# 2 字句解析

♣ 「字句解析」とは [　　] か.

♣ 字句解析の [　　] は.

## 2.1 字句解析とは

- 字句解析の仕事
  1. 文字列中の字句 (単語, トークン) を認識する
  2. それぞれの字句の [　　] がわかるようにする
  3. 字句を必要な [　　] に変換する
- 字句の種類 (C 言語の場合 6 種類)
  1. [　　] ( [　　] )

     変数や関数などの [　　] を表す

     `main`, `a123`, `Max_Value` のように, [　　] と [　　] からなり, [　　] で始まる.

     [　　] 文字と [　　] 文字は区別される. [　] (アンダーライン) は英文字とみなされる.
  2. [　　] (keywords)

     決まった意味で用いる語

     C 言語では, 識別子として用いることはできない. このような語を特に [　　] 語 ( [　　] word) という.

     C 言語 (ISO-C90) では [　　] 個[1]

     ```
     auto, break, case, char, const, continue, default, do, double,
     else, enum, extern, float, for, goto, if, int, long, register,
     return, short, signed, sizeof, static, struct, switch,
     typedef, union, unsigned, void, volatile, while
     ```
  3. [　　] (constants)
     - ある特定の型の定数を表すもの
     - [　　] 定数: `123`( [　　] 定数), `0123` ( [　　] 定数), `0xae86`( [　　] 定数)
     - [　　] 定数: `'a'`, `'0'`, `'\n'` ( [　　] ), `'\t'` ( [　　] ) など
     - [　　] 定数: `1.23e6` ( [　　] ), `2.34e-12` ( [　　] ) など

[1] C 言語の新しい規格 C99 では, `inline`, `restrict`, `_Bool`, `_Complex`, `_Imaginary` の 5 つが追加されている.

4. [ ] 定数 ( [ ] )

   2 重引用符 (") で囲まれた記号列 ("This is a pen."など)

5. [ ] (operators)

   +, =, >>, ++, += など演算を表す記号

6. [ ] ( [ ] , [ ] )

   ;, ,, ( などの区切り記号

7. (番外) [ ]

   [ ] , [ ] , [ ] , [ ] など, 識別子を区切る以外は無視される

   コメント

   [ ] と [ ] で区切られた文字列で, [ ] を含んではならない

   [ ] から改行記号までの文字列

- 字句解析ルーチン
  - 字句解析の 2 形態
    1. プログラム [ ] に対して字句解析を行なう.

       これに続いて [ ] 解析を行なう
    2. [ ] づつ解析しながら [ ] 解析を行なう

       [ ] 解析から一々 [ ] 解析を呼び出す
  - いずれにしても, 一字句を切り出す「字句解析ルーチン」を考えるとよい
    1. 呼び出される毎に次の [ ] を切り出す
    2. 指定された変数に, その字句の種類, 値などがセットされる

## 2.2 字句解析の方法

- プログラムは [ ]

```
int fx()
{
  int ab=23;
}
```

というプログラムは, コンパイラから見ると

| i | n | t |  | f | x | ( | ) | 改 | { | 改 |  |  | i | n | t |  | a | b | = | 2 | 3 | ; | 改 | } | 改 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

という文字列に過ぎない (「改」は改行文字).

ここから字句を [ ] するのが字句解析の役割

- 具体的な字句解析ルーチンの仕事
  - 呼び出されると 1 つのトークンを切り出す.

– 次の [　　] だけが見えている (これを c とする).

– [　　] は読み飛ばす.

– 切り出したトークンを [　　] (token とする) に格納する.

– トークンの [　　] を指定された変数 (type とする) にセットする.

- 解析例

★ 1 回目の呼び出し

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| c | i | | | | | | | | |
| token | \0 | | | | | | | | |
| type | ? | | | | | | | | |

1. 'i' ( [　　] ) で始まる字句は [　　] か [　　] → 'i' を token に格納.

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| c | n | | | | | | | | |
| token | i | \0 | | | | | | | |
| type | ? | | | | | | | | |

2. 次の文字が 'n' ということは, これはトークンの続き. → 'n' を token に追加

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| c | t | | | | | | | | |
| token | i | n | \0 | | | | | | |
| type | ? | | | | | | | | |

3. 同様に次の文字 't' を token に追加

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| c | | | | | | | | | |
| token | i | n | t | \0 | | | | | |
| type | ? | | | | | | | | |

4. 次の文字 = ' ' なので トークンは終りとわかる.

