C言語の基礎6

ポインタ

> https://ja.wikibooks.org/wiki/C%E8%A8%80%E8%AA%9E/%E3%83%9D%E3%82%A4%E3%83%B3%E3%82%BF

- ポインタとは、オブジェクト（変数や関数、構造体など）のメモリ上のアドレスを示すもの
    - javaの参照型変数に近い

アドレスはどのように利用するか

【address.c】

```
#include <stdio.h>

int main(void) {
    int a = 1234;
    printf("変数aの値は%d\n", a);
    printf("変数aのアドレスは%p\n", &a);
    printf("main関数のアドレスは%p\n", &main);
}
```

【実行結果】

```
$ gcc address.c
$ ./a.out
変数aの値は1234
変数aのアドレスは0x7ffe5c128734
main関数のアドレスは0x55688a592169
```

- 実行時の環境によって、aの値以外は異なる。
- 出力書式の `%p` やその他については以下のURLを参照のこと

> https://ja.wikipedia.org/wiki/Printf

C言語でアドレスを取得する方法

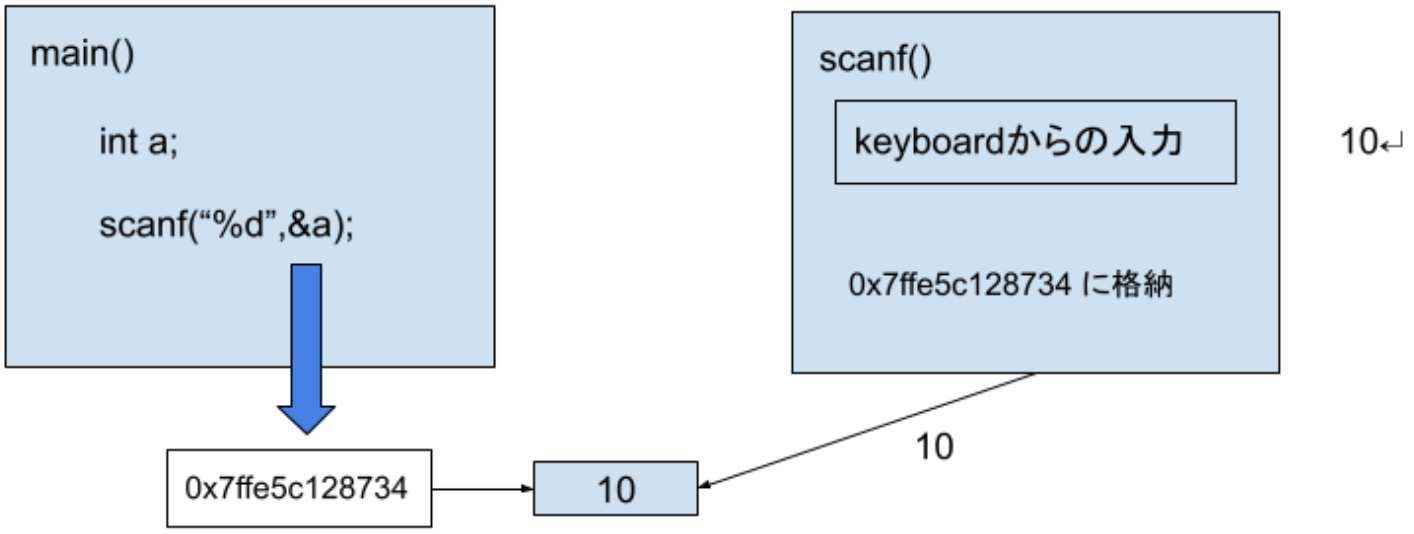
```
&a
```

- 上記のように、オブジェクトに `&` を付けることで、アドレスを取得することができる。

入力関数の仕組み

```
int a;
scanf("%d",&a);
```

- これは具体的に、何を行っているのだろうか？
- 関数 `scanf( )` は、書式文字列 `"%d"` と、 `&a` を引数として受け取っている。
    - 入力された文字を数字と見なし（ `%d` ）、変数 `a` のメモリアドレスに、入力された数字を書き込んでくれる関数



