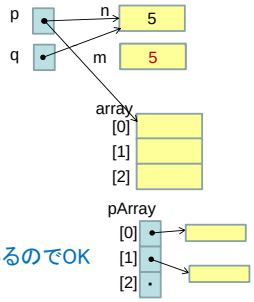
ポインタの基礎

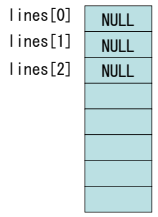


int* p;　int 型の変数を指すポインタ
int* q;　int 型の変数を指すポインタ
int　n=5, m=7;　int 型の変数
int　array[3];
int* pArray[3];

p = &n;　ポインタにアドレスを代入しているのでOK
p = q;　ポインタ同士の代入なのでOK
p = m;　NG
m = p;　NG
m = (*p); OK. pが指している物の中身を参照してmに代入.
p = array; OK. 配列の先頭アドレスを代入
p = array[0];　NG. pはポインタ. array[0]はint型

文字列へのポインタの配列　(76行目)

```
static char *lines[MAXLINES];
```



lines の各要素lines[0], lines[1],…の値は,
各要素が指す文字列領域の先頭アドレスである.
つまり, ポインタとは, 指し示すメモリ領域の
先頭アドレスを格納している.

文字列へのポインタ
　char* p;
ポインタはアドレスを格納する変数.
文字列へのポインタpは、pの指す文字列領域の先頭アドレスを格納.

malloc 関数で確保したメモリ領域のアドレスを，ReadLines関数の引数として渡された文字列へのポインタの配列に順に格納　(26行目)

int cnt: 初期値cnt=0からcnt++することで, lines[0], lines[1]…と次の行へ移動
lines[cnt]: 文字列へのポインタ.
lines[cnt] = malloc(strlen(buf) + 1);

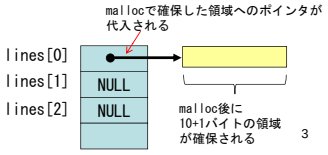
static char buf[LINELENGH]: 標準入力(キーボード)で入力した文字列が入る領域.

strlen(buf): bufの指す文字列の長さ.
　　引数に文字列へのポインタbufを取り, 文字数(¥0は含まない)を返す.

malloc(strlen(buf)+1): 文字列の長さに合わせてメモリ領域を確保する.
　　ただし, C言語における文字列は, ¥0で終端する
　　必要があるため, (文字数＋１)バイトのメモリ領域が必要.

```
char buf[] = "strawberry";

strlen(buf)は10を返す.
```



文字列のコピー　(31行目)

lines[cnt]: 文字列へのポインタ.
buf: 標準入力(キーボード)から読み込んだ文字列へのポインタ.
strcpy(lines[cnt], buf): 文字列のコピー.
　　第1引数にコピー先文字列領域へのポインタlines[cnt],
　　第2引数にコピー元文字列へのポインタbufを取り,
　　bufの指すコピー元文字列からlines[cnt]の指すコピー先
　　文字列領域にコピー.

例　strcpy(lines[2], buf);

文字列のコピー　(31行目)

lines[cnt]: 文字列へのポインタ.
buf: 標準入力(キーボード)から読み込んだ文字列へのポインタ.
strcpy(lines[cnt], buf): 文字列のコピー.
　　第1引数にコピー先文字列領域へのポインタlines[cnt],
　　第2引数にコピー元文字列へのポインタbufを取り,
　　bufの指すコピー元文字列からlines[cnt]の指すコピー先
　　文字列領域にコピー.

例　strcpy(lines[2], buf);

文字列の大小を比較（辞書順）　strcmpの使い方

const char *string1: 文字列string1の**ポインタ**.
const char *string1: 文字列string2の**ポインタ**.

int strcmp(const char *string1, const char *string2): 文字列の大小比較.
　　第1引数に文字列string1の**ポインタ**,
　　第2引数に文字列string2の**ポインタ**を取り,
　　文字列string1, string2を辞書順に大小比較.
　　戻り値は,
　　string1>string2なら正の数,
　　string1<string2なら負の数,
　　string1=string2なら0である.

これは関数の宣言.
引数はconst charの**ポインタだよ**
という意味.

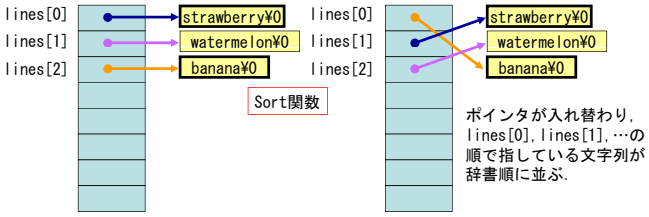
例)
```
char *string1 = "strawberry";
char *string2 = " watermelon";

strcmp(string1, string2); は string1＜string2なので負の数を返す.
```

これは関数の呼び出し.
文字列のポインタである　string1 が引数に入っている.

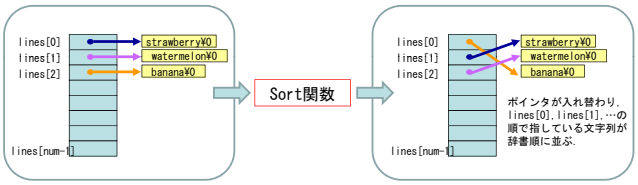
Sort関数：複数の文字列を辞書順にソート　(65行目～)

文字列をソートするには，lines の各要素lines[0],lines[1],…の指している文字列どうしをstrcmp関数で比較して辞書順に整列するように，各要素の値，つまり，**ポインタの中身を入れ替えればよい.**



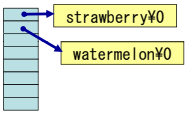
ポインタが入れ替わり，lines[0],lines[1],…の順で指している文字列が辞書順に並ぶ.

ポインタの指している文字列をstrcmp関数で比較して辞書順に整列するようにポインタの中身の入れ替えを行っているだけなので，文字列のメモリ上の位置は変わらない.
つまり，linesの各要素の値である**各文字列の先頭アドレスだけ入れ替える.**

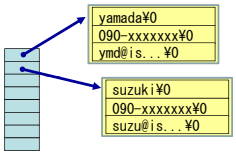


文字列へのポインタの配列を使うと、実際に文字列を動かさなくても並べ替えの働きを実現できる.

また，文字列に限らずXXXへのポインタの配列を使うことで，同様にXXXを並べ替えることもできる.



この例では文字列だったが



構造体の並べ替えにも応用可