Compiler-2022

Grading Policy

Not decided yet.

About the course

编译器设计是ACM班的传统课程，这门课程旨在锻炼大家的编程能力和工程能力。往年的课程都是进行天梯制度赋分，由于大家都写累了好的数据点就贡献的越来越少了，往后逐渐变成了面向数据编程，从2018级开始回归到编译本身可能更为重要。因此，我们修正了原有语言中描述不清晰的部分，按照Standard C++和Java的语言定义方式给出一个定义，并且按照两个语言的标准编译测试集合制定出一个属于我们的语言的标准集合。欢迎大家提出修改意见和建议。

注：本文参照ISO/IEC 14882:2017 Programming Language C++以及往年的Manual做出修改。如果对数据或文档有疑问或者认为数据或文档存在错误的，请发起Issue。如果发现有可修复的错误可以直接发起Pull Requests，十分感谢！

评测的3个阶段：语义检查（Semantic Check），代码生成（Codegen），代码优化（Optimize）。

禁止行为（将会直接0分）：

1. 对数据点的特判（界定：判断数据点具有什么样的特征后输出。排除：对AST、IR等结构分析后对具有特定结构的程序进行优化的过程。）
2. 抄袭。（界定：无法通过Code Review或者代码结构相似。此项无例外排除项）
3. 你的代码不应该调用除了 Parser 以外的第三方库（ANTLR等）用于任何阶段，例如生成LLVM IR后直接调用clang/llvm工具生成是非法的，如果自己实现了生成工具除外。

关于服务器：服务器可能有漏洞，如果可以提供对服务器封堵漏洞的好的建议是十分欢迎的。在不通知负责助教的情况下，任何对服务器的破坏性行为会导致请喝茶以及支付由此产生的额外服务器维护费用。

部分术语定义

1. 未定义行为：指规范并没有定义该情况发生时语言的表现。初衷是为了给同学们提供一些自己发挥的空间，在测试数据里，这些没有定义的情况是不会发生的。可以认为未定义行为是类似于运行时会错误的东西，由于在编译阶段无法确定，因此我们就保证我们的代码不会出现。

   例子：对长度超过1M的代码的编译是未定义的。解释：我们的测试集合中没有长度超过1M的代码。

2. 语法错误：指代码违反规范的行为，你的编译器应返回非0返回值作为编译错误指示信息（必须，作为评测之一）以及你的提示信息（可选，给自己看的）。
3. 源代码：你的编译器即将编译的代码。

语言基本结构

一个标准的Mx*语言包含有以下部分：**大于等于1个函数定义，（可选）类定义，（可选）全局变量声明。**

其中：

（1） 函数定义中有且仅有1个的名字可以为main，main函数的定义仅可为**int main()**，不符合此定义的main函数或者没有定义main函数均视为语法错误。

（2）在Mx*语言中没有接口的定义，所有的函数必须有对应的函数体，反之视为语法错误。

（3）我们默认编译1M以上的代码是未定义行为。

文法规则：

1 编码与符号：

我们称为这个语言是Mx*，这个语言对大小写敏感，在字符串以外，可以使用的符号集合如下：

标识符（包括变量标识符、函数标识符、类对象标识符）：26个小写英语字母，26个大写英语字母，0，1，2，3，4，5，6，7，8，9，下划线（_）；

标准运算符：加号（+），减号（-），乘号（*），除号（/），取模（%）；

关系运算符：大于（>），小于（<），大于等于（>=），小于等于（<=），不等于（!=），等于（==）；

逻辑运算符：逻辑与（&&），逻辑或（||），逻辑取反（!）；

位运算符：算术右移（>>），算术左移（<<），按位与（&），按位或（|），按位异或（^），按位取反（~）；

赋值运算符：赋值（=）；

自增运算符：自增（++），自减（--）；

分量运算符：对象（.）；

下标运算符：取下标对象（[]）；

优先级运算符：括号（()）；

分隔符：分号（;），逗号（,），括号（{}）；

特殊符号：空格（ ），换行符（'\n'），制表符（'\t'），注释标识符（//），字符串标识符（"）。

不包括在以上符号集合内的符号出现在源代码中视为语法错误。

字符串的字符集在6.3中定义。

2 关键字：

`int bool string null void true false if else for while break continue return new class this`

