Чернявский А. С. КМ-71 Системы программирования

Описание подмножества языка (C++)

**1. Имена (идентификаторы)**

Допустимым идентификатором является последовательность из одной или нескольких букв, цифр или символов подчеркивания (\_), всегда начинающаяся с буквы. Ни пробелы, ни знаки препинания не могут быть частью идентификатора. Идентифтикаторы, начинающиеся с символа подчеркивания (\_) или содержащие два символа подчеркивания подряд, обычно являются зарезервированными для ключевых слов компилятора, поэтому мы тоже не будем их допускать.

Ограничение на длину идентификатора отсутствует.

Алфавит терминальных символов:

**Σ** = {

if, else, for, while, do, return, break, continue,

char, int, double, bool, void, const, true, false,

class, public, private,

0|… |9, a|…|z, A|…|Z, \_,

+, -, \*, /, !, %, &, --, ++, &&, ||,

<, >, <=, >=, ==, !=,

=, \*=, /=, %=, +=, -=,

(, ), {, }, ;, ,, ., ->, ::, ', ", ~

}.

Нетерминальные символы указываются в скобках («<» и «>»).

Алфавит нетерминальных символов:

**N** = {<программа>, <декларативный\_раздел>, <класс>, <спецификатор\_доступа>, <метод\_класса>, <реализация\_метода\_класса>, <функция>, <тип\_функции>, <список\_аргументов>, <аргумент>, <блок>, <элемент\_блока>, <выражение>, <возврат>, <оператор>, <оператор\_1>, <оператор\_2>, <оператор\_3>, <оператор\_4>, <оператор\_5>, <оператор\_6>, <оператор\_7>, <оператор\_8>, <оператор\_9>, <вызов\_функции>, <объект>, <декларация\_переменных>, <тип\_переменной>, <список\_переменных>, <идентификатор>, <цикл>, <while-цикл>, <do-while-цикл>, <for-цикл>, <ветвление>, <литерал>, <логический\_литерал>, <константа\_с\_плавающей\_точкой>, <целая\_константа>, <десятичная\_целая\_константа>, <восьмеричная\_целая\_константа>, <шестнадцатеричная\_целая\_константа>, <цифра>, <символьная\_константа>, <буква>, <строковый\_литерал> }.

В правилах грамматики будем использовать следующие метасимволы:

**{ }** — произвольное количество повторений цепочки (в том числе 0 раз);

**[ ]** — необязательная цепочка;

**( A1| A2| … | An)** — факторизация (одна из альтернативных цепочек);

**?** — любой символ (для задания символьных констант и строковых литералов).

Для указания метасимволов в качестве терминальных символов будем использовать кавычки, например: "(", "{" и т. д. Соответственно, чтобы указать кавычки, будем писать """.

Символ **«::=»** используется в правилах для выделения правой и левой частей правила.

Начальный символ грамматики — <программа>.

