1 ОПИСАНИЕ ЯЗЫКА

1.1 Алфавит

Алфавит (или множество литер) языка программирования Chalk основывается на множестве символов таблицы кодов ASCII. Алфавит Chalk включает:

1. строчные и прописные буквы латинского алфавита (мы их будем называть буквами),
2. цифры от 0 до 9 (назовём их буквами-цифрами),
3. символ '_' (подчерк - также считается буквой),
4. набор специальных символов:
   " { } , | [ ] + - % / \ ; ' : ? < > = ! & # ~ ^ . *
5. прочие символы

Алфавит Chalk служит для построения слов, которые в Chalk называются лексемами. Различают пять типов лексем:

- идентификаторы,
- ключевые слова,
- знаки (символы) операций,
- литералы,
- разделители.

Идентификаторы должны выбираться с учетом следующих правил:

1. Они должны начинаться с буквы латинского алфавита (a,...,z, A,...,Z) или с символа подчеркивания (_).
2. В них могут использоваться буквы латинского алфавита, символ подчеркивания и цифры (0,...,9). Использование других символов в идентификаторах запрещено.
3. В языке Chalk буквы нижнего регистра (a, ... ,z), применяемые в идентификаторах, отличаются от букв верхнего регистра (A,...,Z). Это означает, что следующие идентификаторы считаются разными: id, Id, ID и т.д.
4. Идентификаторы для новых объектов не должны совпадать с ключевыми словами языка и именами стандартных функций из библиотеки.

Комментарии обрамляются символами **#!** и **!#**. Их можно записывать в любом месте программы. Все, что находится после знака # до конца текущей строки, будет также рассматриваться как комментарий.

Пробелы, символы табуляции и перехода на новую строку в программах на Chalk игнорируются. Это позволяет записывать различные выражения в

хорошо читаемом виде. Кроме того, строки программы можно начинать с любой позиции, что дает возможность выделять в тексте группы операторов.

1.2 Ключевые слова

Ключевые слова - это зарезервированные идентификаторы, которые наделены определенным смыслом. Их можно использовать только в соответствии со значением известным компилятору языка Chalk.

Приведем список ключевых слов:

String	**Int**	**Float**	**Bool**	**def**
end	**true**	**false**	**while**	**do**
until	**if**	**elsif**	**else**	**for**
in	**return**	**break**	**continue**	**print**
println	**entry**	**CONST**	**readInt**	**readFloat**
readString	**readBool**	**div**	**mod**	

1.3 Структуры данных

В языке Chalk используется 4 типа данных.

Тип **Int** служит для представления 32-битных целых чисел со знаком. Диапазон допустимых для этого типа значений — от **-2147483648** до **2147483647**.

Числа с плавающей точкой (вещественные числа) используются при вычислениях, в которых требуется использование дробной части. В языке Chalk для вещественных чисел используется 64ти четырех разрядный тип **Float**. Диапазон допустимых для этого типа значений — от **1. 7e-308** до **1. 7e+ 308**.

В языке Chalk имеется простой тип **Bool**, используемый для хранения логических значений. Переменные этого типа могут принимать всего два значения — **true** (истина) и **false** (ложь). Значения типа **Bool** возвращаются в качестве результата всеми операторами сравнения, например (a < b). Кроме того, **Bool** — это тип, требуемый всеми условными операторами управления — такими, как **if**, **while**, **do-until**.

Тип **String** служит для работы со строками. Строка — это последовательность символов. Каждый символ занимает 1 байт памяти (код

ASCII). Строки переменной длины. Никаких операций со строками в языке Chalk производить нельзя, их можно только прочитать с клавиатуры и вывести на экран при помощи соответствующих функций.

Одни типы можно приводить к другим типам. В языке Chalk приведение типов осуществляется при помощи следующей конструкции: [**type**]:[**id** | **literal**], где **type —** один из четырех возможных типов, **id** — идентификатор, **literal** — литерал. На данный момент в языке возможно следующее приведение **Float — Int** и **Int — Float**.

1.4 Синтаксические конструкции

1. *Оператор условного перехода*

общий вид: **if** <expr> <stmts> (**elsif** <expr> stmts)* [**else** stms] **end**

После ключевого слова **if** следует логическое выражение и последовательность операторов и операторов присваивания. Ветвление осуществляется при помощи ключевых слов **elsif** и **else**. Оператор заканчивается ключевым словом **end**.

