メモリのヒープ領域について

プログラマのためのC言語 第16回

概要

- ✔ ヒープ領域のイメージ
- ✔ ヒープ領域の利用例

ヒープ領域のイメージ

ヒープ領域のイメージ

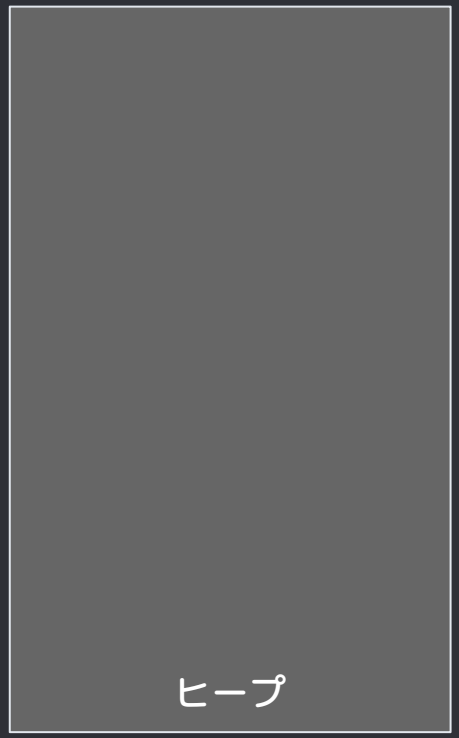
前回の動画ではメモリは大きく4つの領域にわかれると解説しました

ヒープ領域のイメージ

今回の動画ではヒープ領域について詳しくみていきます

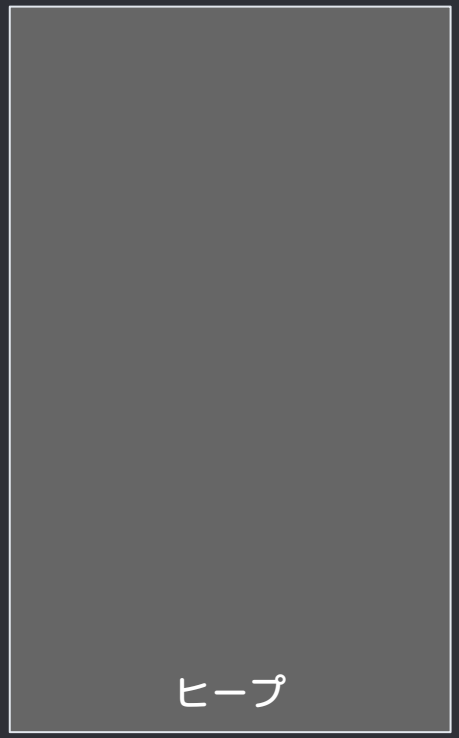
ヒープ領域のイメージ

ヒープ領域はプログラムの処理に応じてメモリが確保・解放される



ヒープ領域のイメージ

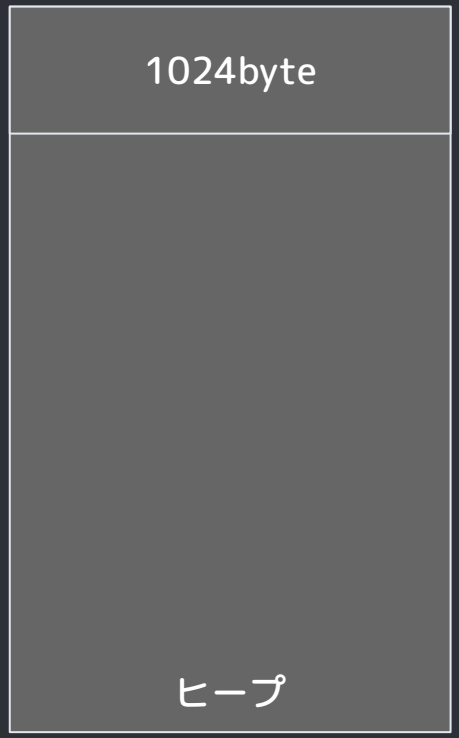
ヒープ領域はプログラムの処理に応じてメモリが確保・解放される




```c
#include <stdlib.h>

int main(void)
{
    void* p = malloc(1024);

    memset(p, 0, 1024);

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

ヒープ領域のイメージ

ヒープ領域はプログラムの処理に応じてメモリが確保・解放される

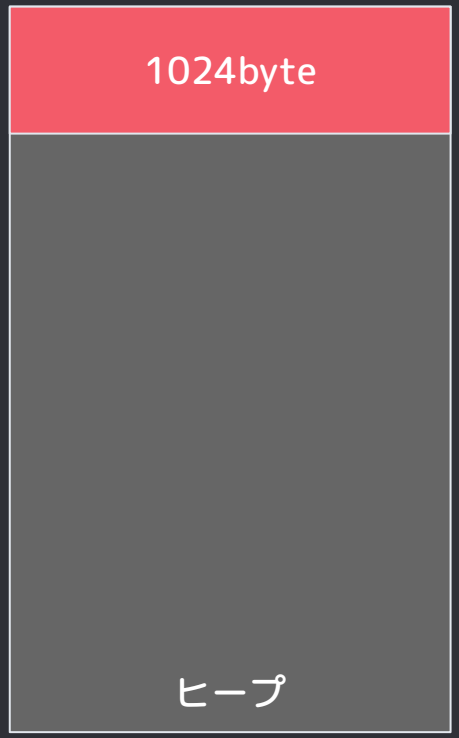


mallocでメモリ確保

```c
#include <stdlib.h>

int main(void)
{
    void* p = malloc(1024);

    memset(p, 0, 1024);

