C 言語 024 malloc 関数

【malloc 関数】

malloc 関数はメモリ内に要素数を自由に変更できる配列のような物を作る関数です。
自動変数（auto 変数）は値を「スタック領域」、静的変数（static 変数）は「スタティック領域」に記録しますが、malloc 関数では値を「ヒープ領域」に記録することができます。
ヒープ領域は他領域と比べて使用できる容量が多く、領域の確保と解放を任意で行うことができ、解放されるまで値を保持し続けることができるので、大きなデータを長く記録したい場合に用います。

＜malloc 関数＞

定　　義	void* malloc(size_t size);
ﾍｯﾀﾞｰﾌｧｲﾙ	stdlib.h
使 用 方 法	ポインタ変数 = (データ型*)malloc(メモリサイズ);
戻　り　値	正常：ヒープ領域に確保したアドレス／異常（領域を確保できなかった）：NULL

void は「型指定無し」、void*は「型指定の無いアドレス」になります。

＜例文＞

```
int* heap;                  /* ポインタ変数を用意 */
heap = (int*)malloc(20);    /* 20 バイトで int 型を 5 個分記録できる */
heap[0] = 10;               /* 代入は配列と同じ。変数 heap は int 型なので整数を代入 */
free(heap);                 /* 確保した領域を解放する */
```

先頭アドレスを 100 番地とする。

＜ヒープ領域の解放＞

ヒープ領域に確保した領域を解放するには free 関数を使用します。解放されたアドレスは再利用されます。無駄にメモリ消費をしないためにも使用後は解放すること。

＜free 関数＞

定　　義	void free(void* pointer);
ﾍｯﾀﾞｰﾌｧｲﾙ	stdlib.h
説　　明	ヒープ領域内の引数のアドレスが指す領域を解放する。

＜サンプル 1＞

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

int main(void) {

    int  len;     /* 要素数 */
    int* heap;    /* ヒープ用のポインタ変数 */
    int  i;       /* ループカウンタ */

    /* 要素数を入力 */
    printf("要素数：");
    scanf("%d", &len);

    /* ヒープを確保する、メモリサイズは「int 型サイズ×要素数」*/
    heap = (int*)malloc(sizeof(int) * len);

    /* ヒープが確保できなければプログラムを終了する */
    if (heap == NULL) {
        exit(-1);
    }

    /* ヒープ変数に代入 */
    for (i = 0; i < len; i++) {
        heap[i] = i;
    }

    /* ヒープ変数の中身を表示 */
    for (i = 0; i < len; i++) {
        printf("%d¥n", heap[i]);
    }

    /* ヒープ領域を開放する */
    free(heap);

    return 0;
}
```

【realloc 関数】

realloc 関数は malloc 関数で作成した領域のサイズ（要素数）を変更する関数です。

＜malloc 関数＞

定　　義	void* realloc(void* pointer , size_t size);
ﾍｯﾀﾞｰﾌｧｲﾙ	stdlib.h
使用方法	ポインタ変数 = (データ型*)realloc(ポインタ変数 , サイズ)
戻　り　値	正常：ヒープ領域に確保したアドレス／異常（領域を確保できなかった）：NULL

＜例文＞

```
int* heap;
heap = (int*)malloc(40);            /* int 型で要素数 10 個分のヒープ領域を確保する */
heap = (int*)realloc(heap, 80);     /* 領域を 80 バイトに変更。要素数は 20 個分になる */
```

【ヒープ領域を解放しなかった場合】

ヒープ領域は領域の確保、解放をプログラマ側で任意に行える領域です。

malloc 関数で領域を確保したが、free 関数で領域の解放を行わなかった場合、ヒープ領域に記録した値が残ることになってしまい、無駄にメモリを消費することになります。

現在は OS が使用していないヒープ領域を解放するようになっているので、解放し忘れた値がずっとメモリ内に残るわけではありませんが、それは OS の仕様の話であって、C 言語の仕様上ではヒープ領域を確保したら必ず解放することとなっています。

【構造体とヒープ】

構造体をヒープに代入するサンプルコード。

＜サンプル 2＞

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

struct personal {
    char name[10];
    int  age;
};

int main(void) {

    /* 10 個分確保する */
    struct personal* heap;
    heap = (struct personal*)malloc(sizeof(struct personal) * 10);
    if (heap == NULL) { exit(-1); }

    /* 1 個だけデータを入力して表示する */
    strcpy(heap[0].name, "A 太郎");
    heap[0].age = 20;
    printf("%s さんは%d 歳です。¥n", heap[0].name, heap[0].age);

    free(heap);    /* ヒープ領域を開放する */
    return 0;
}
```

【memcpy 関数】

定　　義	void* memcpy(void* data1, const void* data2, size_t size);
ﾍｯﾀﾞｰﾌｧｲﾙ	string.h
説　　明	data2 の領域にあるデータを size 分だけ data1 にコピーする関数
戻 り 値	data1 のアドレス

＜例文 1＞

```
int data1[5];
int data2[] = {1,2,3,4,5};
memcpy(data1, data2, sizeof(data1));    /* data2 のデータを data1 にコピーする */
```

＜例文 2＞

```
int  len = 10;
int* data1;
int* data2;
int  i;

/* data1 の領域を確保 */
data1 = (int*)malloc(sizeof(int) * len);

/* data1 の領域にデータを代入 */
for (i = 0; i < len; i++) {
    data1[i] = i;
}

/* data2 の領域を確保 */
data2 = (int*)malloc(sizeof(int) * len);

/* data1 の領域を data2 の領域にコピーする */
memcpy(data2, data1, sizeof(int) * len);

/* data2 の領域を表示して確認 */
for (i = 0; i < len; i++) {
    printf("%d¥n", data2[i]);
}

/* 確保した領域を解放 */
free(data1);
free(data2);
```