ポインタ変数Ⅱ

０．目次

６．ポインタ変数と文字処理

６．１　文字

文字の宣言	char ch
文字の入力	scanf("%c",&ch)
文字とコード	文字はそのまま８ビットの数値（文字コード）とみなされる。
文字定数	文字定数は'で囲む。

●プログラム（ptr611.c）

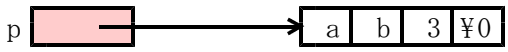
```
 1  /* << ptr611.c >> */
 2  #include <stdio.h>
 3  int main() {
 4    int i;
 5    char ch; /* 文字の宣言 */
 6    scanf("%c",&ch); /* 文字の入力 */
 7    printf("文字       : %c ¥n",ch);
 8    printf("文字コード : %d ¥n",ch);
 9    ch = '0'; /* 文字定数 */
10    printf("文字       : %c ¥n",ch);
11    printf("文字コード : %d ¥n",ch);
12    ch = '¥0'; /* 文字定数 */
13    printf("文字       : %c ¥n",ch);
14    printf("文字コード : %d ¥n",ch);
15  }
```

実行結果

```
% cc ptr611.c
% ./a.out
1
文字       : 1
文字コード : 49
文字       : 0
文字コード : 48
文字       :
文字コード : 0
```

６．２　文字列定数

文字列定数の宣言	**char *p** 文字列定数は、アドレスで示されるのでポインタ変数（アドレスを格納するための変数）を使う。 ポインタ変数pで指されたアドレスから順に文字が格納され、最後に文字コード0の文字（'¥0'で表す）が格納される。ヌル文字と呼ばれる。 文字列定数のi番目の文字は、*(p+i)で取得する。
文字列定数	文字列定数は"で囲む。"ab3"
文字列定数の入力	代入文　p = "abc"



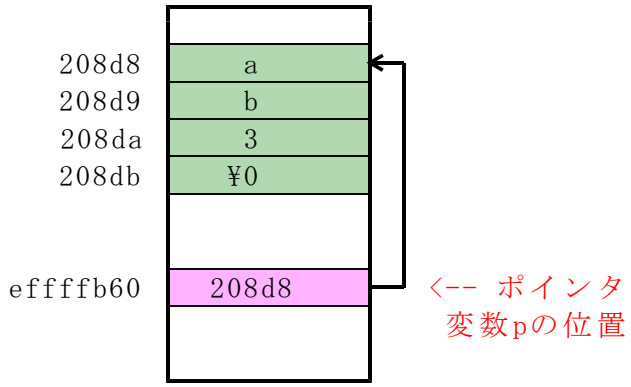
●プログラム（ptr621.c）

```
1  /* << ptr621.c >> */
2  #include <stdio.h>
3  int main() {
4    int i;
5    char *p,*q;    /* ポインタ変数の宣言 */
6
7    /* 代入。*/
8    p = "ab3"; /* 文字列ab3を記憶場所に保存しその先頭の
9                  アドレスをポインタ変数pに格納する。*/
10   /* 文字列定数の出力 */
11   printf("文字列定数：%s ¥n",p);
12   printf("p+0:%x  *(p+0):|%c| ¥n",p+0,*(p+0));
13   printf("p+1:%x  *(p+1):|%c| ¥n",p+1,*(p+1));
14   printf("p+2:%x  *(p+2):|%c| ¥n",p+2,*(p+2));
15   printf("p+3:%x  *(p+3):|%c| ¥n",p+3,*(p+3));
16   printf("&p:%x ¥n",&p);
17 }
```

実行結果

```
% cc ptr621.c
% ./a.out
文字列定数 : ab3
p+0:208d8  *(p+0):|a|
p+1:208d9  *(p+1):|b|
p+2:208da  *(p+2):|3|
p+3:208db  *(p+3):||
&p:effffb60
```

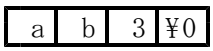
記憶領域



処理手順

①**char *p** でポインタ変数pが確保される。

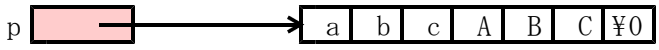
②**"ab3"**で文字列ab3が記憶場所に保存される。



③**p = "ab3"**で文字列先頭のアドレスがポインタ変数pに格納される。

６．３　文字列

文字列の宣言	**char p[7]** 文字列は配列pに格納される。 配列名はポインタ変数である。
文字列の入力	**scanf("%s",p)** 読み込んだ文字は、配列要素のp[0],p[1],p[2]の順に格納され、最後に文字コード0の文字('¥0'で表す)が格納される。 したがって、配列の大きさは、読み込む文字列の長さ+1以上でなければならない。