Пример: ***if a > 1***

a -= 1

elsif a <= 0

a = a + b

end

2. *Оператор цикла while*

общий вид: **while** <expr> <stmts> **end**

После ключевого слова **while** следует логическое выражение и последовательность операторов и операторов присваивания. Оператор заканчивается ключевым словом **end**. Цикл выполняется до тех пор, пока истинно выражение <expr>.

Пример: ***while (c && b)***

b = c && b || g

end

3. *Оператор цикла for*

общий вид: **for** <id**> in [**<expr> **:** <expr>**]** <stmts> **end**

После ключевого слова **for** следует идентификатор, далее ключевое слово **in** и указынный в квадратных скобках отрезок, который задается своей левой и правой границами. В качестве границ могут выступать как числовые литералы, так и числовые переменные. Оператор заканчивается ключевым словом **end**.

Цикл выполняется с шагом **+1** до тех пор, пока не будет превзойдена правая граница заданного отрезка.

Пример: ***for count in [-15 : h]***

print count, «, »

end

4. *Оператор цикла do — until*

общий вид: **do** <stmts> **until** <expr>

После ключевого слова **do** следует последовательность операторов языка, далее - ключевое слово **until**, после которого следует логическое выражение. Цикл выполняется до тех пор, пока выражение <expr> истинно. Отличие этого способа организации цикла, от цикла **while** заключается к том, что в цикле **do - until** условие проверяется после выполнения очередной итерации, а не перед.

Пример: ***do***

b = readInt

until b != 5

5. *Оператор присваивания*

общий вид: <id> = <expr>

Оператор присваивания служит для того, чтобы инициализировать или задать новое значение переменной.
Пример: ***count = count * 5***

var += count

6. *Математические операторы*

Для математических вычислений в языке Chalk существуют следующие операторы:

- +, -, *, / - соответственно сложение, вычитание, умножение, деление

общий вид: <expr> **op** <expr>

- <id>++, ++<id> - оператор инкрементации переменной. В случае префиксной записи значение переменной <id> увеличивается на единицу до начала операции, в случае постфиксной — после.
- <id>--, --<id> - оператор декрементации переменной. В случае префиксной записи значение переменной <id> уменьшается на единицу до начала операции, в случае постфиксной — после.

Возможны комбинации инкрементации и декрементации: --<id>++.

- **>**, **<**, **>=**, **<=**, **!=**, **==** - операторы сравнения, соответственно: больше, меньше, больше или равно, меньше или равно, не равно, равно.

общий вид: <expr> **op** <expr>

- **!** (в префиксной записи, унарный), **&&**, || - логические операторы, дающие в результате логическое значение **true** или **false**. Соответственно: отрицание, конъюнкция, дизъюнкция.

общий вид: **!**<expr>

<expr> **op** <expr>

- ~ (в префиксной записи, унарный), **&**, |, ^ - побитовые логические операторы. Соответственно: отрицание, конъюнкция, дизъюнкция, исключающее или.

общий вид: ~<expr>

<expr> **op** <expr>

- ! (в постфиксной записи, унарный) — факториал от целого числа.

общий вид: <expr>**!**

- унарные + и **-** (в префиксной записи) — соответственно: плюс возвращает модуль числа

общий вид: +<expr>

-<expr>

- ****** - оператор возведения в степень.

общий вид: <expr> ****** <expr>

7. *Onepamop break*

общий вид: **break**

Используется для принудительного прерывания цикла. Оператор прерывания выполняется для ближайшего цикла, внутри которого он находится.

Пример: ***do***

b = readInt

Int c = 0

while c < 6

c = -5 + readInt

if c == -5

break

else

println c

end

end

until b != 5

В этом примере, в случае, если переменная с окажется равна -5, выполнение цикла while будет прервано.

8. *Оператор continue*

общий вид: **continue**

Используется для принудительного перехода цикла к следующей итерации. Оператор **continue** выполняется для ближайшего цикла, внутри которого он находится.

Пример: ***for b in [c : 100]***

if b == 66

continue

else

sum += b

end

end

9. *Оператор return*

общий вид: **return** [<expr>]

Используется для принудительного выхода их подпрограммы. В случае, если функция возвращает некоторое значение, после оператор **return** должно быть указано возвращаемое значение <expr>.