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

ヒープ領域のイメージ

ヒープ領域はプログラムの処理に応じてメモリが確保・解放される

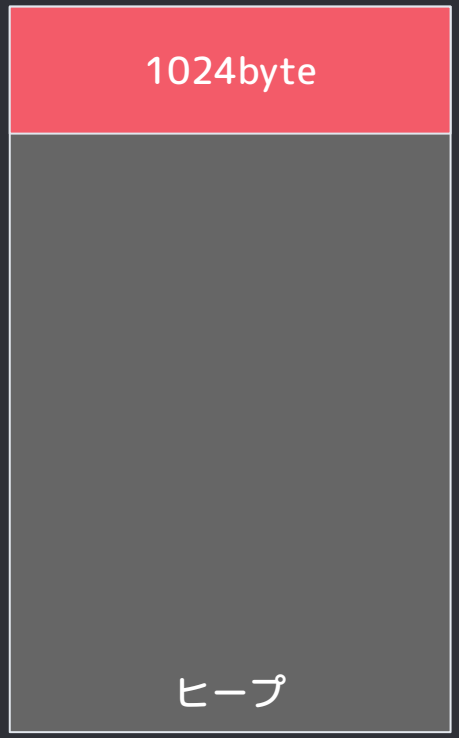


memsetで確保したメモリの値を0にしたり

```c
#include <stdlib.h>

int main(void)
{
    void* p = malloc(1024);

    memset(p, 0, 1024);

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

ヒープ領域のイメージ

ヒープ領域はプログラムの処理に応じてメモリが確保・解放される



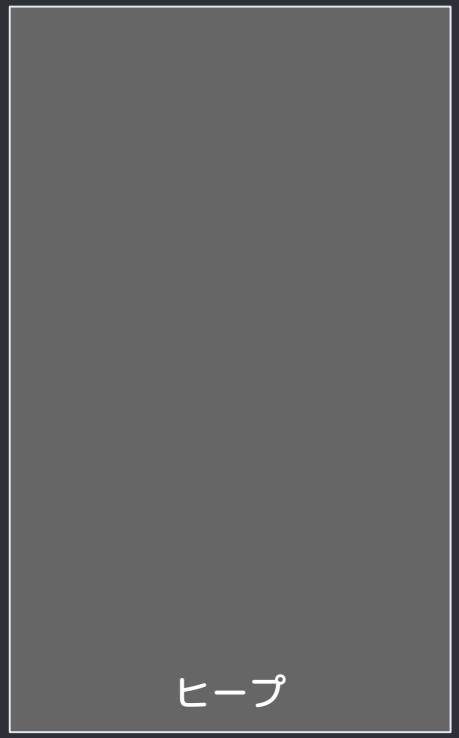
memsetで確保したメモリの値を0にしたり

```c
#include <stdlib.h>

int main(void)
{
    void* p = malloc(1024);

    memset(p, 0, 1024);

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

※確保したメモリをどう使うのかは自由

ヒープ領域のイメージ

ヒープ領域はプログラムの処理に応じてメモリが確保・解放される

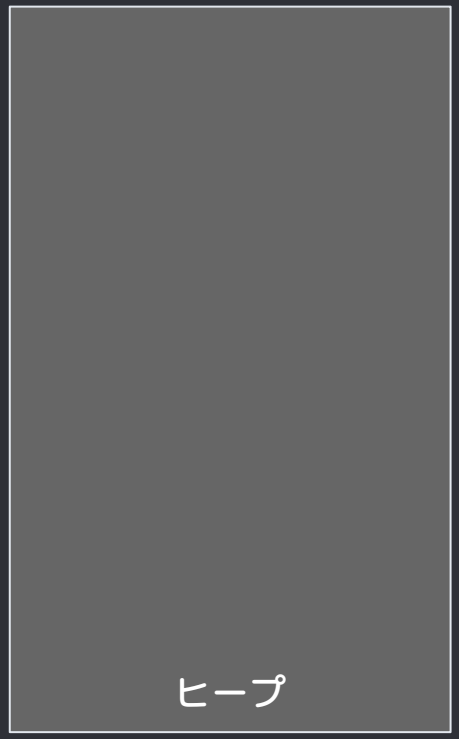


freeを呼ぶと解放される

```c
#include <stdlib.h>

int main(void)
{
    void* p = malloc(1024);

    memset(p, 0, 1024);

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

ヒープ領域のイメージ

ヒープ領域はプログラムの処理に応じてメモリが確保・解放される



freeを呼ぶと解放される

```c
#include <stdlib.h>

int main(void)
{
    void* p = malloc(1024);

    memset(p, 0, 1024);

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

※C言語でmallocなどでメモリを動的確保した場合、メモリは勝手に解放されないので、ちゃんとfree関数を使って解放してあげないといけない

ヒープ領域の利用例

ヒープ領域の利用例

- ✔ ヒープ領域の確保と解放
- ✔ voidポインタ
- ✔ 注意点
- ✔ 配列として使ってみる

ヒープ領域の確保と解放

- ✔ ヒープ領域の確保
- ✔ 確保したメモリの解放
- ✔ 確保したメモリを0でうめる

voidポインタ

- ✔ voidは「型がない」とか「役に立たない」みたいな意味合い
- ✔ 型はわからないけどとりあえずアドレスという型
- ✔ 間接参照や参照先に代入などはできない

注意点

- ✔ メモリの確保に失敗することもある
- ✔ メモリの解放忘れ(メモリリーク)に注意
- ✔ 解放したポインタには0を入れておこう

配列として使って見る

- ✔ いろんな型の配列として使ってみよう
- ✔ ポインタの型を変えるキャスト