プログラミング基礎演習

第8回　ポインタと文字列

担当：藤本　雄一郎（10-414室）
y_fuji@cc.tuat.ac.jp

1. 前回のおさらい

ポインタ: アドレスを保存する変数

```
int   i;
int  *p;


p = &i;
*p = 10;
printf("i = %d \n",i);
```

演算子「＊」を使うと、「指し示す先に」対して演算を行う

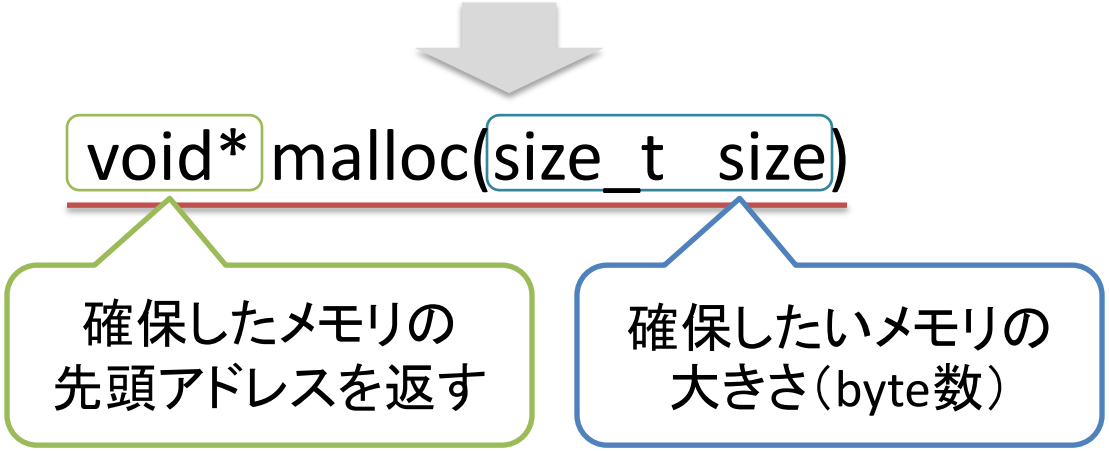
変数名	アドレス	内容
i	100	10
p	104	100
….	….	….
….	….	….

利点： メモリ上の位置を指し示すアドレスを扱うので関数等の引数に用いた場合、呼び出し元の変数を操作することができる　等

1．前回のおさらい

■動的メモリ確保

Cのライブラリには、アプリケーションの実行時にメモリを動的に確保する仕組みが用意されている



Int型データ10個分のメモリを確保したい場合の例：

```
int *pData;
pData = (int*)malloc(sizeof(int)*10);
```

※この関数はstdlib.h内でプロトタイプ宣言

復習: 文字列に関する関数

strlen(文字列)関数 : string length

文字列の文字数を返す関数

```
char  String[8] = "1234";
printf("Len=%d ¥n", strlen( String));
```