3 空白字符：

空白字符、制表符、换行符在Mx*语言中除了区分词素（Token）以外没有作用。

4 注释：

行注释：从"//"开始到这一行末尾的所有内容都会被作为注释，编译的时候应当自动忽略。

该语言中有且仅有一种这样的注释，剩余在C++语言中用于表示注释的方法在我们的Mx*中被认为是未定义的行为（欢迎做一些尝试实现）。

5 标识符：

标识符的第一个字符必须是英语字母（26个大写英语字母和26个小写英语字母，下同）中的一个。第二个字符开始可以是英语字母、数字或者下划线（_）中的。标识符区分大小写并且长度超过64个字符的标识符是未定义的。

6 常量：

注：没有在以下定义的常量都是未定义的。

6.1 逻辑常量

定义`true`为真，`false`为假。

6.2 整数常量

整数常量以十进制表示，整数常量不设负数，负数可以由正数取负号得到。

编译器至少应该能处理大小范围在$[-2^{31}, 2^{31})$内的整数，首位为0的整数常量是未定义的（整数0除外），大小超过上述范围的整数是未定义的。

6.3 字符串常量

字符串常量是由双引号括起来的字符串。字符串长度最小为0，长度超过255的字符串是未定义的。

字符串中的所有字符必须是可示字符（printable character），空格或者转义字符中的一种。

转义字符有三个：`\n`表示换行符，`\\`表示反斜杠，`\"`表示双引号。

其余出现在C++语言里的转义字符是未定义的。

6.4 空值常量

定义`null`为引用类型没有指向任何值。

7 变量：

注：没有在以下定义的变量类型都是未定义的。

7.1 基础类型

1. `bool`类型：`true`为真，`false`为假。
2. `int`类型：大小范围在$[-2^{31}, 2^{31})$内的整数。
3. `void`类型：表明函数没有返回值的特殊类型，仅仅可以用于函数返回值。
4. `string`类型：字符串是引用类型，可以改变它的值但是本身不能被改变（immutable）。

任何形式的类型转换（隐式类型转换，强制类型转换）在本语言中是未定义的。

7.2 数组类型

注：该部分的`<typename>`指的是类型，可以是基础类型（除外`void`）也可以是类。`<identifier>`指的是变量标识符。

数组是一种可以动态创建的引用类型，长度不需要在声明的时候确定。

声明语句的语法要求为`<typename>[] <identifier>(=<initial sentence>);`

例如：`bool[] flag;`是一句合法声明语句，不加创建的数组在创建后对应变量值为`null`，此时访问数组下标是未定义的。

此处注意区分和传统C++声明数组的方式，我们采用的是Java的声明方式。

创建数组可以用new关键字创建。

创建数组的语句语法要求是`(<typename>[]) <identifier> = new <typename>[arraySize];`

例如：`flag = new bool[10];`是一种合法的创建方式。创建数组必须制定数组的长度，方括号中仅可以传入一个整型的数。数组长度一定小于$2^{31}-1$

在我们的Mx*中，数组一般都是通过动态创建的。而静态创建数组（形如：`<typename>[] <identifier> [arraySize]`定义数组）后默认值是Null，此时使用数组下标访问元素是未定义的。

数组内建方法：`<identifier>.size()`返回数组的长度，函数返回值为`int`。对`null`的数组对象执行`.size()`返回数组的长度是未定义的。

多维数组：我们采用交错数组来达到多维数组的效果，交错数组就是数组的数组。声明方法和C#语言保持一致，可以理解为C++语言中vector套vector的效果。

声明交错数组的语句语法要求为`<typename>[]... <identifier>(=<initial sentence>);`

例如声明一个2维数组的语句可以是：`int[][] graph;`

声明**创建**交错数组的语法为：`(<typename>[]...) <identifier> = new <typename>[outerSize][]..;`

