Programming II 0x0c

動的データ構造（2011.06.23）

塩澤秀和 http://vilab.org

ポインタ演算（復習）

ポインタと整数の加減算		説明
`ptr + n`		ptr からデータ n 個分進んだ位置を指すポインタ
`ptr - n`		ptr からデータ n 個分戻った位置を指すポインタ
`ptr += n`		ptr = ptr + n
`ptr -= n`		ptr = ptr - n
`ptr++`	`++ptr`	ptr += 1　※後置と前置の違いは通常の変数と同じ
`ptr--`	`--ptr`	ptr -= 1　※後置と前置の違いは通常の変数と同じ

※ ポインタと（素の）アドレス値の違い
　ポインタには指すデータの“型”がある ⇒ データのバイト長を考慮した加減算

ポインタ同士の演算	説明
ポインタの差	`ptr2 - ptr1`（例：&a[4] - &a[2]） 2つのポインタの間に入れられるデータの個数
ポインタの比較	演算子 `< > <= >= == !=` 2つのポインタのアドレス値の大小を比較

配列とポインタの関係（復習）

教科書には「まったく同じ」と書かれているが厳密には例外がある

- 配列の名前≒ポインタ（p.172）
  - 配列名は、その先頭要素を指すポインタに変換される
    ⇒ ptr = array は ptr = &array[0] と同じ意味になる

- 【公式】 x[i] ⇔ *(x + i) （まったく同じ意味）
  - x が配列でもポインタでも成り立つ関係
    配列：a[i] ⇔ *(a + i)　ポインタ：*(p + i) ⇔ p[i]
  - C言語はポインタ中心 ⇒ 実は x[i] は *(x+i) の簡略表記

- ポインタの扱いは慎重に…
  - ポインタ演算は強力なので、バグの原因になりやすい
  - 特に、ptr++ などポインタの値を増減するときは要注意

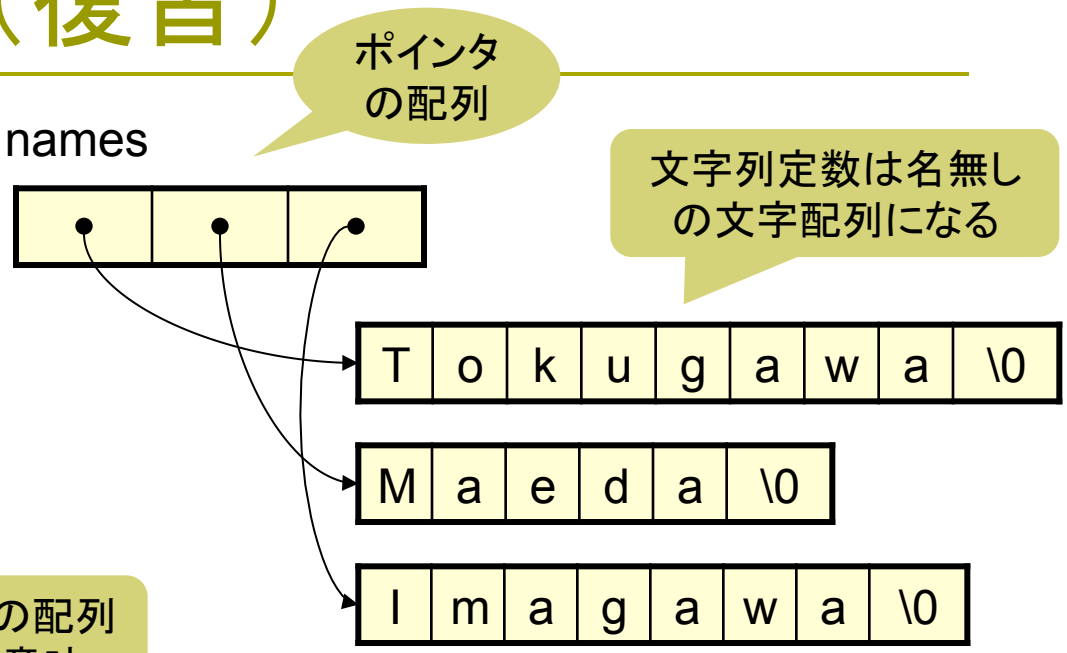
ポインタの配列（復習）

```
#include <stdio.h>

int main(void)
{
  int i;
  char *names[3] = {
    "Tokugawa",
    "Maeda",
    "Imagawa"
  };

  for (i = 0; i < 3; i++) {
    printf("%s\n", names[i]);
  }
  return 0;
}
```

char * の配列という意味



※ 比較せよ

```
char names[3][10] = {
  "Tokugawa",
  "Maeda",
  "Imagawa"
};
```

どこが間違っているか？

```
# include <stdio.h>

int *gyaku(int *a, int n);

int main(void)
{
  int n, i;
  int *rdata;

  printf("データの個数は？ ");
  scanf("%d", &n);
  int data[n];

  printf("データを入力:\n");
  for (i = 0; i < n; i++)
    scanf("%d", &data[i]);
  printf("逆順に並べ替え:\n");
  rdata = gyaku(data, n);

  for (i = 0; i < n; i++)
    printf("%d ", rdata[i]);
  return 0;
}

int *gyaku(int *a, int n)
{
  int i;
  int b[n];

  for (i = 0; i < n; i++)
    b[(n-1)-i] = a[i];
  return b;  /* &b[0] と同じ */
}
```