Len = 4

終端文字は
カウントされない

2. 文字列の配列

単語帳のプログラムを考える

- データは標準入力(scanf)で読み込む
- 複数の文字列の集合である
- 単語数、各単語の文字数は不定である

動的メモリ確保が有効

1文字列の文字数に応じたメモリの確保は前回行った
今回は、複数の文字列に対してメモリの確保を行う

2. 文字列の配列

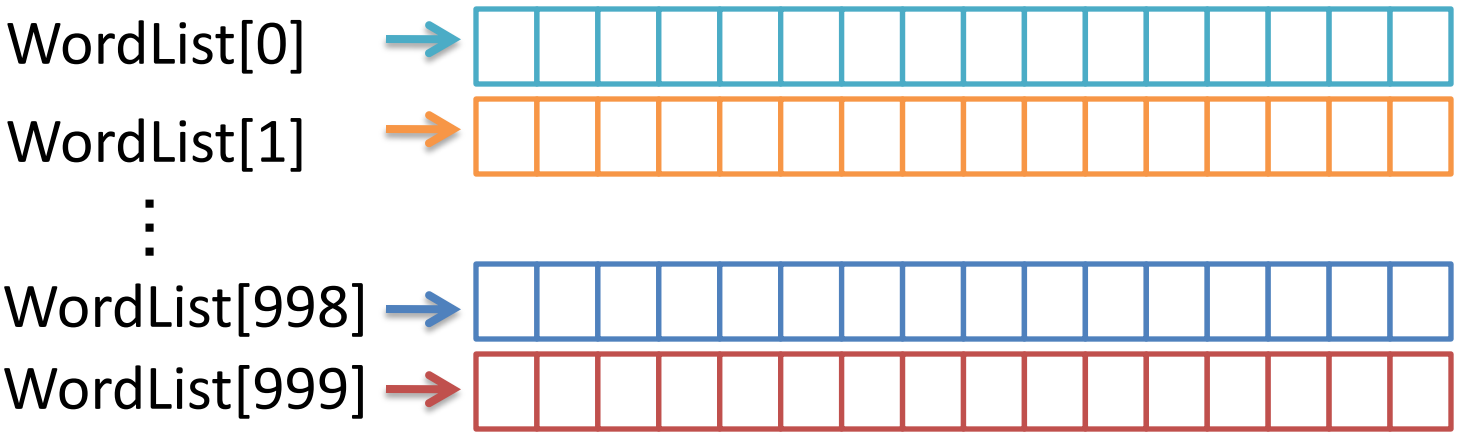
■固定サイズの単語帳つくるとすれば・・・

単語数：1000、単語の最大文字数：15　の場合

```
char   WordList[1000][16];
```

という、2次元配列で十分

■データの構造を考える



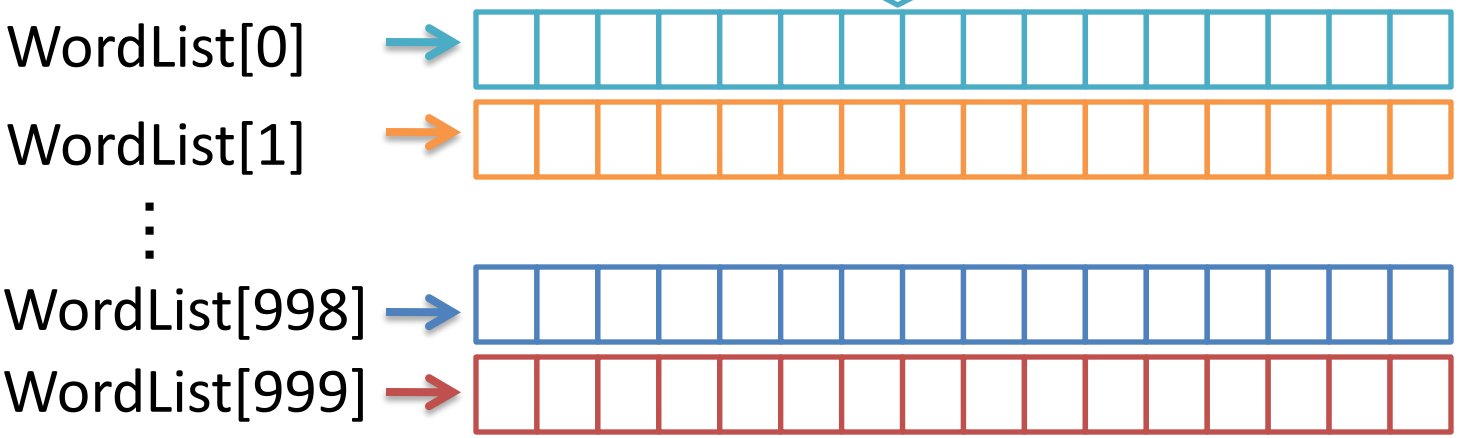
2. 文字列の配列

■固定サイズの単語帳つくるとすれば・・・

```
char   WordList[1000]
```

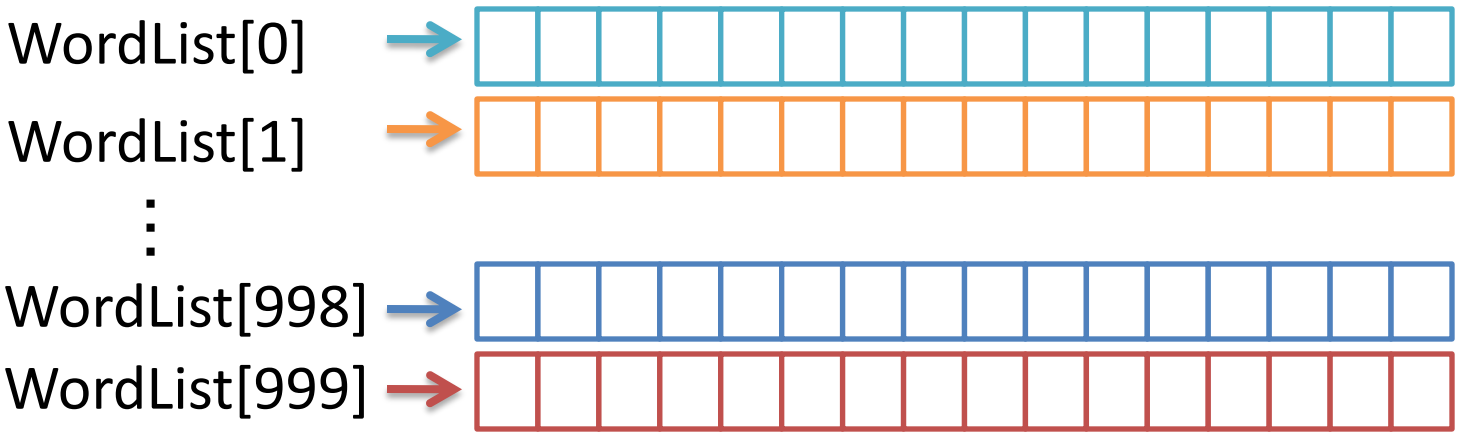
前回の課題で作成したプログラムを用いれば、各単語を格納するための領域を動的に生成できる

■データの構造を考える



2. 文字列の配列

■データの構造を考える




```
char   WordList[1000][16];

WordList[0] = (char*)malloc(50);
```

このような処理で、配列を拡張することは可能だろうか？

第6回のStepUp課題をやってみた人はこの理由が分かるはず（ヒント：配列のメモリを取る時のルール）

不可能

2. 文字列の配列

■ポインタ配列

```
char *WordList[1000];

WordList[0] = (char*)malloc(50);
```

前ページのような
「配列の拡張」
ではない