- `scanf()` 関数内で行った処理で、main側の変数 `a` の値を書き換えることが可能になる。

配列の場合の入力

```
char str[100];
scanf("%s",str);
```

- 文字列の入力では、 `&` 無しで、配列の名前を書けばよかった。
- なぜだろうか？

【address2.c】

```
#include <stdio.h>

int main(void) {
    char str[100];
    printf("変数strのアドレスは%p\n", str);
    printf("変数str[0]のアドレスは%p\n", &str[0]);
}
```

```
$ ./a.out
変数strのアドレスは0x7ffc0120da50
変数str[0]のアドレスは0x7ffc0120da50
```

- 配列名 `str` の値と配列 `str[0]` のアドレスは同じ！

  ```
  str == &str[0]
  ```

  - **この関係はしっかり覚えておく必要がある。**

ポインタ変数

- メモリ上のアドレス（ポインタ）を格納するための変数

```
int *p1;  // int型の変数を示すポインタ変数
char *p2; // char型の変数を示すポインタ変数
```

- 型を指定するのは、配列や構造体（後述）のサイズを勘案するための情報
- 型のサイズを考慮してポインタ（アドレス）を増減する

【int_pointer.c】

```
#include <stdio.h>

int main(void) {
    int a[5] = {0, 1, 2, 3, 4};
    int *p = a; // int *p = &a[0];

    printf("変数a[0]のアドレスは%p\n", &a[0]);
    printf("変数a[1]のアドレスは%p\n", &a[1]);
    printf("変数a[2]のアドレスは%p\n", &a[2]);
    printf("変数a[3]のアドレスは%p\n", &a[3]);
    printf("変数a[4]のアドレスは%p\n", &a[4]);

    printf("変数pは%p\n", p++);
    printf("変数pは%p\n", p++);
    printf("変数pは%p\n", p++);
    printf("変数pは%p\n", p++);
    printf("変数pは%p\n", p++);
}
```

【char_pointer.c】

```
#include <stdio.h>

int main(void) {
    char a[5] = {0, 1, 2, 3, 4};
    char *p = a; // char *p = &a[0];

    printf("変数a[0]のアドレスは%p\n", &a[0]);
    printf("変数a[1]のアドレスは%p\n", &a[1]);
    printf("変数a[2]のアドレスは%p\n", &a[2]);
    printf("変数a[3]のアドレスは%p\n", &a[3]);
    printf("変数a[4]のアドレスは%p\n", &a[4]);

    printf("変数pは%p\n", p++);
    printf("変数pは%p\n", p++);
```

```
    printf("変数pは%p\n", p++);
    printf("変数pは%p\n", p++);
    printf("変数pは%p\n", p++);
}
```

- 実行結果をよく見比べて欲しい。

ポインタ変数の初期化と示す場所の値

- 基本的には、`NULL`で初期化する。
    - もしくは、先に示したように特定の変数のアドレスを格納する
- 指し示す場所の値（メモリアドレスに格納されている値）を取り出す場合は、`*`をポインタ変数の前に付ける

【pointer.c】

```
#include <stdio.h>

int main(int argc, char const *argv[]) {
    int a = 10;
    int *p = NULL;

    printf("aの値:%d\n", a);
    p = &a;
    printf("pの指し示す場所の値:%d\n", *p);
    return 0;
}
```

- 最初の`int *p`は、pがポインタ変数であることを示す`*`
- printf内の`*p`は、pの指し示すアドレスに格納されている値を示す`*`

入れ替え関数（swap関数）

2数の入れ替えを行う処理

【swap_1.c】

```
#include <stdio.h>

int main(int argc, char const *argv[]) {
    int a = 10, b = 20;
    int temp;

    printf("a:%d , b:%d\n", a, b);
```

```
    // 入れ替え処理
    temp = a;
    a = b;
    b = temp;

    printf("a:%d , b:%d\n", a, b);

    return 0;
}
```

