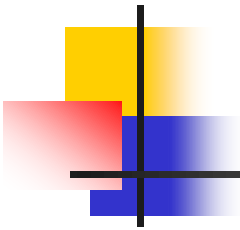
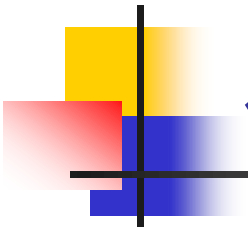
プログラミング及び演習
第9回 列挙型・構造体
(教科書第11章) (2015/06/19)

講義担当
情報連携統括本部情報戦略室
森　健策



本日の講義・演習の内容

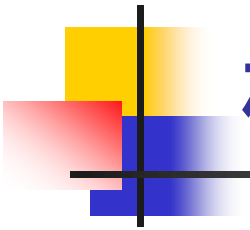
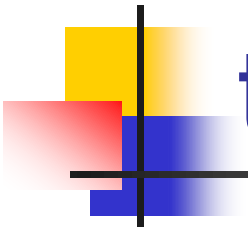
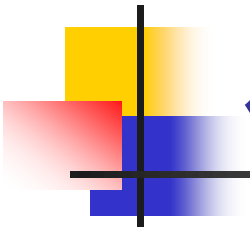
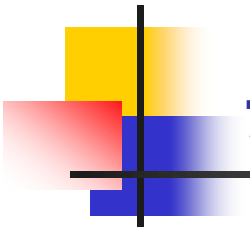
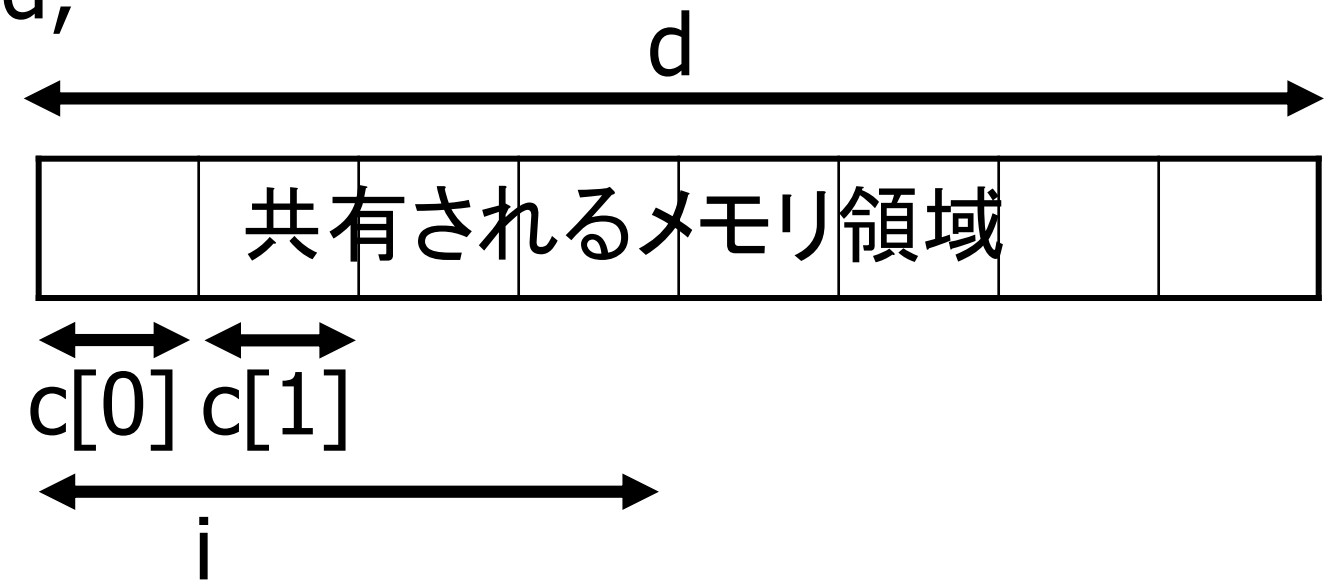
- 列挙型/構造体
  - 第11章
- 講義・演習ホームページ
  - http://www.newves.org/~mori/15Programming
- ところで,
  - だんだん難しくなってきました
  - ポインタは2回目で理解できましたか?
- 来週は休講です
  - 代わりに2015/07/11に補講を実施します