ポインタ型（例は「char *」という型）の配列を作成することで、それぞれの要素の領域を動的に作ることができる
（→アドレスを1000個格納できる．malloc関数でメモリを確保した後にその先頭アドレスをWordListの要素とする）

しかし、まだ単語数が固定（1000個）である

2. 文字列の配列

■ポインタのポインタ

前ページとは違い
単語数も動的に決定
できるようにする

「ポインタ配列」自体を動的に作成すればよい

```
char **WordList;

WordList = (char**)malloc(sizeof(char*)*1000);

WordList[0] = (char*)malloc(50);
```

char型変数へのポインタ
1000個分のメモリを確保
（＝アドレス1000個分）

50バイト分のメモリを確保し
その先頭アドレスを格納
（アドレス1000個分の1番目）

`char *` char型の配列「*」が1つつく

`char **` 「char型の配列」(char*型)の配列なので「*」が2つつく

2. 文字列の配列

ポインタのポインタ（イメージ）

メモリ初期状態
（何も定義されていない
＝Undefined）

1	2	・・・	6	・・・	573	・・・	765	・・・	1024
Undef.	Undef.	・・・	Undef.	・・・	Undef.	・・・	Undef.	・・・	Undef.

List =
(char**) malloc(sizeof(char*)*2);

char型変数へのポインタ
2個分を確保（メモリの2
～5番目，6～9番目）

1	2～5 List[0]	6～9 List[1]	・・・	573	・・・	765	・・・	1024
Undef.	Undef. （アドレス）	Undef. （アドレス）	・・・	Undef.	・・・	Undef.	・・・	Undef.

