Инструкция

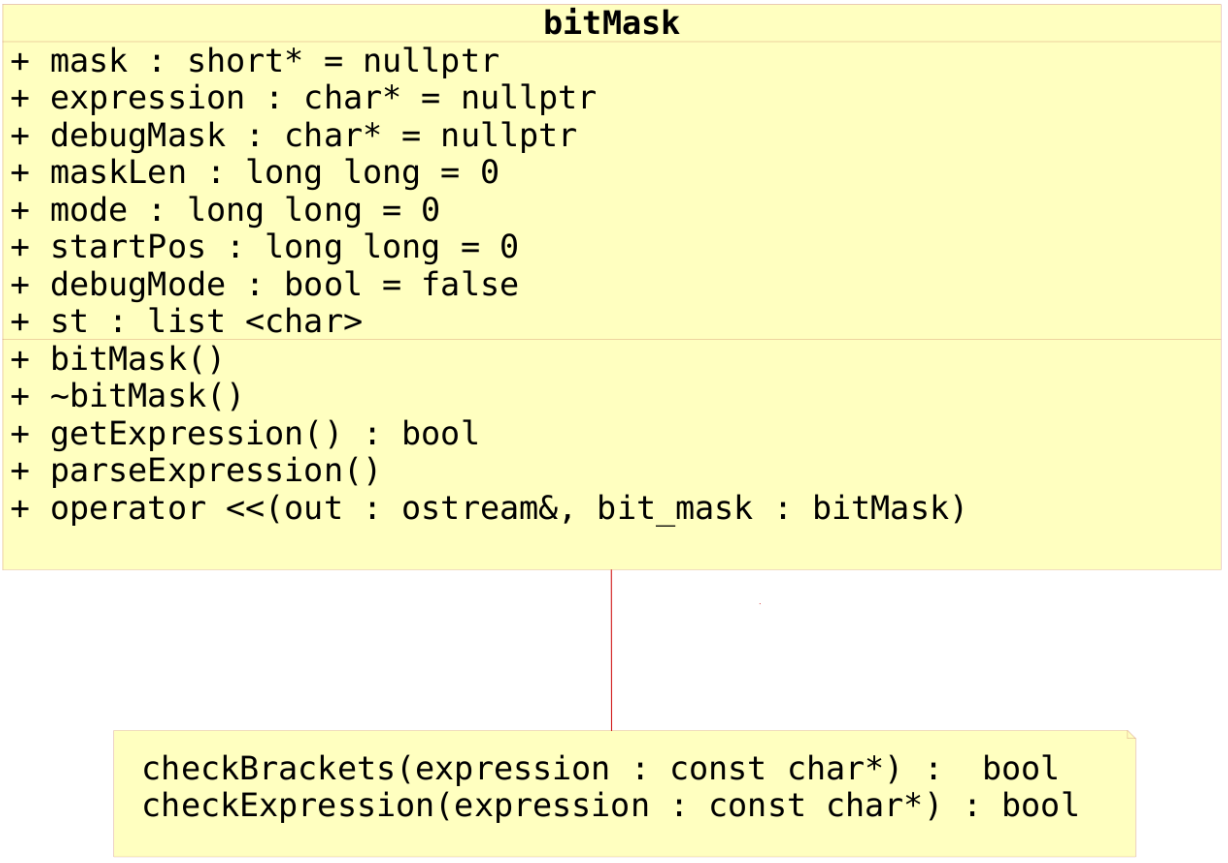
Парсер битовой маски для symCC

Содержание

Описание

Парсер битовой маски - это программный инструмент, предназначенный для анализа и обработки битовых масок. Битовая маска представляет собой последовательность битов, часто используемую для кодирования информации или представления набора флагов. Парсер позволяет создать пользователю битовую маску, удовлетворяющую определенным условиям. Также он поддерживает два режима обработки входных данных битовой маски. Написан для внедрения в исходный код symCC, дабы повысить эффективность последнего.

UML



Структура

- class **bitMask** - класс битовой маски
    - Поля:
        - short* **mask** - битовая маска;
        - char* **expression** - входное выражение, содержащее правила для построения битовой маски и служебные данные;
        - char* **debugMask** - входное выражение, содержащее информацию о включении/отключении режима отладочного вывода;
        - long long **maskLen** - длина **expression**;
        - long long **mode** - режим работы парсера (см. п. “Использование”);
        - long long **startPos** - индекс, с которого в выражении **expression** начинается битовая маска;
        - bool **debugMode** - флаг отладочного вывода;
        - list <char> **st** - стэк, использующийся для построения маски;
    - Методы:
        - bool **getExpression()** - подготовка **expression** для парсинга, получение служебной информации;
        - void **parseExpression()** - парсинг из **expression** и получение битовой маски **mask**;
- Дополнительные функции:
    - bool **checkBrackets**(const char* *expression*) - проверка **expression** на правильную скобочную последовательность для режима работы **mode** = *0* (см. п. “Использование”);
    - bool **checkExpression**(const char* *expression*) - проверка expression на использование только заданного алфавита ({', '}', '-', ',', '_').