创建交叉数组需要先创建最外层数组的空间，然后再创建内层数组空间。类似于C++的std::vector。

例如声明创建一个2维数组的语句可以是：

```
int[][] graph = new int[3][];
graph[0] = null; // Valid
graph[1] = new int[10];
graph[2] = new int[30];
```

交叉数组的声明创建还可以有一种简单的方法，即同时声明创建多层数组空间：

`(<typename>[]...) <identifier> = new <typename>[size1][size2]...([]...);`

例如声明创建一个大小为3*4的2维数组的语句可以是：

`int[][] graph = new int[3][4];`

这个创建方法在Java中支持，并且看上去也比较简洁。

常量数组：这个在我们的语言中是未定义的，但是可以做一下尝试。

特别的，**数组可以被赋值，我们定义数组赋值是指针赋值。**

8 类：

我们的语言需要面向对象，类的定义的方式如下：

```
class <ClassIdentifier> {
  <Type 1> <MemberIdentifier 1>;
  <Type 2> <MemberIdentifier 2>, <MemberIdentifier 3>..;
  <Type 3> <FunctionIdentifier>(<FunctionParameterList>){
    <Expressions and Statements>
  }
  <ClassIdentifier>(){ // Can be ignored
    <Expressions and Statements>
  }
};
```

我们的语言需要面向对象，类的定义的方式如下：

8.1 类成员变量

对于没有**赋初值**的类成员变量访问是未定义行为。所有的类成员变量都是`public`的，我们对于`private`的对象是未定义的行为。

8.2 类方法

对于类方法，要求和第九部分函数的要求相同（除了构造函数），语法如下：

```
<Type> <FunctionIdentifier>(<FunctionParameterList>){
  <Expressions and Statements>
}
```

8.3 类成员访问

对于类成员不论是方法还是变量，都可以用对象标识符.取对象，对于除了字符串`string`的基本类型`int`，`bool`返回一个实值，剩下返回的应当是一个引用。语法如下：
`<ClassObjectIdentifier>.<ClassMember>;`
OR
`<ClassObjectIdentifier>.<ClassMethod>(<FunctionParameterList>);`

8.4 类构造函数

构造函数的定义和C++相同，无返回值无参数（有参数的未定义），可以没有构造函数，语法如下：

```
<ClassIdentifier>(){ // Can be ignored
  <Expressions and Statements>
}
```

无显式构造函数的类有一个和类名相同的默认构造函数，默认构造函数什么都不做。

8.5 this指针

`this`指针返回某个类的引用对象，关键字仅在类作用域内可以使用。不在类作用域内的`this`应当视为语法错误，`this`指针作为左值视为语法错误。

例子：语法正确

```
class foo {
  int a;
  int b;
  int c;
  foo test(){ return this; }
};
```

例子：语法错误

```
class foo {
  int a;
  int b;
  int c;
};
foo test(){ return this; }
```

8.6 类之中的未定义行为

析构函数、虚函数、类的继承、接口、权限标示、抽象类、成员的默认初始化表达式、函数重载。

9 函数：

9.1 函数定义

标准的函数定义应该满足如下语法：

```
<ReturnType> <FunctionIdentifier>(<FunctionParameterList>){
  <Expressions and Statements>
}
```

注意在Mx*中没有方法声明函数的签名，也不支持在一个函数内嵌套申明另一个子函数或类。**Lambda 表达式与匿名函数在Codegen、Optimize阶段是未定义的。**