Пример: ***def foo() : Int =***

return readInt

end

10. *Определение подпрограмм*

общий вид: **def** [**entry**] <id> **(**[**type** <id>**,** (, **type** <id>)*]**)** [: **type**] **=**

<stmts>

end

После ключевого слова **def** может следовать ключевое слово **entry**, которое означает, что эта функция — точка входа в программу. Далее идет идентификатор — имя функции, после имени функции в круглых скобках указывается последовательность параметров функции (параметры перечисляются через запятую, и объявляются в виде **type** <id>, где **type —** один из четырех возможных типов, а <id> - идентификатор). Их количество может быть >= 0. После закрывающейся круглой скобки может быть указан тип возвращаемого функцией значения, если он не указан, то функция никаких значений не возвращает. Далее следует символ **=**, после которого идет множество операторов языка. Объявление функции завершается ключевым словом **end**.

Пример: ***def foo() : Float =***

return readFloat / CONST.pi + 666

end

1.5 Встроенные функции

1. **print** <expr>, (, <expr>)* — выводит агрументы на экран. Не возвращает ничего.
2. **println** [<expr>, (, <expr>)*] — выводит агрументы на экран с переводом на новую строку. Не возвращает ничего.
3. **readInt** — читает целое число с клавиатуры. Возвращает Int.
4. **readFloat** — читает вещественное число с клавиатуры. Возвращает Float.
5. **readString** — читает строку с клавиатуры. Возвращает String.
6. **readBool** — читает логическое значение (**true** или **false**) с клавиатуры. Возвращает Bool.
7. **sin(**<expr>**)** — синус от выражения. Возвращает Float.
8. **cos(**<expr>**)** — косинус от выражения. Возвращает Float.
9. **tan(**<expr>**)** — тангенс от выражения. Возвращает Float.
10. **asin(**<expr>**)** — арксинус от выражения. Возвращает Float.
11. **acos(**<expr>**)** — арккосинус от выражения. Возвращает Float.
12. **atan(**<expr>**)** — арктангенс от выражения. Возвращает Float.
13. **lg(**<expr>**)** — логарифм десятичный от выражения. Возвращает Float.
14. **ln(**<expr>**)** — логарифм натуральный от выражения. Возвращает Float.
15. **log(**<expr1>, <expr2>**)** — логарифм по основанию <expr1> от выражения <expr2>. Возвращает Float.
16. **sqrt(**<expr>**)** — корень квадратный от выражения. Возвращает Float.
17. **max(**<expr>, <expr> (,<expr>)***)** — максимальное по значению из выражений. Возвращает значение в зависимости от типа аргументов.
18. **min(**<expr>, <expr> (,<expr>)***)** — минимальное по значению из выражений. Возвращает значение в зависимости от типа аргументов.
19. **swaps(**<id>, <id>**)** — обменивает переменные знаками. Не возвращает ничего.
20. **swapv(**<id>, <id>**)** — обменивает переменные их значениями. Не возвращает ничего.

В языке Chalk используются следующие константы:

1. **CONST.pi —** число пи.
2. **CONST.pi —** число e.
3. **Float.MAX_VALUE —** максимальное вещественное число.
4. **Float.MIN_VALUE —** минимальное вещественное число.
5. **Float.NEGATIVE_INFINITY —** положительная бесконечность для вещественного числа.
6. **Float.POSITIVE_INFINITY —** отрицательная бесконечность для вещественного числа.
7. **Float.NaN —** неопределенность для вещественного числа.
8. **Int.MAX_VALUE —** максимальное целое число.
9. **Int.MIN_VALUE —** минимальное целое число.

1.6 Примеры

Пример 1

```
# Chalk language syntax examples
String HELLO = "Hello, new world!\n"
# calculating the numbers of fibbonachi
def fibbonachi( Int value ): Int =
   Int count, a1, a2 = 1, a3 = 1
   while true
      if count == value || value < 0
            break
      else
         a3 = a1 + a2
         a1 = a2
         a2 = a3
         count++
      end
   end
   return a3
end

def entry main() =
   print HELLO
   Int flag
```

```
   do
         println "Input number:"
         flag = readInt
         println fibbonachi(flag)
         println "Continue? (type 0 to abort) :"
         flag = readInt
   until flag != 0
end
```