2. 文字列の配列

ポインタのポインタ（イメージ）

1	2～5 List[0]	6～9 List[1]	・・・	573	・・・	765	・・・	1024
Undef.	Undef. （アドレス）	Undef. （アドレス）	・・・	Undef.	・・・	Undef.	・・・	Undef.

List[0] =(char*) malloc(50);
List[1] =(char*) malloc(50);

573あるいは765番目から
50バイト分のメモリを確保し
その先頭アドレスを
List[0]やList[1]に格納

1	2～5 List[0]	6～9 List[1]	・・・	573 List[0][0]	・・・	622 List[0][49]	・・・	765 List[1][0]	・・・	814 List[1][49]	・・・	1024
Undef.	573 （アドレス）	765 （アドレス）	・・・	Undef. （char型の値）	・・・	Undef. （char型の値）	・・・	Undef. （char型の値）	・・・	Undef. （char型の値）	・・・	Undef.

3. 本日の課題

ポインタのポインタを利用した文字列処理

標準入力で，単語数とその単語数分の文字列を入力する．入力された文字列について，入力の逆順で出力するプログラムを作成せよ．

- 入力する各文字列の長さは1文字以上30文字以下である．
- 入力用バッファを用いて良い（後ほど説明）．ただし，文字列を添付ファイルにあるwordlistのような変数に記憶する際はmalloc関数を用い，メモリの確保を無駄に行わないようにする．

3. 本日の課題

入力の形式

入力は以下の通りである.

```
n
s_1
s_2
 ⋮
```

nは1以上の整数(実行に支障のない範囲で指定:あまり大きいとメモリに格納できる途中で止まる可能性があります.).
s_i($1 \leqq i \leqq n$)は長さ1以上30以下の文字列. 'a'-'z'と'-'からなる.

3. 本日の課題

出力の形式

s_n
s_{n-1}
⋮
s_1

入力した文字列s_i($1 \leqq i \leqq n$)について，入力した順とは逆に，
すなわちs_nから順にs_1まで出力する.
各s_iの後には改行が入る.

3. 本日の課題

入力例・出力例

入力:

```
3
abcde
how-is-your-progress
z
```

出力:

```
z
how-is-your-progress
abcde
```

3. 本日の課題

(第8回演習課題)ヒント:プログラムのイメージ

単語数の入力
- 単語数分のポインタ変数が格納できるようメモリの確保
- Wordlist = (char**) malloc ・・・

各文字列の入力
- 文字列を入力する → バッファで文字列を受け取る
- バッファで受け取った文字列の文字数をカウントし、その文字数分のメモリを確保する → strlen関数の利用
- Wordlist[0] = (char*) malloc ・・・

文字列受け取り用配列として buf[30+1]; を準備

受け取りは30文字までOKとしたが,
記憶するときは必要な分だけメモリ確保

入力された文字列について、入力の逆順で出力
- Wordlist[999]出力→Wordlist[998]出力・・・

3. 本日の課題

（第8回演習課題）補足：メモリの解放について

```
char **WordList = (char**)malloc(sizeof(char*)*1000);

WordList[0]   = (char*)malloc(50);
              ⋮
WordList[999] = (char*)malloc(50);
```

このように確保した領域を解放する場合

```
free(WordList);
```

だけでは、不十分（ほとんど解放されない）

```
free(WordList[0]);
              ⋮
free(WordList[999]);
free(WordList);
```

個々の文字列も解放する必要がある（mallocを実行した処理と同じ構造を意識する）

3. 本日の課題

■プログラムの作成（第8回講義演習課題）

- ソースコードのファイル名は次の通りにする（すべて小文字）
  （EDENアカウント）_dpointer.c
- 提出用ディレクトリ：
  /home/y_fuji/prokiso/
- コマンドでコピーを行う場合：
  cp （EDENアカウント）_dpointer.c /home/y_fuji/prokiso/

- コピーする前にアクセス権を変更しておくこと

  [sxx@xx~/prokiso] chmod 755 （EDENアカウント）_dpointer.c

- 提出期限： 12月5日（月） AM10：30（講義開始時）

4. 次回について

第9回（12月5日（月））
「ポインタ配列を用いたソート」