1 switch case 文

1.1 学習のポイント

複数方向への分岐を行なう制御文 switch case 文を学びます。

if else 文は、2 方向への分岐でしたが、複数方向への多方向分岐を判断する文があります。その代表的なものに switch〜case 文があります。

switch の () の式と、case 式 1〜式 n が比較され、一致する case に書かれている文を実行し、break で switch 文を抜けます。もし break がなければ、次の case に書かれている文に実行が進みます。

また一致する case がなければ default が実行されます。default に書くものがなければ省略しても構いません。

switch 文の式に書ける型は char、int、enum です。

1.2 reidai30.c

```
/*
  reidai30.c
  1 文字入力し、それが'b' または'B' なら"basic"、'f' または'F' なら"fortran"、'p' また　は'P' なら"pascal"、これら以外なら"???"と表示しなさい。
  C 言語 128 頁
*/

#include <stdio.h>

int main()
{
  int c;

  c = getchar();

  switch(c){
    case 'b':
    case 'B':
      printf("basic\n");
      break;
```

```
    case 'f':
    case 'F':
      printf("fortran\n");
      break;

    case 'p':
    case 'P':
      printf("pascal\n");
      break;

    default:
      printf("???\n");
  }

  return 0;
}
```

2　AI による大規模データ処理入門

2.1　A アルゴリズム

最良優先探索では次に選択すべき方向をヒューリスティック関数 h により選び、最適経路探索では、探索経路の積算距離 v により最適化を図りました。これらの探索手法ではそれぞれ異なる点に着目して探索を制御していますから、両者を一つに融合することが可能です。探索過程の経路の積算値 v と、次の探索箇所についての推定値 h の両方を用いる探索アルゴリズムを A アルゴリズムと呼びます。

A アルゴリズムでは、最良優先探索におけるヒューリスティック関数 h の値と、最適経路探索における積算値 v の両方から求まる値を、評価値 f として用います。AI による大規模データ処理入門　92 頁　小高　知宏著　オーム社

2.2　最良経路探索プログラム

```
/*
  A-アルゴリズム
  a.c
*/
```

```
#include <stdio.h>

#define LIMITL 256
#define TRUE 1
#define START 1
#define GOAL 999

struct node{
  int    nodeid;
  int    parentid;
  double value;
};

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

double h[] = {0, 5.7, 4.5, 2.8, 4.1, 2, 4, 4, 0};

void initlist();
void printlist();
void expand(struct node cnode);
void movetofirst();
void removefirst();
int  ccheck(struct node cnode, int id, int v);
int  ocheck(struct node cnode, int id, int v);
void printroute(int id);
void sortopenlist();
int  cmp(const void *a, const void *b);

int main()
{
  struct node currentnode;

  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;
    }

    currentnode = openlist[0];
    removefirst();
    expand(currentnode);

    closedlist[closedlistep++] = currentnode;

    sortopenlist();

    printlist();
  }

  return 0;
}

void sortopenlist()
{
  qsort(openlist, openlistep, sizeof(struct node), cmp);
}

int cmp(const void *a, const void *b)
{
  struct node *x, *y;

  x = (struct node *)a;
  y = (struct node *)b;
  if((x->value) < (y->value)){
    return -1;
```

```
  }else{
    if((x->value) > (y->value)){
      return 1;
    }else{
     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 ccheck(struct node cnode, int id, int v)
{
  int i,j;

  for(i = 0; i < closedlistep; ++i)
    if(closedlist[i].nodeid == id){
      if(closedlist[i].value < cnode.value + v){
        return TRUE;
      }
      else{
        for(j = i; j < closedlistep; ++j){
          closedlist[j] = closedlist[j + 1];
```

```
        }
        --closedlistep;
        }
    }
  return 0;
}

int ocheck(struct node cnode, int id, int v)
{
  int i,j;

  for(i = 0; i < openlistep; ++i)
    if(openlist[i].nodeid == id){
      if(openlist[i].value < cnode.value + v){
        return TRUE;
      }else{
        for(j = i; j < openlistep; ++j){
          openlist[j] = openlist[j + 1];
        }
        --openlistep;
        }
      }
  return 0;
}

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(struct node cnode)
{
  double tree[][10] = {
    {1, 2, 2, 4, 3},
    {2, 3, 2, 6, 2},
    {3, 2, 2, 5, 2, 999, 2.8},
    {4, 3, 2.8},
    {5, 7, 2, 999, 2},
    {0}
  };

  int i = 0;
  int j;

  while(tree[i][0] != 0){
    if(tree[i][0] == cnode.nodeid){
      for(j = 1; tree[i][j] != 0; j+=2){

        if(ocheck(cnode, tree[i][j],
                         tree[i][j+1] + h[(int)tree[i][j]]) != TRUE
          && (ccheck(cnode, tree[i][j],
                     tree[i][j + 1] + h[(int)tree[i][j]]) != TRUE)){
            openlist[openlistep].nodeid = tree[i][j];
            openlist[openlistep].parentid = cnode.nodeid;
            openlist[openlistep++].value = cnode.value + tree[i][j + 1]
            + h[(int)tree[i][j]];
        }
      }
      break;
    }
    ++i;
  }
}

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

```
void printlist()
{
  int i;

  printf("\n オープンリスト    ");
  for(i = 0; i < openlistep; ++i){
    printf("%d[%d, %.1lf],", openlist[i].nodeid,
                             openlist[i].parentid,
                             openlist[i].value);
  }
  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

```
./a
```