Использование

Данный парсер использует для своей работы следующие переменные окружения:

- **SYMCC_MASK** - входное выражение, содержащее правила для построения битовой маски и служебные данные;
- **SYMCC_MASK_DEBUG** - входное выражение, содержащее информацию о включении/отключении режима отладочного вывода.
    - Может принимать следующие значения:
        - 1
        - 0
        - true
        - false
        - yes
        - no

Любое вводимое выражение должно начинаться с указания служебной информации:

1) **Длина битовой маски** - число, за которым следует разделитель “:”, например *“12345:”, т.е. “длина_маски:”;*

2) После длины битовой маски следует **режим работы парсера** - число, после которого ставится разделитель ":", например *"...:1", т.е. "режим:".*

Реализовано два режима работы парсера:

- mode = 0:
  В данном режиме выражение составляется по следующим правилам:
  1) Изначально битовая маска, указанной ранее длины, заполняется "0";
  2) В фигурных скобках указывается необходимая пользователю последовательность "0" и "1", которую при необходимости можно повторить несколько раз, например *"...:{10011}"*, или *"...:{1}"*, или *"...:{10101000001001}"* и т.д.
  3) Заполнение битовой маски происходит слева направо, т.е. если дано выражение *"...:{101}{1101}{0011}..."*, то битовая маска будет выглядеть таким образом:

1	0	1	1	1	0	1	0	0	1	1	…

  4) Последовательности в фигурных скобках можно многократно повторять, используя запись с *"_"*, например для выражения *"...:{101}_3"* битовая маска примет вид:

1	0	1	1	0	1	1	0	1	...

  5) Если длина получающейся битовой маски будет превосходить, указанную длину, то программа будет завершена с соответствующим сообщением об ошибке.
  6) Не используется совместно с mode = 1.

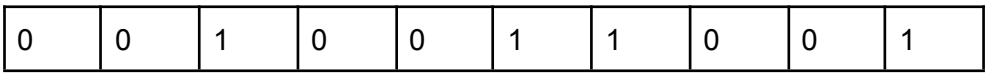
- mode = 1:
  1) Изначально битовая маска, указанной ранее длины, заполняется "0";
  2) В данном режиме указываются индексы, в которых необходимо разместить "1", с помощью "**-**", "**,**", где первое - это диапазон (концы включительно), а второе - перечисление, например: для выражения "...:5-7, 9, 11-12" битовая маска будет:

0	0	0	0	0	1	1	1	0	1	0	1	1	0	...

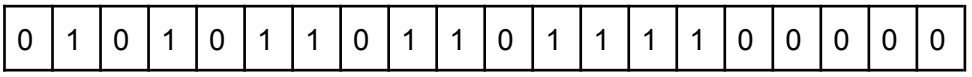
  3) Если запрашивается недопустимый индекс, то программа будет завершена с соответствующим сообщением об ошибке.
  4) Не используется совместно с mode = 0.
- Прочее:
  1) cout << объект_класса_bitMask работает только при включенном отладочном выводе, например SYMCC_MASK_DEBUG = 1.

Примеры

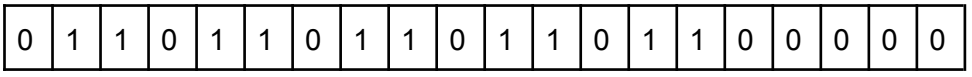
1. SYMCC_MASK = “10:0:{001}_2{1001}”



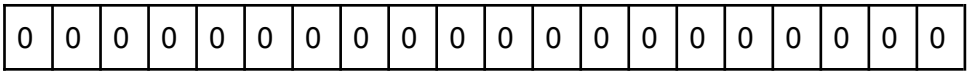
2. SYMCC_MASK = “20:0:{01}_3{101}_2{111}_1”



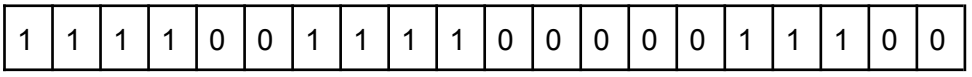
3. SYMCC_MASK = “20:0:{01}{101}_3{1011}”



4. SYMCC_MASK = “5:0:{01}{101}_3{1011}”
   Ошибка (длина описанная маск больше, чем указанная).
5. SYMCC_MASK = “20:3:{01}{101}_3{1011}”
   Ошибка (неизвестный режим работы парсера).
6. SYMCC_MASK = “20:0:”



7. SYMCC_MASK = “20:3:{01}{{101}_3{1011}”
   Ошибка (неверная скобочная последовательность).
8. SYMCC_MASK = “20:1:”
   Ошибка (недопустимое значение индекса).
9. SYMCC_MASK = “14:1:0,5,4,{01}”
   Ошибка (mode = 0 и mode = 1 несовместимы).
10. SYMCC_MASK="20:1:0-3,6,6-9,15-17,2"



11. SYMCC_MASK="20:1:0-10,"
    Ошибка (недопустимое написание выражения).
12. SYMCC_MASK="20:1:1-10,12-,"
    Ошибка (недопустимое написание выражения).