「アルゴリズム」資料 2(g)
C の動的な配列

奈良女子大学理学部情報科学科 鴨浩靖

2009 年 10 月 13 日 初版
2011 年 10 月 11 日 第二版
2020 年 10 月 12 日 第三版

実行時に大きさが変わる配列を使いたい時

- 可変長配列を使う
- 記憶域管理関数を使う
  - `malloc()` を使う
  - `calloc()` を使う
  - (処理系依存な関数を使う)
- (その他、処理系依存な方法)

可変長配列を使う

例：

```
void    ayaya(int n)
{
        int     a[n * 2 + 1];
        /* a[] を使って何かする */
}
```

可変長配列を使う上での注意

一般に局所変数を使う上での注意はすべてあてはまる。

- 局所変数なので、関数の実行が終るとなくなる。

絶対にやってはならないトンデモない例：

```
int     *ayaya(int n)
{
        int     a[n * 2 + 1];
        /* a[] を使って何かする */
        return a; /* 危険！ 危険！ 危険！ 絶対にするな！ */
}
```

可変長配列を使う上での注意（つづき）

- 大きさは、作られたときに決まる。

```
int     *ayaya(int n)
{
        int     a[n * 2 + 1];
        /* a[] を使って何かする */
        n *= 2;     /* a[] の大きさは変わらない */
        /* a[] を使って別の何かをする */
}
```

malloc() を使う

例：

```
void    ayaya(int n)
{
        int     *a;
        a = (int *)malloc((n * 2 + 1) * sizeof(int));
        /* a[] を使って何かする */
        free(a);
}
```

malloc() を使う（つづき）

例：（これは OK）

```
int     *hoyoyo(int n)
{
        int     *a;
        int     i;
        a = (int *)malloc((n * 2 + 1) * sizeof(int));
        for (i = 0; i < n * 2 + 1; i ++) {
                a[i] = i;
        }
        return a;
}
```

calloc() を使う

例：

```
void    ayaya(int n)
{
        int     *a;
        a = (int *)calloc(n * 2 + 1, sizeof(int));
        /* a[] を使って何かする */
        free(a);
}
```

malloc() と calloc() の違い

- `malloc()` は 1 引数。総バイト数を指定。
  `calloc()` は 2 引数。第一引数で要素の個数、第二引数で要素あたりのバイト数を指定。
- `malloc()` で確保した領域にはゴミが入ったまま。だから速い。
  `calloc()` で確保した領域はゼロクリアされる。ちょっと安全。

malloc() や calloc() を使う上での注意

- 使い終ったら必ず free() をすること。
  - ただし、malloc() や calloc() の実行と対応する free() の実行は、同じ関数定義の中でなくとも良い。この点、可変長配列よりも柔軟。
- 使い終る前に free() をしないこと。
  - 逆に言えば、free() をした後で使わないこと。
- すでに free() したものを再度 free() しないこと。
- #include <stdlib.h> を忘れないように。

free() を忘れると

malloc() や calloc() を実行するたびに、使用中のメモリの量が増えていく。ずっと実行していると、やがて、メモリ不足で倒れる。
この現象をメモリリークと呼ぶ。ある程度の時間動かさないと発現しないので、とても取りにくいバグである。

free() が早すぎると

あらぬところを指すポインタの実体を取ることになるので、予期しない結果になる。
この現象をダングリングポインタという。運が良ければセグメント違反で落ちてくれるが、運が悪いと、全然関係ない変数を触って、謎のバグになる。

free() を多重に行うと

後で、同じメモリが多重に割り当てられる危険がある。
最近は、ライブラリが多重 free() を防ぐ機能をもっていることも多いが、頼りすぎないように。

まとめ

実行時に大きさが変わる配列を使いたい時は、

- 可変長配列を使う
- `malloc()` など記憶域管理関数を使う
- (その他)

方法がある。
いずれの場合も、取り扱い上の注意を守ること。