1　参照による呼び出し (call by reference)

1.1　学習のポイント

引数を介して関数から呼び出し元に値を返す方法を学びます。

1.2　参照による呼び出し (call by reference)

関数はその構造からして、戻り値を一つしか持てません。しかし 2 つ以上の値を返したい場合もよくあります。このような場合は、関数の戻り値として 2 つの値を返すのではなく、ポインタを用いた引数渡しを行い、呼ばれた関数での結果を引数を介して呼び出し元に返してやります。これを　call by reference (参照による呼び出し) と呼びます。
参照による呼び出し (call by reference) では、引数を介して呼び出し元と関数側とで双方向のデータ授受を行なうことが、できますが不注意により、呼ばれた関数側で、呼び出し元の引数の値を変化させていまう危険性があります。
「C 言語」(河西朝雄著　ナツメ社)108 頁

1.3　例題 24

```
/*
  reidai24.c
  データ a,b の加算、減算を求める関数 tasuhiku() を作りなさい。
*/

#include <stdio.h>

void tasuhiku(int w, int x, int *y, int *z)
{
  *y = w + x;
  *z = w - x;
}

int main()
{
```

```
    int a, b, c, d;

    printf("data ? ");
    scanf("%d %d", &a, &b);

    tasuhiku(a, b, &c, &d);

    printf("%d + %d = %d\n", a, b, c);
    printf("%d - %d = %d\n", a, b, d);

    return 0;
}
```

「C 言語」(河西朝雄著　ナツメ社)109 頁

2　AI による大規模データ処理入門

2.1　力ずくの探索手法

　探索技術の基礎となるのは、探索対象となる状態空間をもれなく順に探索していく力ずくの探索手法です。ここでは、探索とは何かを概観した上で、力ずくの探索手法である横型探索と縦型探索のアルゴリズムを紹介します。
AI による大規模データ処理入門　24 頁　小高　知宏著　オーム社

2.2　横型探索プログラム

```
/*
  横型探索プログラム
  width.c
*/

#include <stdio.h>

#define LIMITL 256
#define TRUE 1
#define START 1
```

```
#define GOAL 999

struct node{
  int nodeid;
  int parentid;
};

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

void initlist();
void printlist();
void expand(int id);
void movetofirst();
void removefirst();
int  check(int id);
void printroute(int id);

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

  while(TRUE){
    if(openlistep == 0){
      printf("ゴールは見つかりませんでした\n");
      break;
    }

    if(openlist[0].nodeid == GOAL){
      printf("\n ゴールを見つけました\n");
      printf("%d[%d]", openlist[0].nodeid,
                       openlist[0].parentid);
      printroute(openlist[0].parentid);
      break;
    }

    expand(openlist[0].nodeid);
```

```
    movetofirst();

    printlist();
  }

  return 0;
}

void printroute(int id)
{
  int i;

  for(i = 0; i < closedlistep; ++i){
    if(closedlist[i].nodeid == id){
      printf("<-%d[%d]", closedlist[i].nodeid,
                         closedlist[i].parentid);
      break;
    }
  }

  if(closedlist[i].parentid != 0){
    printroute(closedlist[i].parentid);
  }

  printf("\n");
}

int check(int id)
{
  int i;
  int res = 0;

  for(i = 0; i < openlistep; ++i){
    if(openlist[i].nodeid == id){
      res = TRUE;
    }
  }

  for(i = 0; i < closedlistep; ++i){
```

```
    if(closedlist[i].nodeid == id){
      res = TRUE;
    }
  }

  return res;
}

void removefirst()
{
  int i;
  for(i = 0; i < openlistep; ++i){
    openlist[i] = openlist[i + 1];
  }
  --openlistep;
}

void movetofirst()
{
  closedlist[closedlistep++] = openlist[0];
  removefirst();
}

void expand(int id)
{
  int tree[][5] = {
    {1, 2, 3, 5},
    {2, 4},
    {3, 6, 7},
    {5, 4, 8},
    {6, 5, 7},
    {8, 7, 999},
    {0}
  };

  int i = 0;
  int j;

  while(tree[i][0] != 0){
    if(tree[i][0] == id){
```

```
      for(j = 1; tree[i][j] != 0; ++j){

        if(check(tree[i][j]) != TRUE){
          openlist[openlistep].nodeid = tree[i][j];
          openlist[openlistep++].parentid = id;
        }

      }
      break;
    }
    ++i;
  }
}

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

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");
}
```

2.3 使い方

```
./width
```