データ構造とアルゴリズム (第12回)

静岡大学システム工学科

安藤 和敏

2008.01.31

§10.2 malloc()とfree()（メモリの動的割当て）

♠変数とメモリ

```
int x;
```

と宣言すれば，プログラムの実行と同時にsizeof(int)バイト(p.22を見よ) 分の領域がメモリ内に確保される．

♠配列の場合

```
#define N 100
int x[N];
```

と宣言すれば，プログラムの実行と同時にN×sizeof(int)バイト分の領域がメモリ内に確保される．

Nが最初から分かっているなら問題ないが，分かっていない場合は?

⇒ メモリ不足，又は，メモリの無駄使い

メモリの動的割当て

この問題はメモリの動的割当てによって解決できる.

つまり，データサイズNがプログラムを実行する前に分かっていなくても，プログラムの実行中にメモリを動的に確保できる.

C言語では，このメモリの動的割り当てを実装するために，

`malloc()` と `free()`

の二つの関数が用意されている.

これらの関数は，stdlib.h で定義されているので，これらの関数を使用するためには，このヘッダファイルを インクルード

```
#include <stdlib.h>
```

しなければならない.

malloc() (memory allocation の略であろう．)

読み方: エムアロック（マロック）

♠関数のプロトタイプ

```
void *malloc(int size);
```

引数として確保するメモリのサイズ(バイト単位)を与える，メモリを確保に成功すればそのメモリのアドレスを，失敗すれば NULL を返す.

返り値は，void型変数へのポインタ void *であるが，これは任意実体へのポインタを表す．

つまり，整数型実体か文字型実体か分からないが，とりあえずポインタが返ってくる．

実際に使うときには，キャスト演算子（→ p. 22）を用いて適当な実体を指すポインタに変換してやる．

```
int *p;
...
p = (int *)malloc(sizeof(int));
```

free() (メモリの解放)

♠関数のプロトタイプ

```
int free(void *p);
```

ポインタ p が指すメモリ領域を解放する．このポインタは，以前にmalloc()等で動的に確保されたメモリ領域を指すポインタでなければならない．返り値は，解放が成功したら 1 で失敗したら 0 (たぶん)．

プログラム終了時には，mallocで確保された領域は全て解放される．ので，いちいちfreeしてやる必要はない．

プログラム例 Prog 10.5 malloc1.c

```
#include <stdio.h>
#include <stdlib.h>
main(){
  int i,n,*array;
  printf("N= "); scanf("%d",&n);
  if ((array=(int *)malloc(n*sizeof(int)))==NULL) {
    fprintf(stderr,"malloc failed.\n");
    exit(1);
  }
```

```
for(i=0;i<n;i++) {
  array[i]=i;
  printf("%d ", array[i]);
}
putchar('\n');
  free(array);
```

プログラム例 Prog 10.6 malloc2.c

```
#include <stdio.h>
#include <stdlib.h>
#define PrintComplex(x) \
        printf("(%f%s%fi)", (x).real, ((x).img>=0)?"+":"-",
typedef struct {double real; double img;} CPLX;
```

```
main(){
  CPLX *a;
  a = (CPLX *)malloc(sizeof(CPLX));
  a->real = 10;
  a->img =3;
  PrintComplex(*a); putchar('\n');
  free(a);
}
```