9.2 内建函数

以下函数是系统包括的函数，不需要声明就可以使用。

函数：`void print(string str);`

作用：向标准输出流中输出字符串str。

函数：`void println(string str);`

作用：向标准输出流中输出字符串str，并且在行尾处输出一个换行符。

函数：`void printInt(int n);`

作用：向标准输出流中输出数字n。

函数：`void printlnInt(int n);`

作用：向标准输出流中输出数字n，并且在行尾处输出一个换行符。

函数：`string getString();`

作用：从标准输入流中读取一行并且返回。

函数：`int getInt();`

作用：从标准输入流中读取一个整数，遇到空格、回车符、制表符作为分隔，返回这个整数。

函数：`string toString(int i);`

作用：把整数i转换为字符串。

9.3 函数返回值

如果函数声明的返回值不是`void`，应有`return`语句返回函数返回值，反之未定义。（例外：`main`函数例外，可以没有返回值，此时返回值为0。）

如果函数声明的返回值是`void`，`return`语句后必须单独出现表示从此位置跳出函数（`return;`），反之语法错误。该语法错误意思为返回值为`void`的函数只能单独使用，即`return;`，任何`return <expr>;`均为语法错误。

在Semantic Check阶段，返回类型非`void`的返回值需要检查函数中每一个`return`语句的返回类型是否正确，作为语法检查。

返回值可以是自定义类，可以是数组，可以是自定义类的数组。对于数组类型的返回值需要检查维数是否正确，也就是2维数组不可以返回给1维数组，保证维度上长度一定是匹配的。

例如：以下的情况不会出现在测试点中：

```
int[] foo(int args){
  return new int[args];
}

int main(){
  int[] vec = foo(10);
  return vec[11]; // Out-of-range Exception.
}
```

9.4 Lambda 表达式：

Lambda表达式已经成为现代程序语言中非常重要的一个功能，它可以简化代码。例如如下的两种代码是等效的。

```
Arrays.sort(array, new Comparator<String>() {
    public int compare(String s1, String s2) {
        return s1.compareTo(s2);
    }
});
```

```
Arrays.sort(array, (s1, s2) -> {
            return s1.compareTo(s2);
        });
```

为了让大家熟悉Lambda表达式，我们的编译器需要支持一个非常简单的Lambda表达式。**为了简化，Lambda表达式仅出现在semantic阶段，codegen与optimize阶段出现Lambda表达式是未定义的。Lambda表达式本身作为对象是未定义的。**

同样为了简化，我们的Lambda表达式语法很单一，没有特别的语法糖。

基本语法：`[&](Parameters) -> {Statements}`

解释：`[&]`作为关键字符。其余定义同函数。如果参数为空，参数括号可以省略，调用括号不可以省略。

作用域的解释：Parameters如果出现和全局域重名的变量，应当遵循作用域遮蔽原则。为了简化，Expressions和Statements默认可以访问顶层域的所有对象。

在这里，我们借用C++中**省略返回值的形式**。你的编译器应该支持通过return语句分析返回值并判断，如果Lambda表达式函数体没有return语句，那么返回值为void。

参数可以留空。

Lambda表达式的调用同函数。

例子：

```
int sum = [&](int a, int b) -> { return a + b; }(1, 2); // 正确
int sum2 = [&]() -> { return sum; }(12); // 错误
int sum3 = [&]() -> { return sum; }(); // 正确
int foo = [&] -> {return 1;}(); // 正确
```

10 表达式：

10.1 单目表达式

单目表达式有常量，标识符变量名。等等

10.2 双目表达式

双目表达式的定义和C++类似，在类型`int`,`bool`中，要求表达式两边的对象类型必须一致而表达式两边的对象的常量/变量属性没有特别要求（除了赋值，参阅左值部分定义）。`bool`类型仅可以做`==`和`!=`比较，出现`<=, >=, <, >`的比较是语法错误的。数组对象可以和`null`比较（仅限`!=, ==`）但是不能运算。类对象的`==`和`!=`比较是地址的比较，其它运算符重载是未定义的。字符串部分的参阅字符串部分定义。