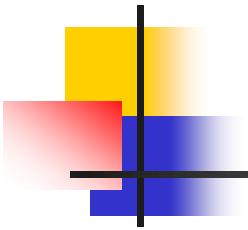
列挙型

- 名前つき整数定数のリスト
- 定義方法
  - enum タグ名 {列挙リスト};
  - enum color_type {red, green, yellow};
- 使用時の宣言
  - enum タグ名 変数名;
  - enum color_type mycolor;
- 定義と宣言を同時
  - enum color_type {red, green, yellow} mycolor;
- 列挙型の定数に整数が代入される
  - リストの最初が0，後は順に加算
  - 強制的に値を与えることも可能
    - enum color_type {red, green=9, yellow} mycolor; (yellowは10)

なぜ列挙型?

- 回答
  - 自己説明的なプログラムを記述するため
- プログラム例
  - ```
    #include <stdio.h>
    enum computer {keyboard, CPU, screen, printer};
    int main()
    {
      enum computer comp;
      comp = CPU;
      printf("%d¥0", comp);
    }
    ```
  - 次のページ
    - 信号の状態を列挙型で表している
    - 整数値で表すことも可能だが列挙型を使えばどんな状態を考えてプログラムが記述されているのか明確
    ```

```
#include <stdio.h>
enum state {RED, YELLOW, GREEN};
int main()
{
  enum state s;
  s = RED;
  while(1){
    if(s==RED) printf("RED %d¥n", s);
    if(s==YELLOW) printf("YELLOW %d¥n", s);
    if(s==GREEN) printf("GREEN %d¥n", s);
    sleep(3);
    switch(s){
    case RED:
      printf("change from RED to GREEN¥n" );
      s = GREEN;
      break;
    case YELLOW:
      printf("change from YELLOW to RED¥n" );
      s = RED;
      break;
    case GREEN:
      printf("change from GREEN to YELLOW¥n" );
      s = YELLOW;
      break;
    default:
      break;
    }
  }
}
```



構造体

- 構造体とは
  - 異なるデータを一つのグループとして取り扱う方法
  - 例 電話帳
    - char型　　名前
    - char型　　電話番号
    - char型　　メールアドレス
    - int型　　累積通話時間
    - int型　　通し番号
  - 例　日付
    - char型　　年号
    - int型　　年
    - int型　　月
    - int型　　日

構造体型・構造体変数の宣言

- 構造体型の宣言
  - struct date{
    char era[10];
    int year;
    int month;
    int day;
    };
  - "date"は構造体型のタグ
  - era, year, month, dateは構造体型のメンバ
- 構造体型変数の宣言
  - struct date a;
  - 構造体date型(struct date型)のaという変数
- これで複数の変数をまとめて取り扱える変数が完成

各メンバにアクセスするには

- "."を利用する
  - struct date{
    char era[10];
    int year;
    int month;
    int day;
    };
    struct date a; の場合
    - a.era[i]でera配列にアクセス
    - a.yearでyearにアクセス
    - a.monthでmonthにアクセス
    - a.dayでdayにアクセス

構造体型変数a

構造体変数のコピー・初期化

- 構造体変数のコピー
  - struct date a, e;と宣言されている場合
  - e=a; でaの内容をeにコピー可能
- 構造体変数の初期化
  - struct date a={"Heisei", 4,5,6};
  - で初期化可能

構造体型と構造体変数の同時宣言

- 以下のように宣言可能
  - struct date{
    char era[10];
    int year;
    int month;
    int day;
    } a;
  - struct person{
    char name[40];
    struct{char adr[90]; char phone[16];} home;
    struct{char adr[90]; char phone[16];} office;
    } x;
    - アクセスは x.home.adr, x.name, x.office.adrなどとする

構造体サンプルプログラム