●プログラム（ptr631.c）

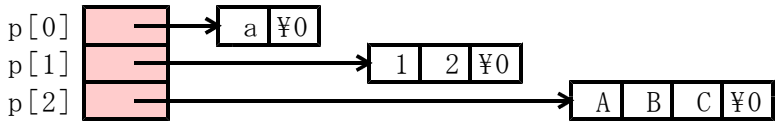
```
 1  /* << ptr631.c >> */
 2  #include <stdio.h>
 3  int main() {
 4    int i;
 5    char p[7];      /* 文字列の宣言 */
 6    scanf("%s",p); /* 文字列の入力 */
 7    /* 文字列の出力 */
 8    printf("文字列：|%s| ¥n",p);    /* 文字列の長さ分の桁に出力*/
 9    printf("文字列：|%8s| ¥n",p);   /* ８桁の中に右寄せで出力  */
10    printf("文字列：|%-8s| ¥n",p); /* ８桁の中に左寄せで出力  */
11    /* １文字ずつの処理 */
12    for( i=0; i<7; i++ ) {
13      printf("p[%d]  ",i);    /* 配列要素 */
14      printf("|%d|  ",p[i]); /* 10進数   */
15      printf("|%x|  ",p[i]); /* 16進数   */
16      printf("|%c|  ",p[i]); /* 文字     */
17      printf("¥n");
18    }
19  }
```

実行結果

```
% cc ptr631.c
% ./a.out
abcABC
文字列：|abcABC|
文字列：|  abcABC|
文字列：|abcABC  |
p[0]  |97|  |61|  |a|
p[1]  |98|  |62|  |b|
p[2]  |99|  |63|  |c|
p[3]  |65|  |41|  |A|
p[4]  |66|  |42|  |B|
p[5]  |67|  |43|  |C|
p[6]  |0|  |0|  ||
```

６．４　文字列配列

文字列配列の宣言	**char *p[3]** ポインタ変数が3個用意される。
文字列の入力	・代入文 **p[0] = "abc"** 読み込んだ文字は、記憶場所に保存され、最後に文字コード0の文字('¥0'で表す)が格納される。 先頭の番地がp[0]に格納される。



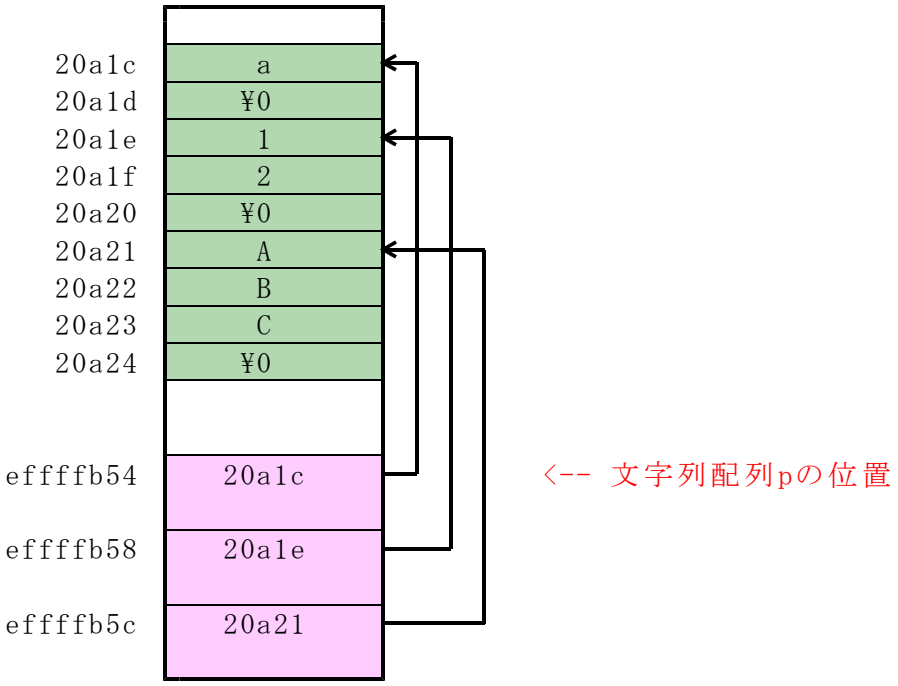
●プログラム（ptr641.c）

```
1  /* << ptr641.c >> */
2  #include <stdio.h>
3  int main() {
4    int i,j;
5    char *p[3]; /* 文字列配列の宣言 */
6    /* 文字型配列への代入 */
7    p[0] = "a"; /* 文字列aを記憶場所に保存しその先頭の
8                   アドレスをp[0]に格納する。*/
9    p[1] = "12";
10   p[2] = "ABC";
11   /* 文字列配列要素 */
12   for( j=0; j<=2; j++ ) {
13     printf("%d番目の文字列：%s¥n",j,p[j]);
14     i = 0;
15     while( 1 ) {
16       printf("p[%d]+%d:%x  ",j,i,p[j]+i); /* 番地 */
17       printf("*(p[%d]+%d):|%c| ¥n",j,i,*(p[j]+i)); /* 文字 */
18       if( *(p[j]+i) == '¥0' ) { break; }
19       i++;
20     }
21   }
22   printf("p:%x¥n",p);
23   printf("&p:%x¥n",&p);
24   for( j=0; j<=2; j++ ) { printf("&p[%d]:%x¥n",j,&p[j]); }
25 }
```