特殊的是：自增自减运算符在前缀加意义下表达式返回本身的值`+1`或`-1`的值。后缀加/减返回`int`。

例如：`a = 1;b = ++a; // After Execution: a = 2, b = 2`

11 语句：

11.1 变量声明语句

此处的变量不是类成员变量，类成员变量的定义参阅类的定义。变量声明语句语法如下：

`<Type 1> <VariableIdentifier 1>, <VariableIdentifier 2>;`

或者

`<Type> <VariableIdentifier 1> = <Initial Expression>;`

或者同时声明多个变量和初始值：

`<Type> <VariableIdentifier 1> = <Initial Expression 1>, <VariableIdentifier 2> = <Initial Expression 2>, <VariableIdentifier 3> = <Initial Expression 3>`

变量在使用之前应当被赋值了，没有赋值的对象直接使用是未定义行为。

对于自定义类的对象声明如果没有进行`new`的实例化操作，默认为null，允许没有赋初值的对象（此时为`null`，保证仅出现在semantic检查阶段）。

举例：

```
class A{
  int a;
};

A a; // 此时a为null，视为语法正确
int t = a.a; // 语法正确
```

实例化操作的语法应当为：`<Type> <VariableIdentifier> = new <Type>();` 或者 `<Type> <VariableIdentifier> = new <Type>;`

即圆括号可以省略。

11.2 条件语句

条件语句语法要求如下：

```
if (condition) {
  <Expressions and Statements if true>
} else {
  <Expressions and Statements if false>
}
```

其中`condition`字段必须返回`bool`值，并且不能为空。如果`condition`返回了非`bool`值或者空应当视为语法错误。

一个`if`语句可以没有`else`部分。若大括号中仅有一个 Expression 或 Statement，则可以省略大括号。如下例：

```
if (condition) <Expression and Statement if true>
```

11.3 循环语句

`while`循环语句语法要求如下：

```
while (condition) {
  <Expressions and Statements if true>
}
```

如果`condition`返回非`bool`值或者为空应当视为语法错误。

`for`循环语句语法要求如下：

```
for (init; condition; incr) {
  <Expressions and Statements if true>
}
```

如果`condition`返回了非`bool`值视为语法错误，但是可以为空。

若大括号中仅有一个 Expression 或 Statement，则可以省略大括号。如下例：

```
while (condition) <Expression and Statement if true>
for (init; condition; incr) <Expressions and Statements if true>
```

11.4 跳转语句

`return/break/continue`语法要求如下：

```
return <Expression>;
break;
continue;
```

`return`只在函数中有效，不在函数中的`return`视为语法错误。
`break`，`continue`只在循环中有效，不在循环中的`break`，`continue`视为语法错误。

11.5 表达式语句

语句可以只有一个单目表达式或双目表达式，此时返回值被丢弃。形如以下的语句都是合法的：
`++a; (a); (++a); a + a;`

12 字符串：

12.1 字符串对象

字符串对象赋值为`null`是语法错误。

12.2 字符串的双目运算

`+`表示字符串拼接

`==, !=`比较两个字符串是否完全一致（不是内存地址）

`<, >, <=, >=`用于比较字典序大小

剩余双目运算符都是语法错误，字符串双目运算符要求两边类型相同，不满足则语法错误。

12.3 字符串的内建方法

函数：`int length();`

使用：`<StringIdentifier>.length();`

作用：返回字符串的长度。

函数：`string substring(int left, int right);`

使用：`<StringIdentifier>.substring(left, right);`

作用：返回下标为`[left, right)`的子串。

函数：`int parseInt();`

使用：`<StringIdentifier>.parseInt();`

作用：返回一个整数，这个整数应该是该字符串的最长前缀。如果该字符串没有一个前缀是整数，结果未定义。如果该整数超界，结果也未定义。

函数：`int ord(int pos);`

使用：`<StringIdentifier>.ord(pos);`

作用：返回字符串中的第pos位上的字符的ASCII码。下标从0开始编号。

常量字符串不具有内建方法，使用内建方法的常量字符串未定义。

13 作用域：

1. 在一段语句中，由{和}组成的块会引进一个新的作用域。
2. 用户定义函数入口会引入一个新的作用域。
3. 用户定义类的入口会引入一个新的作用域，该作用域里声明的所有成员，作用域为整个类。
4. 全局变量和局部变量不支持前向引用，作用域为声明开始的位置直到最近的一个块的结束位置。
5. 函数和类的声明都应该在顶层，作用域为全局，支持前向引用（forward reference）。
6. 不同作用域的时候，内层作用域可以遮蔽外层作用域的名字。