この処理をそのまま関数化してみる

【swap_2.c】

```
#include <stdio.h>

void swap(int, int);

int main(int argc, char const *argv[]) {
    int a = 10, b = 20;

    printf("a:%d , b:%d\n", a, b);
    // 入れ替え処理
    swap(a, b);

    printf("a:%d , b:%d\n", a, b);

    return 0;
}

void swap(int x, int y) {
    int temp;
    printf(" x:%d , y:%d\n", x, y);
    temp = x;
    x = y;
    y = temp;
    printf(" x:%d , y:%d\n", x, y);
}
```

- swap内では、入れ替えを正しく行っている。（デバッグしてください）
    - 呼び出し元のmainでは、値の入れ替えが行われない。
    - 関数swapには、aとbのコピーが渡されるのみ

ポインタを使用する

【swap_3.c】

```c
#include <stdio.h>

void swap(int *, int *);

int main(int argc, char const *argv[]) {
    int a = 10, b = 20;

    printf("a:%d , b:%d\n", a, b);
    // 入れ替え処理
    swap(&a, &b);

    printf("a:%d , b:%d\n", a, b);

    return 0;
}

void swap(int *x, int *y) {
    int temp;
    printf(" *x:%d , *y:%d\n", *x, *y);
    temp = *x;
    *x = *y;
    *y = temp;
    printf(" *x:%d , *y:%d\n", *x, *y);
}
```

練習1

前回作成した05_prog2.cを以下の条件で書き換える（ren1_1.c）

- 関数lower_c() で出力している場合、main内で出力する
- 関数lower_c() は、ポインタで値を受取る

【05_prog2.c】

```
#include <stdio.h>

char lower_c(char str);

int main(int argc, char const *argv[]) {
    char str[] = "HELLO, World!";

    for (int i = 0; str[i] != '\0'; i++) {
        printf("%c", lower_c(str[i]));
    }
    printf("\n");
    return 0;
}

/**
 * @brief 文字を受け取り小文字に変換する
 *
 * @param str
 * @return char
 */
char lower_c(char str) {
    char c;
    if ('A' <= str && str <= 'Z') {
        c = str + ('a' - 'A');
    } else {
        c = str;
    }
    return c;
}
```

【ren1_1.c】

```
#include <stdio.h>

void lower_c(char *str);

int main() {
    char str[] = "HELLO, World!";
```

```
    printf("before:%s\n", str);
    lower_c(str);
    printf("after :%s\n", str);

    return 0;
}

void lower_c(char *str) {
  // 各自コードを作成



}
```

文字数を数える

- ポインタで文字列を受け取り、文字数を返す関数 `my_strlen()` を作成する。
- 動作を確認するmainも作成する

【ren1_2.c】

```
#include <stdio.h>

int my_strlen(char *);

int main(int argc, char const *argv[]) {
    char string[] = "Hello";

    printf("%s は %d文字\n", string, my_strlen(string));
    return 0;
}

int my_strlen(char *str) {


}
```

文字列の結合

- 前回作成した【concat.c】を関数化してみる（ren1_3.c）
    - 関数名 `fconcat()` とし、結合後の配列、と結合したい文字列を2つ渡す

【concat.c】

```c
#include <stdio.h>

int main(int argc, char const *argv[]) {
    char str1[] = "Hello";
    char str2[] = "World";
    char concat[100]; // 結果を入れる配列
    int j = 0;        // concat[j]の添字

    // str1 を concat にコピー
    for (int i = 0; str1[i] != '\0'; i++, j++) {
        concat[j] = str1[i];
    }
    // str2 を concat にコピー
    for (int i = 0; str2[i] != '\0'; i++, j++) {
        concat[j] = str2[i];
    }
    // 終端記号を付加
    concat[j] = '\0';
    printf("concat : %s\n", concat);

    return 0;
}
```

【ren1_3.c】

```c
#include <stdio.h>

void fconcat(char[], char[], char[]);

int main(int argc, char const *argv[]) {
    char str1[] = "Hello";
    char str2[] = "World";
    char concat[100]; // 結果を入れる配列

    fconcat(concat, str1, str2);

    printf("fconcat : %s\n", concat);

    return 0;
}

void fconcat(char result[], char str1[], char str2[]) {
  // 動作するように書き換える




}
```

