精密工学科プログラミング基礎 II

第5回資料

今回の授業で習得してほしいこと:

- 構造体 (教科書 91 ページ)
  - 定義の方法
  - 宣言と参照
  - 構造体の配列

- 動的な配列・メモリの確保 (教科書98ページ)

資料のURL : http://www.den.t.u-tokyo.ac.jp/prog

構造体 (structure) とは

- 異なる型 (int, float, char等) をまとめて扱う.
  - 例えば，以下のような原子に関するデータなど

name (char 型)	**group** (char 型)	**atomic mass** (double 型)
Hydrogen	non-metal	1.00797
Helium	non-metal	4.0026
Lithium	metal	6.939
Berylium	metal	9.0122
Boron	semi-metal-conductor	10.811

各原子を表す構造体

```
struct atom{
  char name[64];
  char group[64];
  double mass;
};
```

文字列

構造体の定義

- main 関数の外で,
  以下のように行う.

```
struct 構造体名{
   データ型1 メンバ名1;
   データ型2 メンバ名2;
    ・・・
};
```

プログラムの例

```
#include
<stdio.h>

struct atom{
   char name[64];
   char group[64];
   double mass;
};

int main(void){
    ・・・
}
```

構造体の宣言と参照

宣言

struct 構造体名 変数名;

参照

変数名.メンバ名

ピリオド
(ドット演算子)

プログラムの例

```
. . .
int main(void){

  struct atom a;

  a.name = “Helium” ;
  strcpy(a.name, “Helium” );
  a.mass = 4.0026;

  printf( “%s¥n” ,a.name);
  printf( “%lf¥n” ,a.mass);
}
```

構造体の配列

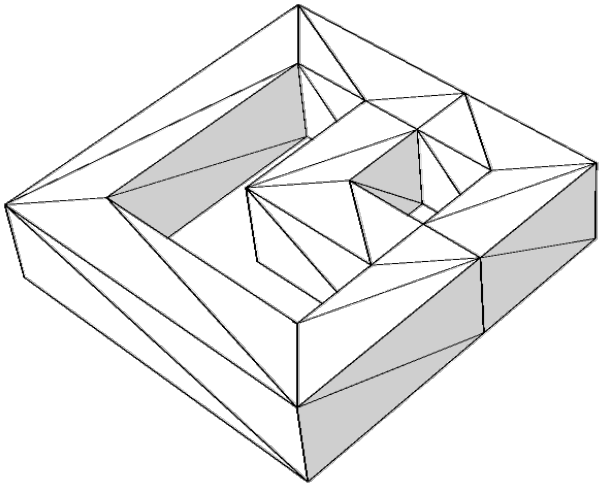
```
struct 構造体名 配列名[長さ];
```

プログラムの例

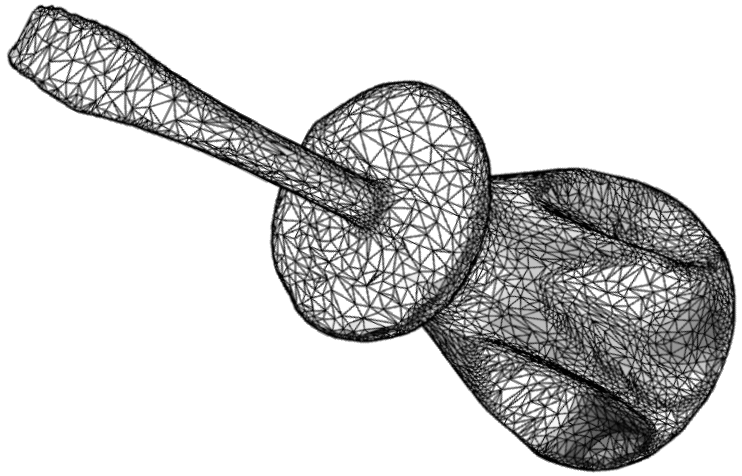
```
struct atom c[128];

c[0].name = “Helium” ;
strcpy(c[0].name, “Helium” );
c[0].mass = 4.0026;

c[1].name = “Hydrogen” ;
strcpy(c[1].name, “Hydrogen” );
c[1].mass = 1.00797;
```

動的配列とは

- プログラム実行時に，長さを決める配列
  - 例えば，以下のような時に使う



64個の三角形で表された形



13,000個の三角形で表された形

以下のようにプログラムを書けば，どちらも扱えるが・・・

```
struct triangle tris[20000];
```

効率がよくない。

動的配列の確保

注意：古いC 言語の規格では，配列の長さに変数は使用不可

できないことがある例

```
int n;
scanf( “%d” ,&n);
int a[n];
```

n 個分の int 型データ領域を確保 (memory allocation) する

```
int * a = (int *)malloc( n * sizeof(int) );
```

動的配列の宣言 ：（ * は，配列へのポインタ ）

```
  型 * 配列名;
```

動的配列の確保 ：

```
  配列名 = (型 *) malloc( 長さ * sizeof(型) );
```

動的配列の例

- 確保: malloc
- 参照: 配列と同じ
- 解放: free (使い終わったら行う)

```
int n, i;
double * a;
double s = 0;

scanf( "%d" ,&n);
a = (double*)malloc( n * sizeof(double) );

for(i=0; i<n; i++)
  s += a[i];

free(a);
```

宣言 (double 型の動的配列)

確保 (double 型を n 個分)

参照 (配列の i 番目)

解放

2次元の動的配列

- 動的配列の動的配列を使う
  - サイズが n × m の int 型の場合

宣言

```
(int*)* a;
```

確保

```
/* n 個の int* 型を確保する */
a = ((int*)*) malloc( n * sizeof(int*) );
for(i=0; i<n; i++)
  /* m 個の int 型を確保する */
  a[i] = (int*) malloc( m * sizeof(int) );
```

解放

```
for(i=0; i<n; i++)
  free( a[i] );
free( a );
```