注意：诸如`for`等表达式没有大括号也会引入一个新的作用域，如下：

```
int a = 0;
for(;;) int a = 0;
```

可以通过clang编译。

14 命名空间：

在同一个作用域里，变量，函数，和类，都分别不能同名（即变量不能和变量同名，其余同理）。如果重名视为语法错误，注意作用域规则。在作用域内，变量和函数可以重名，但是类不可以和变量、函数重名。

15 左值：

由以下方法给出的对象为左值，可以被赋值。

1. 函数的形参。
2. 全局变量和局部变量。
3. 类的一个成员。
4. 数组对象的一个元素。
5. 前置++/--的返回值 （e.g. (++(++x))++是合法的） （特别地：前缀/后缀加出现在等号左边是未定义的。）

我们的Mx*要求至少支持上述五种类型的左值。更多的左值是未定义的（注意不是语法错误）。

编译规则：

1 编译规则：

目标程序将通过`stdin`传入，编译后的程序应当通过`stdout`输出。

编译要求：

目标汇编：RISC-V **32bit, Integer Extended**

gcc构筑命令：`./configure --prefix=/opt/riscv --with-arch=rv32ima --with-abi=ilp32`

基本的运行阶段：

1. 生成你的编译器 / Build your compiler : 使用系统编译器编译你的编译器代码构建你的编译器的过程。
2. 编译目标代码 / Compile target code : 使用构建的编译器编译Mx*语言，如果编译正确输出目标汇编代码，反之编译器应当以非0返回值退出。
3. 执行目标代码 / Execute target code : 使用模拟器运行你的代码的过程。

2 评测指标和基准线的划分

评测指标：时间、准确性

1. 时间：采用模拟器运行，计算准确的周期数作为程序运行时间。对于同一个Commit不会重复评测。
2. 准确性：给定输入的情况下，评测输出和程序返回值是否和标准相同。评测输出**不会去除行末空格换行符，也不会去除文末换行符**

3 FAQ

有任何问题请在这个Repo直接发起Issue，对语言规则有疑问的使用Question标签，发现编译器评测的bug的使用Bug标签，对测试集的问题/发现测试集重的bug的使用`benchmark and data`/`benchmark-URGENT`标签。

1. 关于提交：

你需要提交一个repo，且repo的根目录中必须包括：build.bash（类似于makefile中的make all的角色），semantic.bash（类似于g++ -fsyntax-only），codegen.bash（类似于g++ xxx.cpp），optimize.bash（类似于g++ xxx.cpp -O2）。

2. 关于内部已经存储的库：

为了减少git所消耗的时间，所需要的部分依赖库已经集成在Docker中，以下均为绝对路径。

Component	File Name	Path	Language
ANTLR 4.9.1	antlr-4.9.1-complete.jar	`/ulib/java/antlr-4.9.1-complete.jar`	Java
ANTLR 4.9	antlr-4.9-complete.jar	`/ulib/java/antlr-4.9-complete.jar`	Java
ANTLR 4.8	antlr-4.8-complete.jar	`/ulib/java/antlr-4.8-complete.jar`	Java
ANTLR 4.7.2	antlr-4.7.2-complete.jar	`/ulib/java/antlr-4.7.2-complete.jar`	Java
ANTLR 4.7.1	antlr-4.7.1-complete.jar	`/ulib/java/antlr-4.7.1-complete.jar`	Java
ANTLR 4.7	antlr-4.7-complete.jar	`/ulib/java/antlr-4.7-complete.jar`	Java
ANTLR 4.6	antlr-4.6-complete.jar	`/ulib/java/antlr-4.6-complete.jar`	Java

需要的库请联系TA，放入Docker。

Update Log

Apr 23rd: 补全变量声明语句的语法；更新了类的`==`和`!=`规则。