1 break 文

1.1 学習のポイント

switch case 文またはループの途中から外に脱出するためのスマートな制御文 break について学びます。

break は switch またはループから強制的に脱出するときに使います。break はすでに switch〜case 文のところで説明していますので、ここでは for、while、do〜while などのループ内から抜ける場合の例を説明します。

ループ中の break が実行されると最も内側のループから抜けます。break により何重ものネスト (ループの中のループが入る構造) から一気に抜けることはできません。

1.2 書式

```
while（式）{
      文 1
  if（ある条件なら）
      break;
        •
      文 2
}
```

ループ内の途中からループ外に脱出したい場合、BASIC や FORTRAN では goto を使わなければなりませんでしたが、C では、break によりスマートに (飛び先のラベルが不要のため) ループ外へ脱出することができます。

配列を探索する時、プログラムは配列の先頭から最後まで探索しますが、大抵の場合、探索の途中で解が見つかります。この様なときは、break 文を使ってループから抜け出すと処理が速くなります。

1.3 reidai32.c

```
/*
  reidai32.c
  二次元配列の各要素を表示するときに、各行要素に 0 が現れたら、その行の表示
を中止するようにしなさい。
  C 言語 132 頁
```

```
*/

#include <stdio.h>

int main()
{
  static int a[4][5] = {{ 1, 2, 3, 4, 0},
                        { 2, 2, 0, 0, 0},
                        { 3, 4, 5, 0, 0},
                        { 5, 0, 0, 0, 0}};
  int       j, k;

  for(j = 0; j < 4; j++){
    for(k = 0; k < 5; k++){
      if(a[j][k] == 0){
        break;
      }
      printf("%d ", a[j][k]);
    }
    printf("\n");
  }
  return 0;
}
```

2 AI による大規模データ処理入門

人口知能の研究は、1950 年代から始まりました。当時の電子計算機は、開発されたばかりで主に数値計算を行なっていました。次に人間とチェスをするような人口知能の研究がはじまりました。コンピュータ・チェスプレーヤーの実装のひとつの考え方が探索アルゴリズムです。当初、人口知能の実現可能性は楽観視されていましたが、チェスの指し手順にそって、すべての盤面の状態を主記憶に記憶させておいて、手順を評価していくことが不可能であることが分かって来ました。

第 2 章は、迷路を使って探索プログラムの動作を説明されています。最初から探索木で説明することが難しいようです。ここからは、ひとまず迷路を離れて、探索木で「width.c」プログラムの動作を一通り解説することにします。

前回は、探索木がどのように、記憶、表現されているかを説明しました。プログラムでは探索木が以下の様に配列として記憶されています。

```
/* 接続関係の情報 */
  int tree[][5] = {
    {1, 2, 3, 5},
    {2, 4},
    {3, 6, 7},
    {5, 4, 8},
    {6, 5, 7},
    {8, 7, 999},
    {0}
  };
```

探索の準備は、initlist() 関数で行なわれています。initlist() 関数は、これから探索する節点数を openlistep に保存します。

次は、オープンリストとクローズドリストについて説明します。オープンリストは、これから探索することができる節点を保存しているリストです。クローズリストは、探索が終わった節点を保存しています。

オープンリストは、下記の様な構造体で定義されていて、クローズドリストも同じです。

```
struct node{
  int nodeid;
  int parentid;
};

struct node openlist[LIMITL];
int         openlistep = 0;
```

initlist() 関数実行後の各変数の値は、次の様になっています。

```
openlist[0].nodeid   : 1
openlist[0].parentid : 0
openlistep   : 1
closedlistep : 0
```

printlist() 関数は、オープンリストとクロースドリストの内容を表示します。このとき、openlistep の値は、1 なので openlist[0] の内容を表示します。また、closedlistep の値は、0 なので closedlist の内容はを表示しません。

次に、プログラムは while 文の中に入ります。この while 文は常に実行されます。

次は、openlistep の値を判定します。最初は、openlistep の値は、1 なのでこの if 文は実行されません。

次の if 文で、openlist[0].nodeid と GOAL(999) の値を判定しています。このとき、openlist[0].nodeid は 1 なので、この if 文は実行されません。

次は、expand(openlist[0].nodeid) 関数の動きを見てみます。

expand() 関数の展開とは、次に探索する節点を決めることです。

展開とは、迷路の分岐点で次に進む地点を決めたり、ハイパーテキストの連鎖の中で次のリンク先を探したり、将棋などでは、次に着手できる場所探して駒を動かしたりする作業です。

expand() 関数は、openlist[0].nodeid を引数とします。最初の id は、1 です。また、openlistep と closedlistep の値を使います。

先ほど示した変数の値は、下記の様になっています。

```
oepnlist[0].nodeid   : 1
oepnlist[0].parentid : 0
openlistep   : 1
closedlistep : 0
```

while 文は、tree[i][0] を判定して、tree[i][0] が 0 になるまで tree[i][0] を調べます。

次の if 文では、tree[0][0] が、1 で id の値と一致するので for 文に入ります。

次は、tree[0][1] から tree[0][3] までと、id が重複していないか、2 回探索していないか、判定します。

id は 1、tree[0][1] から tree[0][3] は、2,3,5 なので、この if 文は実行されません。

次の if 文は、closedlistep は、0 なので実行されません。

次は、openlist に節点を下記の様に書き込みます。

```
openlist[1].nodeid   : 2
openlist[1].parentid : 1

openlist[2].nodeid   : 3
openlist[2].parentid : 1

openlist[3].nodeid   : 5
openlist[3].parentid : 1
```

このとき、openlistep は、4 になっています。

次の movetofirst() 関数では、openlist[0] を closedlist[closedlistep] に、複写しています。この時、closedlistep は、0 なので、openlist[0] を closedlist[0] に、複写することになり、

```
closedlist[0].nodeid   : 1
closedlist[0].parentid : 0
closedlistep : 1
```

になります。

次の、removefirst() 関数では、openlist の内容を下記の様に書き換えます。

```
oepnlist[0].nodeid   : 1
oepnlist[0].parentid : 0

openlist[1].nodeid   : 2
openlist[1].parentid : 1

openlist[2].nodeid   : 3
openlist[2].parentid : 1

openlist[3].nodeid   : 5
openlist[3].parentid : 1
```

を

```
openlist[0].nodeid   : 2
openlist[0].parentid : 1
```

```
openlist[1].nodeid   : 3
openlist[1].parentid : 1

openlist[2].nodeid   : 5
openlist[2].parentid : 1

openlistep : 3
```

に編集します。

次に printlist() 関数を実行します。

これで、初回の繰り返しが終了しました。