Пример 2 (с ошибками)

```
String HELLO    = "Hello, new world!\n",
       FIRST    = "First example it is a simple math expression: "
Float GLOBAL_VAR = 5.7
# a function representation
# math expression
def mathExpr( Int a, Int b, Int c ): Float =
   if (a > b && b < c)
         return sin(a) + a! - b**c + asin(a) + GLOBAL_VAR
   elsif b != c
         Int sum
         for v in [b:c]
            sum += v
         end
         return +(sum/b) + log(a, c) + ln(CONST.pi) - 1.4E+2
   else
         return max(a, b, c) * min(a, b, c) ##+ { swaps(a, b); b mod +a }
   end
end

def entry main() =
   print HELLO
   print FIRST
   println mathExpr(1, 2)
end
```

Пример 3 (с ошибками)

```
# simple comment

#!
a
multiline comment
```

```
b
!#
String HELLO    = "Hello, new world!\n",
       FIRST    = "First example it is a simple math expression: ",
       SECOND   = "Second example it is a computation of the integral: ",

Float GLOBAL_VAR = 5.7

# calculating the numbers of fibbonachi
def fibbonachi( Int value ): Int =
   Int count, a1, a2 = 1, a3 = 1

   while true
      if count == n || n < 0
            break
      else
            a3 = a1 + a2
            a1 = a2
            a2 = a3
            count++
      end
   end

   return a3
end

# a total number with count of members n
def evalMathRow( Float a, Int n ): Float =
   # that macros definition is questionable
   # def macros Y(b,c) = c + b
   # def macros Z(a,k) = a^k / k! + Y(k,k)

   Float store = 5
   for i in [0:n]
         Int j
         for j in [n: n + 20]
            store += fibbonachi(i + j) + GLOBAL_VAR
         end
   end
   return store
end
```

```
def entry main() =
    print HELLO
    print SECOND
    println evalMathRow()
    println "Bye..."
end
```

3 ВСПОМОГАТЕЛЬНЫЕ КЛАССЫ

3.1 NamesTable

```
// Класс используется для хранения сведений об идентификаторах программы.
// по строковому идентификатору имени
public class NamesTable<T extends Name>
{
	// проверяет, находится ли данное имя в таблице имен
	public boolean isExist( T name ) {...}
	// проверяет по строковому идентификатору имени, находится ли данное имя в
	// таблице имен
	// name - значение, которое возвращает метод getId() класса Name префикса
	public boolean isExist( String name ) {...}
	// добавляет новое имя в талицу имен
	public void add( T name ) {...}
	// извлекает по строковому идентификатору имени целиком все имя
	// name - значение, которое возвращает метод getId() класса Name префикса
	public T get( String name ) {...}
	// извлекает из таблицы имен по объекту имени нужное имя. Для сравнения
	// имен используется метод equals(Object)
	public T get( T name ) {...}
	// печатает содержимое таблицы имен в выходной поток out
	public void print( PrintStream out ) {...}
	// возвращает колечцию имен таблицы имен
	public Collection<T> getValues() {...}
}
```