```
#include <stdio.h>
struct date{
    char era[10];
    int year;
    int month;
    int day;
};
main()
{
  struct date a,e;
  strcpy(a.era, "Meiji");
  a.year = 40;
  a.month = 5;
  a.day = 10;
  e=a;
  printf( "%s %d Nen %d Gatsu %d Nichi¥n", a.era, a.year, a.month, a.day);
  printf( "%s %d Nen %d Gatsu %d Nichi¥n", e.era, e.year, e.month, e.day);
}
```

← 大域変数宣言ではないことに注意

構造体へのポインタ

- 構造体変数に対するポインタも作成可能
- 例
  - struct date a, *p;
    - pはstruct date型へのポインタ変数
    - p = &a; とすれば構造体型変数aへのポインタがpに代入される
  - ポインタ変数時のアクセス方法
    - (*p).year もしくは
    - p->year

構造体を関数とやりとりする

- 方法は2つ
- 値呼び出し (call by value)
  - 関数呼び出し時に構造体引数の内容が複写される
- 参照呼出し (call by reference)
  - 関数呼び出し時に構造体へのポインタを渡す
  - 構造体が大きな場合(メンバに大きな配列が含まれている場合)に構造体自身の複写をしなくてもよいので高速

値呼び出しの例

```
#include <stdio.h>
struct date{
  char era[10];
  int year;
  int month;
  int day;
};
void writeDate(struct date a){
  printf( "%s %d Nen %d Gatsu %d Nichi¥n", a.era, a.year, a.month, a.day);
}
struct date readDate()
{
  struct date d;
  scanf( "%s %d %d %d", d.era, &(d.year), &(d.month), &(d.day));
  return d;
}
main()
{
  struct date d;
  d=readDate();
  writeDate(d);
}
```

正しく動かない例 → 何故?

```
#include <stdio.h>
struct date{
  char era[10];
  int year;
  int month;
  int day;
};
void writeDate(struct date a){
  printf( "%s %d Nen %d Gatsu %d Nichi¥n", a.era, a.year, a.month, a.day);
}
void readDate(struct date d)
{
  scanf( "%s %d %d %d", d.era, &(d.year), &(d.month), &(d.day));
}
main()
{
  struct date d;
  readDate(d);
  writeDate(d);
}
```

参照呼出しの例

```
#include <stdio.h>
struct date{
  char era[10];
  int year;
  int month;
  int day;
};
void writeDate(struct date *a){
  printf( "%s %d Nen %d Gatsu %d Nichi¥n", a->era, a->year, a->month, a->day);
}
void readDate(struct date *d)
{
  scanf( "%s %d %d %d", d->era, &(d->year), &(d->month), &(d->day));
}
main()
{
  struct date d;
  readDate(&d);
  writeDate(&d);
}
```

構造体実践使用例　表の作成

- 住所検索プログラム
- 方針
  - 1人の住所情報(氏名, 郵便番号,　住所等)は構造体personに格納
  - 構造体personの配列を作成することで多数人の住所を管理

構造体の配列

- 構造体person
  - struct person{
    char name[40];
    char address[80];
    char phone[12];
    };
- struct person table[100];
  - 100人分のデータを格納するtableを作成
  - アクセス方法
    - table[i].name , table[i].address, table[i].phone

住所録検索プログラム

```
#include <stdio.h>
#define TABLESIZE 100
struct person{
  char name[40];
  char address[80];
  char phone[12];
};
void quit(char *message)
{
  fputs(message,stdout);
  exit(1);
}
int main(int argc, char **argv)
{
  struct person table[TABLESIZE];
  FILE *in;
  char target[40];
  int num;
  int i,j;
  if((in=fopen(argv[1],"r"))==NULL){
    quit( "File Not Found");
  }
  for(num=0;num<TABLESIZE;num++){
    if(fscanf(in,"%s %s %s",
              table[num].name,table[num].address,table[num].phone
              )==EOF)
      break;
  }
```