実行結果

```
% cc ptr641.c
% ./a.out
0番目の文字列 : a
p[0]+0:20a1c  *(p[0]+0):|a|
p[0]+1:20a1d  *(p[0]+1):||
1番目の文字列 : 12
p[1]+0:20a1e  *(p[1]+0):|1|
p[1]+1:20a1f  *(p[1]+1):|2|
p[1]+2:20a20  *(p[1]+2):||
2番目の文字列 : ABC
p[2]+0:20a21  *(p[2]+0):|A|
p[2]+1:20a22  *(p[2]+1):|B|
p[2]+2:20a23  *(p[2]+2):|C|
p[2]+3:20a24  *(p[2]+3):||
p:effffb54
&p:effffb54
&p[0]:effffb54
&p[1]:effffb58
&p[2]:effffb5c
```

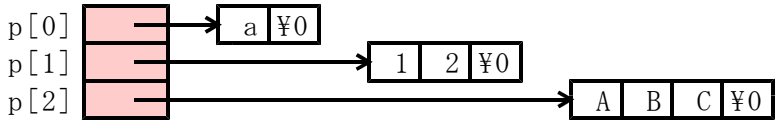
記憶領域



処理手順

①**char *p[3]**によって、ポインタ変数を要素とする配列pが確保される。
要素数は３個。

②**p[0] = ″a″** で文字列aが記憶場所に保存され、先頭のアドレスがp[0]に格納される。p[1],p[2]も同様。



ポインタ変数の配列の応用例（引数の取り込み）

●プログラム（ptr651.c）

```
1  /* << ptr651.c >> */
2  #include <stdio.h>
3  int main (int argc, char *argv[]) {
4    int i;
5    printf("argc = %d¥n",argc);
6    for( i=0; i<argc; i++ ) {
7      printf("argv[%d] = %s¥n",i,argv[i]);
8    }
9  }
```

実行結果

```
% cc ptr651.c
% ./a.out p1 p2
argc = 3
argv[0] = ./a.out
argv[1] = p1
argv[2] = p2
```

７．ポインタ変数と関数

７．１　引数とポインタ変数

変数を引数とすると、値が関数側に渡され関数内での変更は呼び出した側には影響を与えない。

ポインタ変数を引数とすると、番地が関数側に渡されるので関数内での変更は呼び出した側に影響を与えることになる。

７．１．１　変数が引数の場合

●プログラム（ptr711.c）

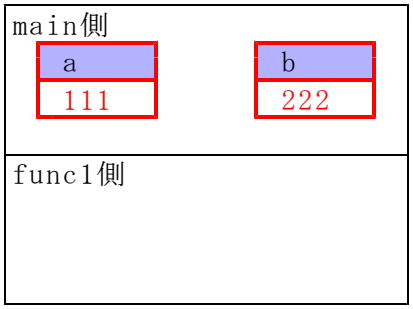
```
/* << ptr711.c >> */
#include <stdio.h>
int main () {
  int a,b;
  void func1(int x, int y);
①  a = 111; b = 222;
  printf("main: a=%d b=%d¥n",a,b);
②  func1(a,b);
  printf("main: a=%d b=%d¥n¥n",a,b);
}
void func1(int x, int y) {
③  x = -x; y = -y;
  printf("func1: x=%d y=%d¥n",x,y);
}
```

実行結果

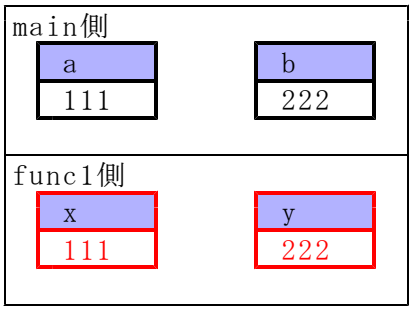
```
main: a=111 b=222
func1: x=-111 y=-222
main: a=111 b=222
```

