历所有节点，并根据类型打印。
98 print() : SELF_TYPE {
99   { -- <--- 这里是修正的关键：添加了起始花括号
100     if isEmpty() then
101       out_string("Stack is empty.\n")
102     else
103       {
104         out_string("---- Top of Stack ----\n");
105         let current : StackNode <- top in
106         -- 使用循环遍历链表
107         while not (isvoid current) loop
108           {
109             -- 使用 case 语句判断元素的具体类型，并调用合
110             case current.getItem() of
111               s : String => { out_string(s); out_string("\n"); }
112               i : Int => { out_int(i); out_string("\n"); }
113               o : Object => out_string("Unprintable Ob");
114             esac;
115             current <- current.getNext();
116           }
117           pool;
118           out_string("---- Bottom of Stack ----\n");
119         fi;
120         self; -- self 作为整个块的返回值，类型是 SELF_TYPE，符合方法
121       } -- <--- 这里是修正的关键：添加了结束花括号
122     };
123   };
124 };
125
126 (*
127 * Main 类
128 * 用于测试我们实现的Stack。
129 *)
130 class Main inherits IO {
131   main() : Object {
132     let my_stack : Stack <- new Stack in
133     {
134       out_string("---- Stack Demo ----\n\n");
135
136       -- 1. 测试初始状态
137       out_string("Is stack empty? ");
138       if my_stack.isEmpty() then out_string("Yes\n") else out_
139       my_stack.print();
140       out_string("\n");
141
142       -- 2. 推入一些元素 (字符串和整数)
143       out_string("Pushing 'Alice', 100, 'Bob'...\n");
144       my_stack.push("Alice");
145       my_stack.push(100);
146       my_stack.push("Bob");
147       my_stack.print();
148       out_string("\n");
149
150       -- 3. 测试 peek
151       out_string("Peeking top element: ");
152       case my_stack.peek() of
153         s : String => out_string(s);
154         i : Int => out_int(i);
155         o : Object => out_string("Object");
156       esac;
157       out_string("\n\n");
158     }
159   }
160 }
```