```
while(1){
    printf("Input name:");
    fgets(target,40,stdin);
    j=0;
    while(j<40){
      if(target[j]=='¥n') target[j]='¥0';
      j++;
    }
    if(target[0]=='¥0') return 0;
    for(i=0; i<num; i++){
      if(strcmp(table[i].name,target)==0){
        printf( "Address %s Phone %s¥n",
                                        table[i].address, table[i].phone);
        break;
      }
    }
    if(i==num)
      printf("Unknow person");
  }
}
```

新しいデータ型を定義する

- 既存の型・プログラマが新しく定義した型に自由に名称をつけることが可能
- typedefを用いる
- 定義の例
  - typedef int Seisuu32;
  - typedef unsigned short weight;
  - typedef int lengthTable[10];
  - typedef struct {
    int x;
    int y;
    } Coord2D;



typedefの例

- 定義の例
  - typedef int Seisuu32;
  - typedef unsigned short weight;
  - typedef int lengthTable[10];
  - typedef struct {
    int x;
    int y;} Coord2D;
- 使用例
  - Seisuu32 a;
  - weight b;
  - lengthTable b;
  - Coord2D a;
- メリット
  - プログラム自体が説明的になる
  - typedef宣言のみを書き換えれば型変更が可能

関数ポインタにおけるtypedef

- typedef int funcType(int);
  - funcType はint型引数を持ち，int型を変数を返す関数の型
- typedef funcType *funcPtrType;
  - funcTypePtrはfuncType型の関数へのポインタ型
- funcPtrType funcTable[10];
  - funcTableは関数へのポインタの配列



列挙型とtypedef

- 独自名の列挙型を作成可能
  - typedef enum{RED, GREEN, BLUE} SignalStatus;
  - SignalStatus signal_1;
  - SignalStatus signal_2;



共用体

- 同じメモリ領域を複数の変数で共有するもの
- メモリを共有する変数の型が同じである必要は無い
- 複数の変数を同時に使用することは不可能
  - 同じメモリ領域を共有しているため
- 定義自体は構造体と類似
- アクセス方法は構造体と同様
  - ドット演算子
  - アロー演算子

共用体の宣言

- unionがキーワード
- union u_type{
  int i;
  char c[2];
  double d;
  }sample;



共用体の使用

```
#include <stdio.h>
typedef enum {intType, floatType} Type;
typedef union{
  int intNum;
  float floatNum;
} Value;
typedef struct{
  Type type;
  Value value;
} Number;
void numPrint(Number x)
{
  switch(x.type){
  case intType:
    printf("%d¥n",x.value.intNum);
    break;
  case floatType:
    printf("%f¥n",x.value.floatNum);
    break;
  default:
    fprintf(stderr, "Unknown Type¥n");
    exit(1);
  }
}
```

```
main()
{
  Number a;
  a.type=floatType;
  a.value.floatNum = 10.67;
  numPrint(a);
  a.type=intType;
  a.value.intNum = 2;
  numPrint(a);
  a.type=floatType;
  numPrint(a);
}
```

共用体を用いたバイト順入れ替え

```
#include <stdio.h>
typedef union{
  int i;
  unsigned char c[4];
} SwabUnion;
void swabint(SwabUnion *u)
{
  unsigned char tmp[4];
  tmp[3]=u->c[0]; tmp[2]=u->c[1]; tmp[1]=u->c[2]; tmp[0]=u->c[3];
  u->c[0]=tmp[0]; u->c[1]=tmp[1]; u->c[2]=tmp[2]; u->c[3]=tmp[3];
}
main()
{
  SwabUnion swabunion;
  swabunion.i=132430;
  printf( "%d(%x)¥n",swabunion.i, swabunion.i);
  swabint(&swabunion);
  printf( "%d(%x)¥n",swabunion.i, swabunion.i);
}
```

以降来週の内容（予定）

自己参照構造体 (K&R p.169)

- 構造体内部に同じ構造体へポインタが定義されている
- 構造体自身を構造体メンバとして定義するのではなく、構造体へのポインタが定義される
- 例
  - struct tnode{
    char *word;
    int count;
    struct tnode *left;
    struct tnode *right;
    }

相互参照構造体

- 異なる構造体がお互いの構造体へのポインタを持つ
- 実体を宣言するのではなくポインタとして宣言するのがポイント
- 例
  - ```
    struct t {
      int a;
      int b;
      struct s *st_s;
    };
    struct s {
      int c;
      int d;
      struct t *st_t;
    }
    ```
- それぞれでそれぞれの実体を宣言してしまうと"chicken-egg"問題となってしまう
    ```