3.2 Name

```
// Класс представляет собой имя — наименьшую единицу, которая хранится в таблице
// имен NamesTable. Имя хранит сведения о типе переменной, её имени и позиции
// в которой она объявлена.
public class Name
{
	// префикс для идентификатора. Используется для исключения совпадения имен
	// переменных и функций с ключевыми словами целевого языка
```

```
public final static String ID_PREFIX = "_";
public final static String ID_SEPARATOR = ".";
// константа для типа Int
public final static String INT_TYPE_ID = "Int";
// константа для типа Float
public final static String FLOAT_TYPE_ID = "Float";
// константа для типа Double
public final static String DOUBLE_TYPE_ID = "Double";
// константа для типа Char
public final static String CHAR_TYPE_ID = "Char";
// константа для типа String
public final static String STRING_TYPE_ID = "String";
// константа для типа Bool
public final static String BOOL_TYPE_ID = "Bool";
// константа для неопределенного типа
public final static String UNDEF_TYPE_ID = "Undef_Type";
// константа для типа Void
public final static String VOID_TYPE_ID = "Void";
// константа для NULL-значения
public final static String NULL_TYPE_ID = "NULL";
// конструкторы класса
      // задает только идентификатор
public Name( String id ) {...}
      // задает имя и относительно него, через специальный разделитель,
      // идентификатор
public Name( String relative, String id ) {...}
      // задает идентификатор, тип и номер строки
public Name( String id, String type, int line) {...}
      // задает имя и относительно него, через специальный разделитель,
      // идентификатор, далее тип и номер строки
public Name( String relative, String id, String type, int line) {...}
// добавляет номер линии, в которой встречается данный идентификатор
public void addLineUses( int line ) {...}
// получает список номеров строк, в которых встречается данный
// идентификатор
public ArrayList<Integer> getLinesUses() {...}
// получает номер строки, в которой впервые встречается данный
// идентификатор
public Integer getLine() {...}
// получает тип имени
public String getType() {...}
// устанавливает тип имени
public void setType( String type ) {...}
```

```
	// получает идентификатор имени
	public String getId() {...}
	// устанавливает идентификатор имени
	public void setId( String id ) {...}
	// проверяет, равен ли тип имени заданному типу type
	public boolean typeIs( String type ) {...}
	// true, если тип имени Float
	public boolean isFloat() {...}
	// true, если тип имени Int
	public boolean isInt() {...}
	// true, если тип имени Bool
	public boolean isBool() {...}
	// true, если тип имени String
	public boolean isString() {...}
	// true, если type это тип Float
	public static boolean isFloatT( String type ) {...}
	// true, если type это тип Int
	public static boolean isIntT( String type ) {...}
	// true, если type это тип Bool
	public static boolean isBoolT( String type ) {...}
	// true, если type это тип String
	public static boolean isStringT( String type ) {...}
	// true, если type1 равен  type2
	public static boolean typeIsT( String type1, String type2 ) {...}
	// true, если id равно идентификатору имени. id — значение без префикса
	public boolean idIs( String id ) {...}
	// возвращает строковое представление имени
	public String toString() {...}
	// сравнивает объекты класса Name
	public boolean equals( Object obj ) {...}
}
```

3.3 FuncName

```
// Класс наследует все свойства класса Name и служит для хранения инфы о функции
public class FuncName extends Name
{
	// Иннер класс, который наследует все свойства класса Name и служит
	// для хранения инфы об аргументе функции
	public static class FuncArg extends Name
```

```
{
      // конструктор класса. Задает идентификатор и тип аргумента
      public FuncArg(String id, String type) {...}
      // конструктор класса. Задает идентификатор, тип аргумента, номер
      // строки, в которой он встречается
      public FuncArg(String id, String type, int line) {...}
      // возвращает строковое представление аргумента функции
      public String toString() {...}
}
// конструктор класса. Задает идентификатор и номер строки, в которой он
// встречается
public FuncName( String id, int line ) {...}
// конструктор класса. Задает идентификатор, тип аргумента, номер
// строки, в которой он встречается
public FuncName( String id, String retType, int line ) {...}
// конструктор класса. Задает идентификатор, возвращаемое функцией
// значение, флаг «главная функция» и номер
// строки, в которой он встречается
public FuncName( String id, String retType, boolean isMain, int line ) {...}
// возвращает строковое представление функции
public String toString() {...}
// проверяет правильность вызова функции с данным набором аргументов
// callArgs
public boolean isCallCorrect( List<ExprValue> callArgs ) {...}
// возвращает строковое представление списка аргументов функции
public String args2String() {...}
// возвращает true, если это мэйн-функция
public boolean isMain() {...}
// делает функцию мэйн или не мэйн
public void setMain( boolean b ) {...}
// устанавливает список аргументов функции
public void setArgs( List<FuncArg> args ) {...}
// возвращает true, если среди аргументов функции есть аргумент с
// идентификатором argId
// argId - значение, которое возвращает метод getId() класса Name префикса
public boolean containsArgId( String argId ) {...}
// получает аргумент функции по идентификатору аргумента
// argId - значение, которое возвращает метод getId() класса Name префикса
public FuncArg getArgById( String argId ) {...}
// проверяет, есть ли среди аргументов функции данный аргумент
public boolean containsArg( FuncArg arg ) {...}
// сравнивает на равентсво два объекта класс FuncName
public boolean equals( Object obj ) {...}
```

```
}
```