|  |
| --- |
| **<программа>** ::= [<декларативный\_раздел>]int main(<список\_аргументов>)**"{"**{<элемент\_блока>}**"}"**[<декларативный\_раздел>]  **<декларативный\_раздел>** ::= {(<класс>|<реализация\_метода\_класса>|<декларация\_переменных>|<функция>)}  **<класс>** ::= class <идентификатор> **"{"**{[<спецификатор\_доступа>:]{(<декларация\_переменных>|<метод\_класса>)}}**"}"**  **<спецификатор\_доступа>** ::= (public|private)  **<метод\_класса>** ::= <тип\_функции> [~]<идентификатор>"("<список\_аргументов>")"<блок>  **<реализация\_метода\_класса>** ::= <тип\_функции> <идентификатор>::[~]<идентификатор>**"("**<список\_аргументов>**")"**<блок>  **<функция>** ::= <тип\_функции> <идентификатор>**"("**<список\_аргументов>**")"**<блок>  **<тип\_функции>** ::= (<тип\_переменной>|void)  **<список\_аргументов>** ::= [<аргумент>{, <аргумент>}]  **<аргумент>** ::= <тип\_переменной>[&] <идентификатор>  **<блок>** ::= (<элемент\_блока>|**"{"**{<элемент\_блока>}**"}"**)  **<элемент\_блока>** ::= (;|<декларация\_переменных>|<выражение>;|<цикл>|<ветвление>|<возврат>)  **<выражение>** ::= (<идентификатор>|<литерал>|<оператор>)  **<возврат>** ::= return <выражение>;  **<оператор>** ::= (<оператор\_1>|<оператор>{,<оператор\_1>})  **<оператор\_1>** ::= (<оператор\_2>|<объект>(=|+=|-=|\*=|/=|%=)<оператор\_2>)  **<оператор\_2>** ::= (<оператор\_3>|<оператор\_2>"||"<оператор\_3>)  **<оператор\_3>** ::= (<оператор\_4>|<оператор\_3>&&<оператор\_4>)  **<оператор\_4>** ::= (<оператор\_5>|<оператор\_4>(==|!=)<оператор\_5>)  **<оператор\_5>** ::= (<оператор\_6>|<оператор\_5>(<|<=|>|>=)<оператор\_6>)  **<оператор\_6>** ::= (<оператор\_7>|<оператор\_6>(+|-)<оператор\_7>)  **<оператор\_7>** ::= (<оператор\_8>|<оператор\_7>(\*|/|%)<оператор\_8>)  **<оператор\_8>** ::= (<оператор\_9>|(++|--)<объект>|(+|-|!|\*)<оператор\_9>|&<объект>)  **<оператор\_9>** ::= ("("<выражение>")"|<объект>(++|--)|<вызов\_функции>|<идентификатор>(.|->)<идентификатор>)  **<вызов\_функции>** ::= [<идентификатор>(.|->)]<идентификатор>**"("**<выражение>**")"**  **<объект>** ::= <идентификатор>[(.|->)<идентификатор>]  **<декларация\_переменных>** ::= [const ]<тип\_переменной> <список\_переменных>;  **<тип\_переменной>** ::= (int|double|char|bool)[\*]  **<список\_переменных>** ::= <идентификатор>[=<выражение>]{, <идентификатор>[=<выражение>]}  **<идентификатор>** ::= <буква>{(<буква>|<цифра>|\_)}  **<цикл>** ::= (<while-цикл>|<do-while-цикл>|<for-цикл>)  **<while-цикл>** ::= while**"("**<выражение>**")"** <блок>  **<do-while-цикл>** ::= do <блок> while **"("**<выражение>**")"**;  **<for-цикл>** ::= for**"("**<выражение>; <выражение>; <выражение>**")"** <блок>  **<ветвление>** ::= if**"("**<выражение>**")"** <блок>[ else <блок>]  **<литерал>** ::= (<логический\_литерал>|<целая\_константа>|<константа\_с\_плавающей\_точкой>|<символьная\_константа>|<строковый\_литерал>)  **<логический\_литерал>** ::= (true|false)  **<константа\_с\_плавающей\_точкой>** ::= <десятичная\_целая\_константа>[.<десятичная\_целая\_константа>[(e|E)<десятичная\_целая\_константа>]]  **<целая\_константа>** ::= (<десятичная\_целая\_константа>|<восьмеричная\_целая\_константа>|<шестнадцатеричная\_целая\_константа>)  **<десятичная\_целая\_константа>** ::= [(+|-)]цифра{<цифра>}  **<восьмеричная\_целая\_константа>** ::= 0{(0|…|7)}  **<шестнадцатеричная\_целая\_константа>** ::= 0x(0|…|9|a|…|f|A|…F){(0|…|9|a|…|f|A|…F)}  **<цифра>** ::= (0|…|9)  **<символьная\_константа>** ::= '?'  **<буква>** ::= (a|…|z|A|…|Z|)  **<строковый\_литерал>** ::= **"""**{?}**"""** |

Такие последовательности, как комментарии и переносы строк, удаляются на подготовительном этапе лексического анализа. Примеры:

|  |
| --- |
| /\* описание переменных \*/  int a, b, c; // без инициализации  // продолжаем |

После подготовки останется только:

|  |
| --- |
| int a, b, c; |

**2. Базовые типы данных**

В базовые типы входят целочисленный (*int*, от -2147483648 до 2147483647), с плавающей точкой (*double*, значения +/-1.7e+/-308), символьный (*char*), булевый (*bool*, true или false). При декларировании переменной можно тут же инициализировать ее некоторым значением (путем присваивания, без конструктора). Примеры:

|  |
| --- |
| int a = 1, b = 0;  double x =1.5, y;  char c = 'c';  bool yes = true;  y = 15 / x; |

Примеры представления констант:

|  |
| --- |
| const double pi = 3.14159;  const char\* s = "string here"; |

Тип *void* задает пустое множество значений и используется для обозначения типа функций, которые не возвращают результат. При этом нельзя описывать объекты типа *void*.

Целые числа могут задаваться в десятичной, восьмеричной (с «0» в начале) или шестнадцатеричной (начинается с «0x») системах.

**3. Операции и выражения**

Таблица операций

|  |  |  |  |
| --- | --- | --- | --- |
| *Опера-тор* | *Описание* | *Порядок вычисления операндов* | *Пример использования* |
| :: | разрешение области видимости | cлева направо | class-name::member |
| ++ --  ()  .  -> | постфиксный инкремент/декремент  вызов функции  выбор элемента по ссылке  выбор элемента через указатель | lvalue++ lvalue--  expr(expr-list)  obj.member  pointer->member |
| ++ --  + -  !  \*  & | префиксный инкремент/декремент  унарный плюс/минус  логическое НЕ  разыменование  получение адреса | cправа налево | ++lvalue --lvalue  +expr -expr  !expr  \*expr  &lvalue |
| \* /  % | умножение, деление  остаток | слева направо | expr\*expr expr/expr expr%expr |
| + - | сложение, вычитание | expr+expr expr-expr |
| < <=  > >= | сравнение | expr<expr expr<=expr  expr>expr expr>=expr |
| == != | expr==expr expr!=expr |
| && | логическое И | expr&&expr |
| || | логическое ИЛИ | expr||expr |
| =  += -=  \*= /=  %= | прямое присваивание  присваивание по сумме/разности  умножение/деление и присваивание  остаток и присваивание | cправа налево | lvalue=expr  lvalue+=expr lvalue-=expr  lvalue\*=expr lvalue/=expr  lvalue%=expr |
| , | последовательность | cлева направо | expr,expr |