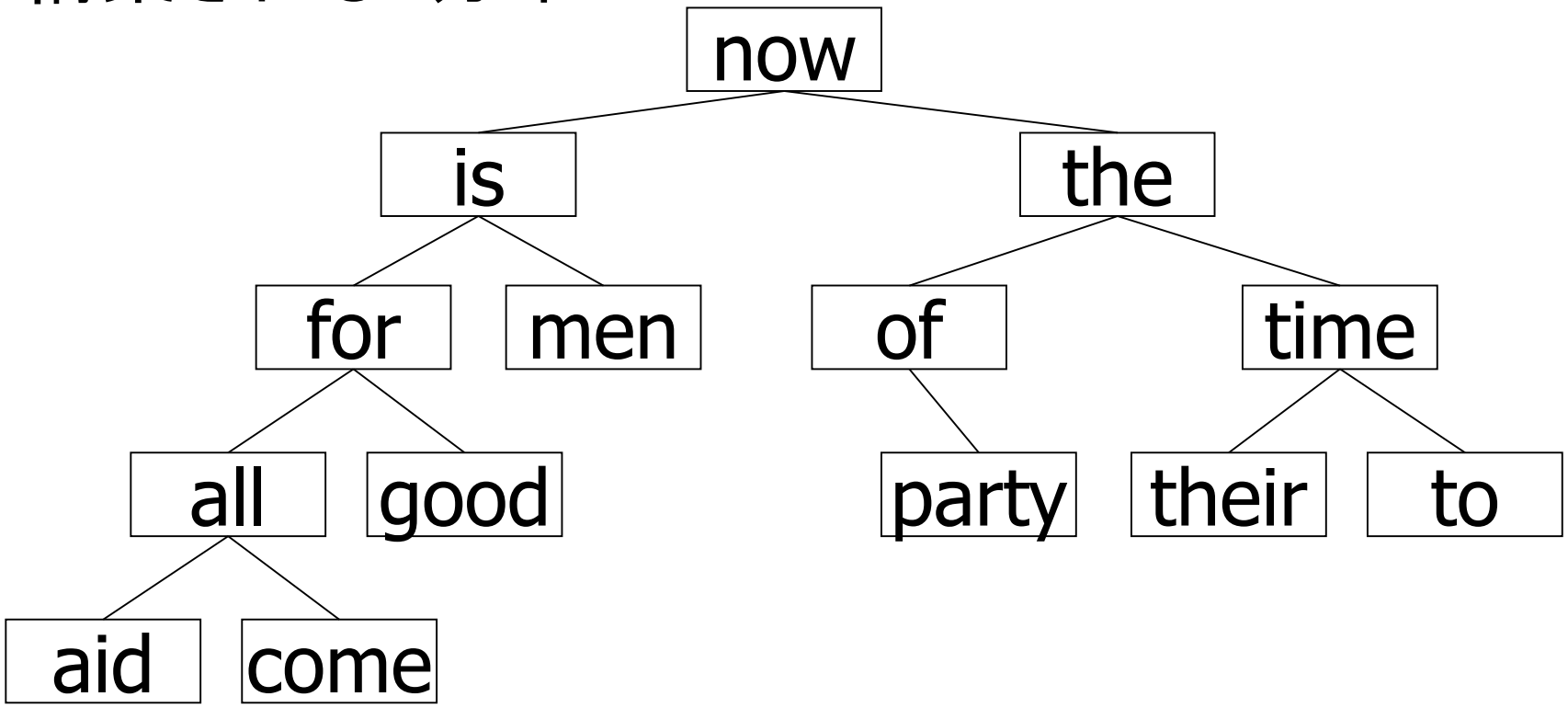
自己参照構造体の使用例



- 2分木リスト
- 例「入力されるテキストの出現頻度をカウント」
- 単語毎に1つのノードを持つ
  - 単語テキストへのポインタ
  - 出現回数のポインタ
  - 右の子ノードへのポインタ
  - 左の子ノードへのポインタ
- 任意のノード
  - 左の部分木:辞書順で小さい単語
  - 右の部分木:辞書順で大きな単語

2分木の構築

- 入力テキスト "now is the time for all good men to come to the aid of their party"
- 構築される2分木

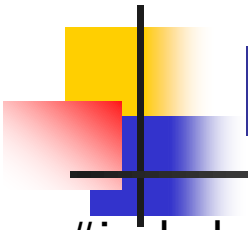


新たな単語か否かのチェック

- ルートから出発
- ノードに格納されている単語と入力単語をチェック
- 2つが一致したら→既出
- 入力単語がノードの単語よりも
  - 小さければ左の子供に対して探索を続行
  - 大きければ右の子供に対して探索を続行
- 求める方向に子供がなければ「新たな単語」
  - 「新たな単語」を2分木に追加

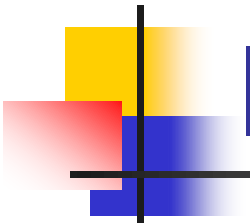
ノードを構造体で定義

- struct tnode{
  char *word;
  int count;
  struct tnode *left;
  struct tnode *right;
  }
- ツリー構造表現は自己参照構造体の使用の典型例



lec9-wordsearch.c (1/5)

```
#include <stdio.h>
#include <ctype.h>
#include <string.h>
#include <stdlib.h>
#define MAXWORD 100
struct tnode {
  char *word;
  int count;
  struct tnode *left;
  struct tnode *right;
};

struct tnode *addtree(struct tnode *, char *);
struct tnode *talloc(void);
void treeprint(struct tnode *);
int getword(char *, int);
int getch(void);
void ungetch(int c);
```



lec9-wordsearch.c (2/5)

```
int
main(int argc, char **argv)
{