3.4 Namespace

```
// Класс служит для реализации механизма пространства имен, при котором для
// переменных и функций формируются области видимости, в пределах которых они
// доступны и недоступны за их пределами. Например, переменная, объявленная
// внутри некоторой функции доступна только для неё, но не за её пределами; а
// переменная, объявленная вне всяких функций доступна для всех функций
// (это все ложь, так как она будет доступна только для функций, объявленных
// после объявления этой переменной...мир суров, суров так же, как и
// однопроходный компилятор:( )
public class Namespace
{
    // конструктор класса. Задает имя name пространства имен
    // и его владельца owner, т е пространство имен, внутри которого
    // объявляется данное пространство имен
    public Namespace( Namespace owner, String name) {...}
    // конструктор класса. Задает имя name пространства имен
    // его владельца owner, т е пространство имен, внутри которого объявляется
    // данное пространство имен, и имя-функцию, которое говорит о том, что
    // данное пространство имен — это функция
    public Namespace( Namespace owner, String name, FuncName funcName ) {...}
    // устанавливает список дочерних для данного пространств имен
    public void setChilds( List<Namespace> childs ) {...}
    // устанавливает владельца для данного пространства имен
    public void setOwner( Namespace owner ) {...}
    // получает владельца данного пространства имен
    public Namespace getOwner() {...}
    // устанавливает имя для данного пространства имен
    public void setName( String name ) {...}
    // получает имя данного пространства имен
    public String getName() {...}
    // получает таблицу имен данного пространства имен
    public NamesTable<Name> getNamesTable() {...}
    // добавляет новое ПИ в данное пространство имен
    public void addSpace( Namespace space ) {...}
    // добавляет новое имя в таблицу имен данного пространства имен
    public void addName( Name name ) {...}
    // true, если данное пространство имен — это функция
    public boolean isFuncSpace() {...}
    // возвращает объект функции данного пространства имен
    public FuncName getFuncName() {...}
```

```
        // устанавливает объект функции для данного пространства имен
        public void setFuncName( FuncName funcName ) {...}
        // возвращает тип возвращаемого данным пространством (или его родителями)
        // имен значения, разумеется, в случае, если мы наткнемся на функцию
        public String getSpaceRetType() {...}
        // найдет и вернет, если найдет объект функции по входному оргументу func
        public FuncName findFunc( FuncName func ) {...}
        // найдет и вернет, если найдет объект функции по входному оргументу id
        // id — это значение, которое возвращает метод getId() класса Name
        public FuncName findFuncById( String id ) {...}
        // найдет и вернет, если найдет объект функции по входному оргументу id
        // id — это значение, просто значение
        public FuncName findFuncById( String id, boolean isMain ) {...}
        // true, если существует main-функция
        public boolean mainExist() {...}
        // распечатывает все пространства имен со вложенностью в выходной поток
        // out с начальным сдвигом вложенности shift
        public void print( String shift, PrintStream out ) {...}
}
```

3.5 ExprValue

```
// Класс служит для описания типа значения, возвращаемого продукцией
public class ExprValue
{
        // конструкторы класса
        public ExprValue(int line, int pos ) {...}
        public ExprValue( String type, int line, int pos ) {...}
        public ExprValue( String type, int line, int pos, boolean isId ) {...}
        public ExprValue( String type, int line, int pos, boolean isId, boolean
isFunc ) {...}
        public ExprValue( String type, Point point ) {...}
        public String getType() {...}
        // проверка типов, все аналогично соответствующим методам класса Name
        public boolean isT( String type ) {...}
        public boolean isFloat() {...}
        public boolean isInt() {...}
        public boolean isBool() {...}
        public boolean isString() {...}
        // возвращает номер строки
        public int line() {...}
```

```
	// устанавливает номер строки и позицию в строке ввиде точки
	public void setPoint( Point point ) {...}
	// возвращает позицию в строке
	public int pos() {...}
	// true, если данное значение — просто идентификатор
	public boolean isId() {...}
	// true, если данное значение - функция
	public boolean isFunc() {...}
	// возвращает строковое представление данного объекта
	public String toString() {...}
}
```

3.6 ChalkEntry

```
// Мэйн-класс. Точка входа в компилятор. Обрабатывает входные параметры,
// запускает компиляцию и выводит сообщения о её ходе
public class ChalkEntry
{
	// КЭП: -Да это же метод мэйн!
	public static void main( String[] args ) throws Exception {...}
	// выводит сообщение в выходной поток (консоль либо файл)
	public static void printMsg( String frmtMsg, Object... args ) {...}
	// получает группу шаблонов
	public static StringTemplateGroup getStg() {...}
	// true, если текущая ОС семейства Windows
	public static boolean isWindows() {...}
	// true, если текущая ОС Mac
	public static boolean isMac() {...}
	// true, если текущая ОС семейства Unix
	public static boolean isUnix() {...}
}
```

4 ОШИБКИ

1. Ошибка повторного объявления переменной.