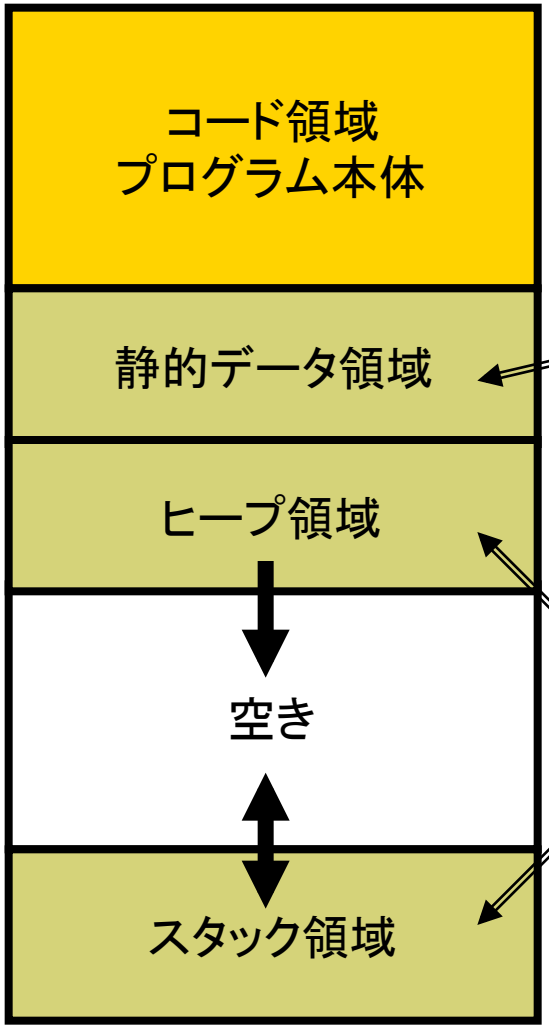
malloc / free

- 動的なメモリ割り付け
  - プログラム実行中に（動的に）、メモリ領域を確保する
  - 関数 malloc と free を使用する（#include <stdlib.h>）

- malloc（p.153）
  - 指定されたバイト数の“消えない”メモリを確保する
  - 戻り値はメモリ領域の先頭アドレス（失敗するとNULL）
  - 例： char *ptr;
    ptr = (char *) malloc(文字数 + 1); ← +1は'\0'の分

- free（p.154）
  - malloc で確保したメモリを解放する（戻り値はない）
  - 例： free(ptr);

変数やデータの格納場所




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

/* グローバル変数 */
char a[100];

int main(void)
{
   /* ローカル変数 */
   char b[100];
   char *c;

   /* mallocで割り当て */
   c = (char *) malloc(100);
   return 0;
}
```

プロセス空間（実行中のプログラムメモリ空間）

mallocの使用例

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

int main(void)
{
  char *str;   /* ポインタを用意 */
  int n;

  printf("何文字？ ");
  scanf("%d", &n);

  /* n文字の文字列のメモリを確保 */
  /* ('\0'を含めたn+1バイトが必要) */
  str = (char*) malloc(n+1);

  /* 確保失敗時のエラー処理 */
  if (str == NULL) {
    printf("メモリ確保失敗\n");
    exit(1);
  }

  /* 確保したメモリの利用 */
  printf("文字列: ");
  scanf("%s", str);
  printf("%s\n", str);

  /* 最後に確保したメモリを解放 */
  free(str);
  return 0;
}
```

記憶域の割り付け方式

- 静的な記憶域の割り付け
  - プログラム実行前に、固定のメモリ領域を確保しておく
  - プログラム実行中ずっと存在し、追加・拡張はできない
  - C言語: グローバル(外部)変数、static(静的)変数
- 動的な記憶域の割り付け
  - プログラム実行中に、必要な量のメモリ領域を確保する
  - 必要に応じて、拡張したり、解放(削除)したりできる
  - C言語: 関数 malloc を使う(プログラマが管理する)
- (自動的・一時的な割り付け)
  - 関数に入るときに確保され、出ると自動的に解放される
  - C言語: (static以外の)ローカル変数、仮引数

動的な配列の確保

- 一般的な malloc
  - ポインタ = (型名 *) malloc(要素数 * sizeof(型名));
  - 例： int *ptr;
    ptr = (int *) malloc(n * sizeof(int));
- malloc 使用上の注意
  - 本当に必要なときに使う（普通の配列のほうが速くて安全）
  - 自前でメモリ使用を管理して、使わなくなったら free する
- NULLポインタとは？
  - 無効なことが保証されているポインタ（ポインタの値が0）
  - 注意： NULLポインタとヌル文字（'\0'）は別の意味

動的配列の例

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

int main(void)
{
  int *p; /* ポインタを用意する */
  int i, sum;

  /* バイト数を計算してメモリを確保 */
  /* int は1つ sizeof(int) バイト必要 */.
  p = (int *) malloc(
          3 * sizeof(int));
  if (p == NULL) {
    printf("メモリ確保失敗\n");
    exit(1);
  }
```

```
/* 動的に確保したメモリ領域は */
/* 配列のように [ ] で参照できる */
p[0] = 5;
p[1] = 6;
p[2] = 7;

sum = 0;
for (i = 0; i < 3; i++){
  sum += p[i];
}
printf("sum: %d\n", sum);

free(p);
return 0;
}
```

演習問題

12a. mallocで確保した20バイトの領域に文字列を読み込んで、1文字ずつ別々に表示するプログラムを作成しなさい。

12b. mallocを使って、「どこが間違っているか？」のプログラムを正しく動くように修正しなさい。

12c. キーボードから n 個の数値データ(double)を読み込んで、マイナスのものだけを再表示するプログラムを作成しなさい。ただし、n も最初にキーボードから読み込み、数値データを読み込む配列は malloc で動的に確保すること。

12d. 前問12cを、キーボードの代わりにファイルから読み込むようにしなさい。つまり、ファイルの形式は、1行目にデータの個数があり、2行目以降にその個数分の数値データが続く。

- 教科書 pp.157-160, 164-165, 171 は高度なので各自学習