●動作

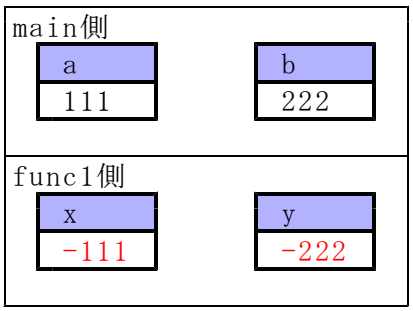
① a = 111; b =222;



② func1(a,b);



③ x = -x; y = -y;



７．１．２　ポインタ変数が引数の場合

●プログラム（ptr712.c）

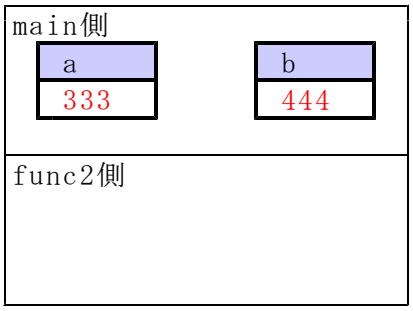
```
/* << ptr712.c >> */
#include <stdio.h>
int main () {
  int a,b;
  void func2(int *a, int *b);
①  a = 333; b = 444;
  printf("main: a=%d b=%d¥n",a,b);
②  func2(&a,&b);
  printf("main: a=%d b=%d¥n",a,b);
}
void func2(int *x, int *y) {
③  *x = -*x; *y = -*y;
  printf("func2: *x=%d *y=%d¥n",*x,*y);
}
```

実行結果

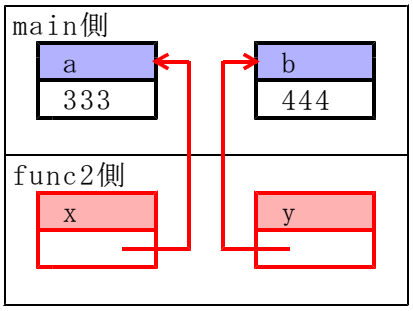
```
main: a=333 b=444
func2: *x=-333 *y=-444
main: a=-333 b=-444
```

●動作

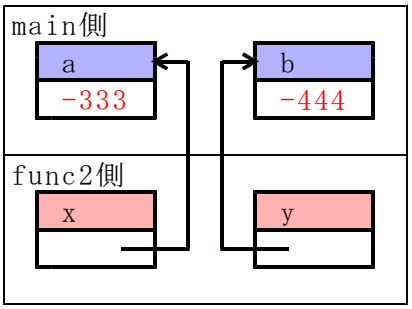
① a = 333; b = 444;



② func2(&a,&b);



③ *x = -*x; *y = -*y;



７．２　引数と配列

配列を引数とすると、配列の先頭番地が関数側に渡される。関数内で別の配列が作成されていないことに注意。また、関数内での変更は呼び出した側に影響を与えることになる。

●プログラム（ptr721.c）

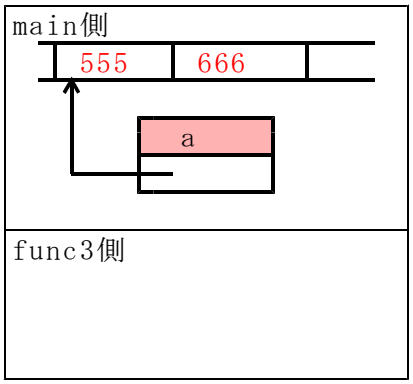
```
/* << ptr721.c >> */
#include <stdio.h>
int main () {
  int a[9];
  void func3(int x[]);
①  a[0] = 555; a[1] = 666;
  printf("main: a[0]=%d a[1]=%d¥n",a[0],a[1]);
②  func3(a);
  printf("main: a[0]=%d a[1]=%d¥n¥n",a[0],a[1]);
}
void func3(int x[]) {
③  x[0] = -x[0]; x[1] = -x[1];
  printf("func3: x[0]=%d x[1]=%d¥n",x[0],x[1]);
}
```