Формируется по шаблону "***error(\%d:\%d): Duplicate variable \"\%s\" declaration;"***

2. Ошибка несоответствия типов

Формируется по шаблону ***"error(\%d:\%d): Type mismatch: cannot convert from \%s to \%s;"***

3. Ошибка неверных параметров функции

Формируется по шаблону ***"error(\%d:\%d): Wrong function \"\%s\" parameters;"***

4. Ошибка несовместимости типов

Формируется по шаблону ***"error(\%d:\%d): Incompatible types \"\%s\" and \"\%s\";"***

5. Ошибка невозможности использования данной операции для этого типа

Формируется по шаблону ***"error(\%d:\%d): Operation \"\%s\" not implemented for this type \"\%s\";"***

6. Ошибка невозможности использования данной операции для этого набора типов

Формируется по шаблону ***"error(\%d:\%d): Operation \"\%s\" not implemented for this types \"\%s\" and \"\%s\";"***

7. Ошибка, переменная неопределена

Формируется по шаблону ***"error(\%d:\%d): Variable \"\%s\" is undefined;"***

8. Ошибка, функция неопределена

Формируется по шаблону ***"error(\%d:\%d): Function \"\%s\" is undefined;"***

9. Ошибка, повторное определение функции

Формируется по шаблону ***"error(\%d:\%d): Multiplay function \"\%s\" definition;"***

10. Ошибка, больше одной входной функции

Формируется по шаблону ***"error(\%d:\%d): More then one entry function \"\%s\" definition;"***

11. Ошибка, дублирование объявления параметра функции

Формируется по шаблону ***"error(\%d:\%d): Duplicate function parameter \"\%s\";"***

12. Ошибка вызова функции с данным набором параметров

Формируется по шаблону ***"error(\%d:\%d): The function \%s(\%s) in the type \%s is not applicable for the arguments \%s;"***

13. Ошибка, входная функция неопределена

Формируется по шаблону ***"error: The main function is undefined;"***

14. Ошибка, операция не может быть выполнена для данного аргумента

Формируется по шаблону ***"error(\%d:\%d): Operation \"\%s\" not implemented for this argument;"***

15. Ошибка, входная функция не может иметь никаких входных параметров и не возвращает значение

Формируется по шаблону ***"error(\%d:\%d): The main \"\%s\" function doesn't have any parameters and doesn't return any value;"***

16. Ошибка, мэйн функцию невозможно вызывать где быто-то не было

Формируется по шаблону ***"error(\%d:\%d): You can't call the main \"\%s\" function yourself anywhere;"***

17. Ошибка, в данном месте ожидается оператор присваивания

Формируется по шаблону ***"error(\%d:\%d): Assigment operator expected;"***

5 ПРИМЕР РАБОТЫ

Для работы необходимы JDK 5 и выше (в переменных окружения необходимо прописать путь к папке, в которой находится компилятор java, например, c:\Program Files\Java\jdk1.6.0_18\bin), jar-файл **chalkc.jar,** а также папка **grammar** с файлом шаблонов преобразования(это необязательно, просто по-умолчанию компилятор ищет необходимые ему файлы именно там). Файл шаблонов можно задать в опциях компилятора.

Компилятор запускается со следующими опциями:

```
chalkc.jar [[-ont] [output_file] [prog_name] [stg_file]] <src_file>
```

-o == задает имя файла, в который будет выводиться отчет о ходе компиляции. Если опция отсутствует, то вывод происходит в консоль.

-n == задает имя программы. Если опция отсутствует, то программе присвоится имя исходного файла.

-t == задает файл с шаблонами преобразования. Если опция не задана, то используется файл по умолчанию.

После флагов следуют соответственно, в той же последовательности, значения опций. Данные опции можно не указывать.

В конце указывается путь к исходному файлу с программой **<src_file>**. Файл должен иметь расширение *.halk.

На рисунке 5.1 показан успешный пример компиляции программы. В консоль выведены все переменные и функции, задействованные в программе. Сдвигами показаны пространства имен. Обнаружено 0 ошибок. В результате компиляции мы получаем файл hello.jar — скомпилированную и готовую к запуску и исполнению на JVM программу.