動的メモリーの確保

山本昌志*

2007年6月5日

概 要

コンピューターがプログラムを実行する場合のメモリーがどのように使われるか—を学習する．そして，動的メモリーの割り当て方法を学ぶ．

1 前回の復習と本日の学習内容

1.1 復習

先週の学習内容は「ポインター」である．以下のようなことを学習した．

- ポインターはオブジェクトのアドレスである．オブジェクトとはメモリー上のデータ領域のことでアドレスがあり，型を持つのでサイズもある．ポインターはオブジェクトの先頭アドレスを示すことにより，オブジェクトを指し示す．実際には，ポインター変数にオブジェクトの先頭アドレスを格納している．
- つぎのようなポインターがある．
  - 構造体へのポインター．アドレス演算子&を使い先頭アドレスを取り出し，ポインターが指し示す構造体のメンバーへのアクセスは，アロー演算子 (->) を使う．
  - 配列へのポインター．配列名は，配列へのポインターを示す．
  - 関数へのポインター．関数名は，配列へのポインターを示す．
- ポインターの配列をつくることもできる．これは文字処理に有用である．

1.2 本日の学習内容

コンピューターがプログラムを実行するとき，どのようにメモリーを使うか— をはじめに説明する．つぎに，プログラムにメモリー領域を確保する方法について述べる．本日の学習のゴールは，以下のとおりである．

- メモリーにプログラムの格納の仕方が分かる．プログラム領域，データ領域，ヒープ領域，スタック領域に格納されるプログラム要素が理解できる．

*独立行政法人　秋田工業高等専門学校　電気情報工学科

- 動的メモリー確保に必要な関数 `malloc()` と `free()` 関数が使える．

教科書 [1] の pp.110–112 と 133–135 が本日の範囲である．

2　実行時のメモリーの配置

プログラムを実行するとき，命令やデータはメモリーにどのように配置されるのだろうか? ふたつの例をとおして考える．

2.1　簡単なの例

ひとつの変数—データ領域—をもつ単純なプログラムで，命令とデータのメモリーの配置を調べる．リスト 1 の例で main 関数と変数 hoge の先頭アドレスを調べる．このプログラムを実行した結果より，アドレスは図 1 のようになっていることが分かる．すなわち，次の通りである．

- 命令—関数—はメモリーの比較的，メモリー前方に格納される．
- ローカル変数のデータは比較的，メモリーの後方に格納される．

これはたまたまではなく，大体いつもこのように配置される．プログラムはメモリーの前半，ローカル変数のデータはメモリーの後半に格納される．

リスト 1: 関数とデータのアドレスを調べるプログラム．

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5    int hoge;
6
7    hoge=0x1234abcd;
8    printf("Hello world!\n");
9
10   printf("adress main:%p\n", main);
11   printf("adress hoge:%p\n", &hoge);
12
13   return 0;
14 }
```

実行結果

```
Hello world!
adress main:0x80483b4
adress hoge:0xbfacb7d0
```

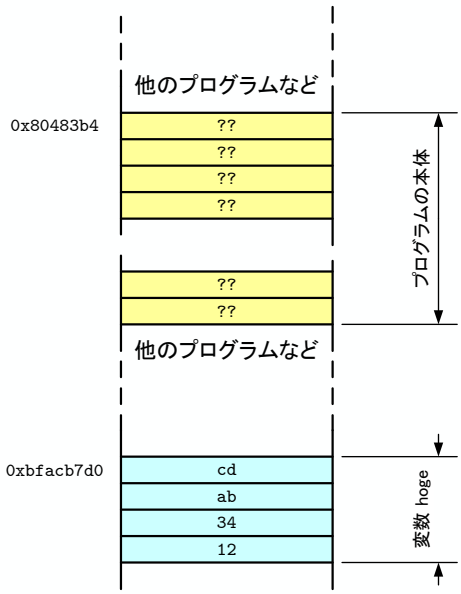

図 1: リスト 1 の場合のメモリーの様子.

2.2 複雑な例

もう少し複雑な場合，文字定数や静的変数やユーザー定義関数のアドレスなデータではどうなるであろうか? リスト 2 を実行させ，それぞれのアドレスを調べる．このプログラムを実行した結果より，アドレスは図 1 のようになっていることが分かる．すなわち，次の通りである．

- 命令—main 関数とユーザー定義関数—は，メモリーの前の方に格納される．
- 文字列定数—ここでは"HELLO"は，メモリーの前の方に格納される．
- 静的変数もメモリーの前の方に格納される．
- ローカル変数のデータは，メモリーの後ろの方に格納される．

これもたまたまではなく，大体いつもこのように配置される．プログラムや文字列定数静的変数はメモリーの前半，ローカル変数のデータはメモリーの後半に格納される．ここでは示さなかったが，グローバル変数もメモリーの前半に格納される．要するにローカル変数のみメモリーの後半に格納されるのである．

なぜ，ローカル変数のみ特別なのであろうか? ローカル変数は関数が呼び出されたときのみメモリーが割り当てられる．関数が呼ばれていないときにはメモリーが割り当てられていないのである．このようにすることにより，必要な時のみメモリーを割り当てることあでき，効率的にメモリーを使うことができる．また，再帰関数が可能となる．必要な時にメモリーを割り当てることにより，再帰呼び出しを行っても，追加でメモリーの確保ができる．

中間試験の後に学習することになるが，ローカル変数のメモリー割り当てにはスタックと呼ばれる仕組みが使われている．このようなことから，ローカル変数の領域を**スタック領域**と呼ばれる．また，プログラ

ムの関数が格納される領域を**コード領域**，グローバル変数や文字列定数が格納される領域を**データ領域**と言う．本日の講義の後半で述べる malloc() により確保されるメモリー領域を**ヒープ領域**と言う．

リスト 2: 文字定数や静的変数やユーザー定義関数のアドレス．