   現在の token "int" は [　　] である.

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| c | | | | | | | | | |
| token | i | n | t | \0 | | | | | |
| type | キーワード | | | | | | | | |

これで 1 トークンの切り出しが完了

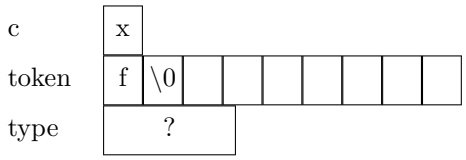
★ 2 回目の呼び出し

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| c | | | | | | | | | |
| token | \0 | | | | | | | | |
| type | ? | | | | | | | | |

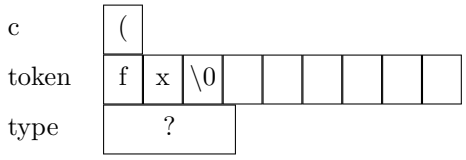
1. 空白を読み飛ばす

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| c | f | | | | | | | | |
| token | \0 | | | | | | | | |
| type | ? | | | | | | | | |

2. c = 'f' (アルファベット) で始まるので, 識別子かキーワード→ 'f' を token に格納.

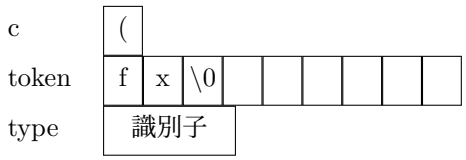



3. 続く文字 'x' を token に追加



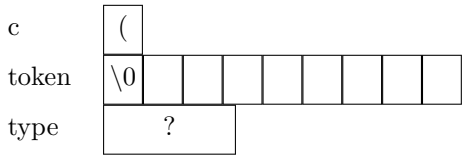

4. 次の文字 = '(' なので トークンは終り.

現在の token "fx" は [　　] ではないので, [　　] と判定.

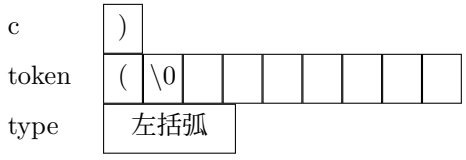



これで 1 トークンの切り出しが完了

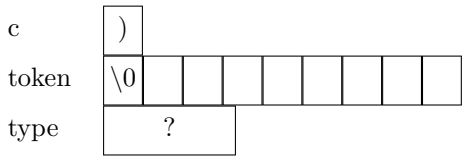
★ 3 回目の呼び出し



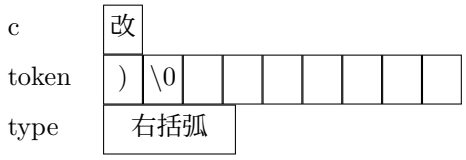

1. c = '(' を見た時点で, これが一つのトークンであると判定できる. '(' をトークンに入れ, タイプをセットして終了.



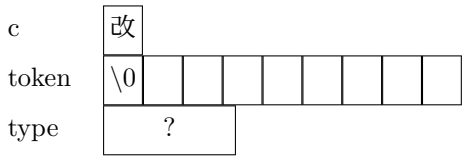

★ 4 回目の呼び出し




1. c = ')' を見た時点で, これが一つのトークンであると判定できる. ')' をトークンに入れ, タイプをセットして終了.




★ 5 回目の呼び出し



1. 改行記号を読み飛ばす

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| c | { | | | | | | | |
| token | \0 | | | | | | | |
| type | ? | | | | | | | |

2. c = '{' を見た時点で, これが一つのトークンであると判定できる. '{' をトークンに入れ, タイプをセットして終了.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| c | 改 | | | | | | | |
| token | { | \0 | | | | | | |
| type | 左ブレース | | | | | | | |

- 字句解析ルーチンのおおよその構造

[List 2.1]

```
void 字句解析 () {
  空白の読み飛ばし;
  if      (先頭がアルファベット) { [　　] , [　　] の処理; }
  else if (先頭が数字) { [　　] 定数, [　　] 定数の処理; }
  else if (先頭が ' ) { [　　] 定数の処理; }
  else if (先頭が " ) { [　　] 定数の処理; }
  else if (先頭が記号) { [　　] および [　　] の処理; }
  else { エラー; }
}
```

Nagisa ISHIURA