- ポインタを使用するように書き換えてください。（ren1_4.c）

【ren1_4.c】

```c
#include <stdio.h>

void pconcat(char *, char *, char *);

int main(int argc, char const *argv[]) {
    char str1[] = "Hello";
    char str2[] = "World";
    char concat[100]; // 結果を入れる配列

    pconcat(concat, str1, str2);

    printf("pconcat : %s\n", concat);

    return 0;
}

void pconcat(char *result, char *str1, char *str2) {
  // 動作するように書き換える




}
```

仕様を変更

- 標準ライブラリ `string.h` に用意されている `strcat()` と同じ動作をする `my_strcat()` を作成する。（ren1_5.c）

> https://monozukuri-c.com/langc-funclist-strcat/

参考【strcat.c】

```c
#include <stdio.h>
#include <string.h>

int main(void) {
    char hello[100] = "Hello";  //  結合先
    char world[6]   = "World";  //  結合文字列

    strcat(hello, world);

    printf("%s", hello);
    return 0;
}
```

【ren1_5.c】

```c
#include <stdio.h>

char *my_strcat(char *, char *);

int main(int argc, char const *argv[]) {
    char str1[100] = "Hello";
    char str2[]    = "World";

    my_strcat(str1, str2);

    printf("my_strcat : %s\n", str1);

    return 0;
}

char *my_strcat(char *str1, char *str2) {









}
```

string.h

> https://ja.wikibooks.org/wiki/C%E8%A8%80%E8%AA%9E/%E6%A8%99%E6%BA%96%E3%83%A9%E3%82%A4%E3%83%96%E3%83%A9%E3%83%AA/string.h

- 文字列に関する一般的な関数が用意されている。
    - どのような関数が用意されているか、一度確認しておいてほしい。

> http://www.c-lang.org/detail/lib_function_header.html#string_h

問題

以下のプログラムを作成してください。
動作を確認するため、mainも作成する。（提出）

1. 関数名：findMax
   説明：整数の配列とその要素数を受け取り、配列内の最大値を返す関数を実装してください。
   ポインタを使用して配列を受け取る方法を採用してください。(prog1.c)

```c
#include <stdio.h>

int findMax(int *, int);

int main() {
    int numbers[] = {5, 2, 9, 1, 7};
    int size = sizeof(numbers) / sizeof(int);

    int max = findMax(numbers, size);
    printf("The maximum number is: %d\n", max);

    return 0;
}

int findMax(int *arr, int size) {









}
```

2. 関数名：reverseString
   説明：文字列を逆順にする関数を実装してください。
   ポインタを使用して文字列を受け取る方法を採用してください。（prog2.c）

   - 必要であれば、文字数を数える関数 `my_strlen` を使用しても良い。
     my_strlen("abc") → 3

```c
#include <stdio.h>
#include <string.h>

void reverseString(char *);
int my_strlen(char *);
```

```c
int main() {
    char message[] = "Hello, World!";

    printf("Before: %s\n", message);
    reverseString(message);
    printf("After : %s\n", message);

    return 0;
}

int my_strlen(char *str) {
    int n = 0;
    while (*str++) {
        n++;
    }
    return n;
}

void reverseString(char *str) {




}
```

3. 関数名：incrementArray
   説明：整数の配列とその要素数を受け取り、配列の全ての要素に1を加える関数を実装してください。
   ポインタを使用して配列を受け取る方法を採用してください。（prog3.c）

```c
#include <stdio.h>

void incrementArray(int *, int);

int main() {
    int numbers[] = {1, 2, 3, 4, 5};
    int size = sizeof(numbers) / sizeof(numbers[0]);

    printf("Before: ");
    for (int i = 0; i < size; i++) {
        printf("%d ", numbers[i]);
    }
    printf("\n");

    incrementArray(numbers, size);

    printf("After : ");
    for (int i = 0; i < size; i++) {
        printf("%d ", numbers[i]);
    }
```

```
    printf("\n");

    return 0;
}

void incrementArray(int *arr, int size) {









}
```