В каждом блоке расположены операторы с одинаковым приоритетом. Операторы в блоке, расположенном выше, имеют более высокий приоритет.

Так как возможность определения пространств имен (*namespace*) отутствует, то оператор разрешения области видимости (::) используется только при описании реализаций методов классов.

**4. Типы данных пользователя и операции над ними**

Возможно создание классов с указанием уровня доступа к его членам (полям (свойствам) и методам (функциям)): либо *public* (есть доступ у методов, не являющихся членами класса), либо *private* (доступ только у членов класса). При декларации свойств запрещена их инициализация.

В качестве примера используем описание класса CRectangle с методом для подсчета площади прямоугольника:

|  |
| --- |
| class CRectangle {  private:  int x, y;  public:  int area;  void set\_values (int, int);  int calc\_area () { area = x \* y; return area; }  };  void CRectangle::set\_values (int a, int b) {  x = a;  y = b;  }  int main () {  CRectangle rect;  rect.set\_values (5, 10);  int calculated\_area = rect.calc\_area(); // 50  int double\_area = rect.area \* 2; // 100  return 0;  } |

**5. Операторы управления**

Для организации управления применяется оператор *if* и циклы *while, do‑while, for*.

Общие формы условных структур:

*if (условие) оператор*

*if (условие) оператор1 else оператор2*

*if (условие1) оператор1 else if (условие2) оператор2 else оператор3*

В качестве примера можно привести такой участок программы:

|  |
| --- |
| int x = 10;  if (x < 15) x = 15; |

Структура циклов:

*while (условие) оператор*

*do оператор while (условие);*

*for (оператор1; условие; оператор2) оператор3*

Цикл *while*: если *условие* истинно, выполняется *оператор* и цикл повторяется. Иначе выход из цикла.

Цикл *do-while*: выполняется *оператор* и, если *условие* истинно, цикл повторяется. Иначе выход из цикла.

Цикл *for*: выполняется *оператор1* (инициализация). Затем сам цикл: если *условие* истинно, то выполняется *оператор3*, а затем *оператор2* (обычно регулирующий шаг цикла) и цикл повторяется. Иначе выход из цикла.

Примеры:

|  |
| --- |
| int n = 5, a = 1, b = 1;  while (n>0) {  a \*= b;  b++;  --n;  }  // now a = -30  a = 0;  do {  a -= b;  n++;  } while (n != 5);  a = 0;  for (n = 10; n > 0; n--) {  a = b \* b;  }  // now a = 36 |

В теле цикла могут использоваться операторы перехода, такие как *break* и *continue*: *break* позволяет мгновенно покинуть цикл, даже если условие его окончания не выполнено; *continue* указывает на то, что оставшаяся часть тела цикла на текущей итерации будет пропущена (как если бы был достигнут конец тела цикла), что приведет к переходу на следующую итерацию.

Примеры:

|  |
| --- |
| int n, a = 0;  for (n = 10; n > 0; n--) {  a += n;  if (n == 3) {  break;  }  }  // now a = 52  a = 0;  for (n = 10; n > 0; n--) {  if (n == 5) continue;  a += n;  }  // now a = 50 |

**6. Управление данными**

Способ передачи параметров в вызовах процедур и функций — по значению (значения переменных копируются в параметры функции) и по ссылке (переменные ассоциируются с параметрами функции, т.о. функция может работать с внешней для нее переменной). Для первого случая:

|  |
| --- |
| int add(int a, int b) { return (a + b); }  int x = 5, y = 10, z;  z = add(x, y);  // z = 15 |

Если же передавать параметры по ссылке:

|  |
| --- |
| void to\_double(int& a, int& b) { a \*= 2; b \*= 2; }  int x = 5, y = 10;  to\_double(x, y);  // x = 10, y = 20; |

**7. Структура программы**

Программа представляет собой последовательность описаний — набор деклараций классов, функций, а также основную функцию *int main()* (тип возвращаемого значения только *int*), с которой и начинается выполнение программы. Возможность локальной декларации классов внутри блоков отсутствует. В классах допускается создание публично доступных и приватных (частных) свойств и методов. Методы классов, как и функции, представляют собой последовательность операторов и определений переменных.