```
1  #include <stdio.h>
2
3  void prt(char *c);
4  int main(void)
5  {
6    static int si=9999;
7    int i=8888;
8
9    printf("main\t%p\t\n", main);
10   printf("prt\t%p\t\n", prt);
11   printf("%d\t%p\n", si, &si);
12   printf("%d\t%p\n", i, &i);
13   prt("HELLO");
14
15   return 0;
16 }
17
18 void prt(char *c)
19 {
20
21   printf("%s\t%p\n",c,c);
22 }
```

実行結果

```
main    0x8048384
prt     0x8048415
9999    0x8049620
8888    0xbff44440
HELLO   0x804850e
```

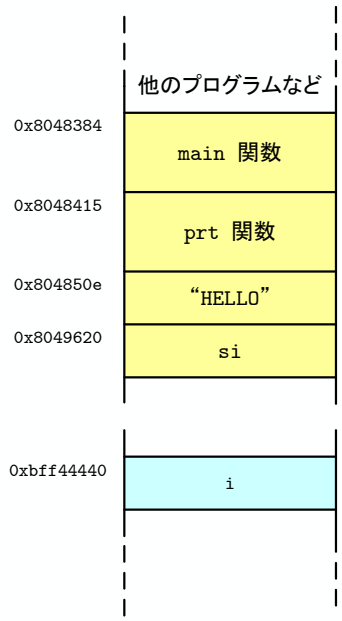

図 2: リスト 2 の場合のメモリーの様子.

3　動的なメモリーの確保

3.1　スタック領域のメモリー割り当ては小さい

大量のデータを処理するために，大きな配列をローカル変数で宣言するとコンパイルはできても，実行時にエラーとなることがある．例えば，リスト 3 のように，$1024 \times 1024 \times 10$ 個の整数型のメモリー領域を使う場合である．実行結果を見て分かるように,「セグメンテーション違反です」と表示されてプログラムが止まっている．この配列を確保するために必要なメモリーは，40[Mbyte] である．このプログラムを実行した PC のメモリーは 1[Gbyte] である．このように大きなメモリーをあるにかかわらず，たった 40[Mbyte] がユーザーが使えないなんて，おかしい!!

大きなメモリー使えない理由は，スタック領域が狭いことによる．先に述べたようにローカル変数はスタック領域を使うため，それに制限されるのである．大きなスタック領域を用意するわけにもいかない．これは，様々なプログラムのローカル変数が使う場所で，ここに大きな領域を用意すると，プログラム領域などが不足する．大きなスタック領域を用意しても，ほとんどの場合は使われないだろう．

リスト 3: 文字定数や静的変数やユーザー定義関数のアドレス.

```
1 #include <stdio.h>
2
3 int main(void)
4 {
5   int a[1024*1024*10];
```

```
6
7     a[0]=1;
8     printf("a[0]=%d\n",a[0]);
9
10    return 0;
11  }
```

実行結果

セグメンテーション違反です

3.2 malloc と free

大きな配列を使いたい場合，ヒープ領域をつかう．malloc() 関数によりメモリーを確保して，free() 関数によりメモリーを開放する．その例をリスト 4 に示す．

リスト 4: 文字定数や静的変数やユーザー定義関数のアドレス．

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main(void)
5  {
6     int *a;
7
8     a=malloc(sizeof(int)*1024*1024*10);
9
10    a[0]=1;
11    printf("a[0]=%d\n",a[0]);
12
13    free(a);
14
15    return 0;
16 }
```

実行結果

```
a[0]=1
```

malloc() 関数を使って，メモリーを確保している．ただし，この関数でいつも思い通りのメモリーを確保できるとは限らない．この関数は，メモリー確保に失敗した場合，NULL ポインターを返す．したがって，教科書のリスト 4.8 のように，エラー処理を書かなくてはならない．

3.3 メモリー配置

C 言語では大雑把に言って，コード (code)、データ (data)、ヒープ (heap)、スタック (stack) の 4 つの領域にメモリーを分けて，管理する．これらの使い分けに，プログラマーはほとんど気にする必要はない．

ただし，変数—配列や構造体を含む—を使う場合, メモリーは次のような使い方があると，プログラマーは認識しておくべきである．

- 静的変数 (static) やグローバル変数，文字列定数などは，メモリーの確保も解放もしない変数である．これらの変数はデータ領域に格納される．
- 自動変数 (auto) の場合，メモリー確保と解放は自動に行われる．自動変数は，スタック領域に格納される．
- プログラム中で malloc や calloc 関数を使い，ヒープ領域にメモリーの確保ができる．確保したメモリーを開放する場合には，free() 関数をつかう．

4 プログラム作成の練習

[**練習 1**] ユーザー定義関数がある適当なプログラムを作成して，main 関数とユーザー定義関数のアドレスを調べよ．

[**練習 2**] 適当なプログラムを作成して，様々な変数のアドレスを調べよ．

[**練習 3**] 次のようなプログラムを作成して，malloc() と free() 関数の使い方を練習せよ．

- malloc() 関数により整数型の領域 10 個を用意する．
- そこに 0〜9 の整数を代入する．
- 代入した整数を表示する．

[**練習 4**] 前問の malloc() 関数で確保したヒープ領域の先頭アドレスを示せ．

参考文献

[1] 内田智史監修, (株) システム計画研究所編. C 言語によるプログラミング　応用編　第 2 版. (株) オーム社, 2006.