  struct tnode *root;
  char word[MAXWORD];
  root = NULL;
  while(getword(word,MAXWORD)!=EOF){
    if(isalpha(word[0])){
      root = addtree(root,word);
    }
  }
  treeprint(root);
  return(0);
}
```

lec9-wordsearch.c (3/5)

```
struct tnode* addtree(struct tnode *p, char *w)
{
  int cond;
  if(p==NULL){
    p = talloc();
    p->word = strdup(w);
    p->count = 1;
    p->left = p->right = NULL;
  }else if ((cond=strcmp(w,p->word))==0){
    p->count++;
  }else if(cond<0){
    p->left = addtree(p->left,w);
  }else{
    p->right = addtree(p->right,w);
  }
  return p;
}
```

lec9-wordsearch.c (4/5)

```
void treeprint(struct tnode *p)
{
  if(p!=NULL){
    treeprint(p->left);
    printf("%4d %s¥n", p->count, p->word);
    treeprint(p->right);
  }
}

struct tnode *talloc(void)
{
  return((struct tnode *)malloc(sizeof(struct tnode)));
}
```

lec9-wordsearch.c (5/5)

```
int getword(char *word, int lim)
{
  int c; char *w = word;
  while(isspace(c=getch()))
    ;
  if(c!=EOF)
    *w++=c;
  if(!isalpha(c)){
    *w = '¥0';
    return c;
  }
  for( ; --lim>0; w++){
    if(!isalnum(*w=getch())){
      ungetch(*w);
      break;
    }
  }
  *w = '¥0';
  return word[0];
}
```