1 else if 文

1.1 学習のポイント

switch case 文と同様な複数方向分岐を行なう else if 文について学びます。

switch case 文の < 式 > に指定できる型は、char、int、enum だけです。これ以外の型の式を条件判定したい場合は else if 文を使います。

else if 文というのは、if else 文の else 節に if 文がネストしたもので、一般形は次のように書きます。

```
if (式 1)
     文 1
else if (式 2)
     文 2
       •
       •
       •
else if (式 n)
     文 n
else
     文
```

式 1 から式 n までが上から評価され、一致したところを実行し else if 文から抜けます。一致する条件がなかったときは else 節が実行されますが、else 節に書くものが無ければ省略できます。

文が複数のときは {} で囲みます。

1.2 reidai31.c

```
/*
  reidai31.c
  生徒の得点が point[] にあったとき、優、良、可、不可を判定して表示しなさい。
  C 言語 130 頁
*/

#include <stdio.h>
```

```
int main()
{
  int point[] = {56, 76, 55, 80, 90, 86, 70, -9999};
  int       i = 0;

  while(point [i] != -9999){
    printf("no %2d : %3d ", i + 1, point[i]);
    if(point[i] > 85)
      printf("優\n");
    else if(point[i] > 70)
      printf("良\n");
    else if(point[i] >= 60)
      printf("可\n");
    else
      printf("不可\n");
    i++;
  }
  return 0;
}
```

2 AI による大規模データ処理入門

2.1 探索手法のプログラムの説明

問題空間の木構造を 2 次元配列で表現しています。例えば、width.c では下記の様に表現しています。

```
/* 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}
  };
```

ここでの、int tree[][5] は、int tree[7][5] と同じです。間違って、int tree[6][5] と書くとエラーになります。

```
/*
  エラーになります。
  er.c
*/

#include <stdio.h>

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

  printf("er\n");

  return 0;
}
```

先頭の配列 tree[0][5] = {1, 2, 3, 5}; は、1 の地点から 2、3、5 の地点に進むことができる十字路を表しています。

次の配列 tree[1][5] = {2, 4}; は、2 の地点から 4 の地点に進むことができる一方通行路を表しています。

次の配列 tree[2][5] = {3, 6, 7}; は、3 の地点から 6、7 の地点に進むことができる T 字路を表しています。

これで、迷路と木構造が関連付けられました。

将棋、囲碁ゲームなど探索すべき状態の数が膨大になる現象を組み合わせ爆発 (combinatorial explosion) とよびます。検索すべき状態数が増えてくると、探索木全体をコンピュータの記憶装置に格納するのは不可能で、計算時間も莫大なものになります。

最初の探索の例は、力ずくの探索手法です。すべての経路を通って、すべての状態を探索できるか確認することが目的です。

それでは、横型探索を始めましょう。

このプログラムの最初の仕事は、開始地点を覚えることです。

それは、initlist() 関数でしています。

width1.c は、横型探索プログラムに開始地点を覚えさせるプログラムです。

```
/*
  横型探索プログラム
  width1.c
*/

#include <stdio.h>

#define LIMITL 256
#define START 1

struct node{
  int nodeid;
  int parentid;
};

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

void initlist();

int main()
{
  initlist();

  return 0;
}

void initlist()
```

```
{
  openlist[0].nodeid = START;
  ++openlistep;
}
```

これでは、上手く動作しているか分からないので、width2.c でそれを表示して確認しましょう。

```
/*
  横型探索プログラム
  width2.c
*/

#include <stdio.h>

#define LIMITL 256
#define START 1

struct node{
  int nodeid;
  int parentid;
};

struct node openlist[LIMITL];
int         openlistep = 0;
struct node closedlist[LIMITL];
int         closedlistep = 0;

void initlist();
void printlist();

int main()
{
  initlist();
  printlist();

  return 0;
}

void printlist()
```

```
{
  int i;

  printf("\nオープンリスト    ");
  for(i = 0; i < openlistep; ++i){
    printf("%d[%d],", openlist[i].nodeid,
                      openlist[i].parentid);
  }
  printf("\n");

  printf("クロースドリスト  ");
  for(i = 0; i < closedlistep; ++i){
    printf("%d[%d],", closedlist[i].nodeid,
                      closedlist[i].parentid);
  }
  printf("\n");
}

void initlist()
{
  openlist[0].nodeid = START;
  ++openlistep;
}
```