実行結果

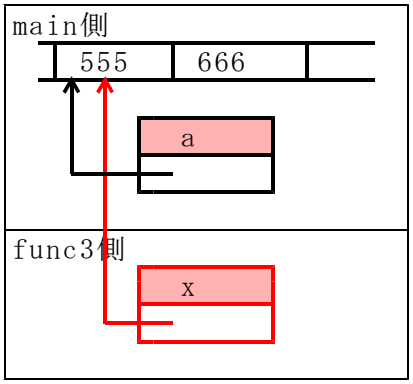
```
main: a[0]=555 a[1]=666
func3: x[0]=-555 x[1]=-666
main: a[0]=-555 a[1]=-666
```

●動作

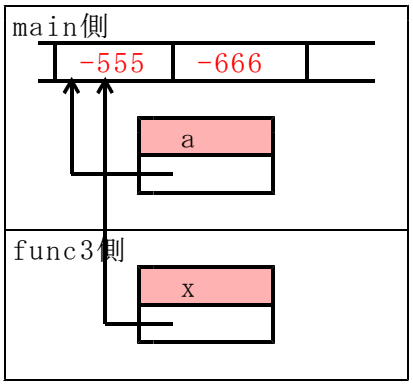
① a[0] = 555; a[1] = 666;



② **func3(a);**



③ **x[0] = -x[0]; x[1] = -x[1];**



７．３　戻り値とポインタ変数

戻り値としてポインタを指定できる。

●プログラム（ptr731.c）

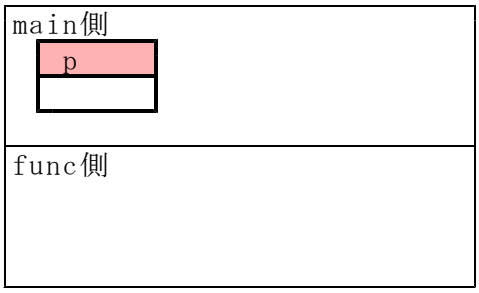
```
/* << ptr731.c >> */
#include <stdio.h>
int main () {
①   char *p;
    char *func(); /* 文字型へのポインタを返す関数func。*/
②   p = func();
    printf("main: p=%x¥n",p);
    printf("main: %s¥n",p);
}
char *func() {
    char *q;
    q = "abc";
    printf("func: q=%x¥n",q);
    return(q);
}
```

実行結果

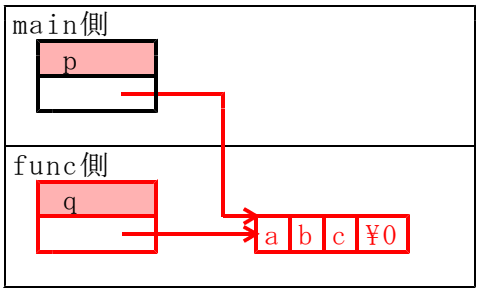
```
func: q=40064e
main: p=40064e
main: abc
```

●動作

① char *p;



② p = func();



８．問題

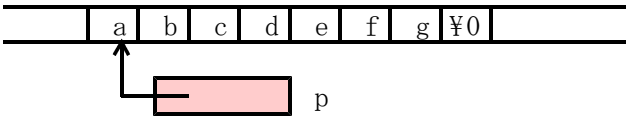
問題１

（１）文字列定数の長さを出力するプログラムを作成せよ。

```
 1  /* << ptr811.c >> */
 2  #include <stdio.h>
 3  int main () {
 4    int i;
 5    char *p;
 6    p = "abcdefg";
 7    i = 0;
 8    while( *(p+i) != ①        ) { i++; };
 9    printf("文字列定数 %s の長さ：%d¥n",p,i);
10  }
```

実行結果

```
% cc ptr811.c
% ./a.out
文字列定数 abcdefg の長さ：7
```

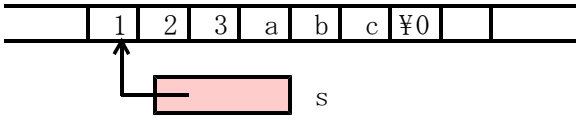


（２）１つの文字列を読み込み、文字列の長さを出力するプログラムを作成せよ。

```
 1  /* << ptr812.c >> */
 2  #include <stdio.h>
 3  int main () {
 4    int i;
 5    char s[100];
 6    scanf("%s",s);
 7    i = 0;
 8    while( s[i] != ②        ) { i++; };
 9    printf("文字列 %s の長さ：%d¥n",s,i);
10  }
```

実行結果

```
% cc ptr812.c
% ./a.out
123abc
文字列 123abc の長さ：6
```



問題２　文字列の分離

（１）入力文字列をピリオドで分離するプログラムを完成せよ。
　　ただし、処理前と処理後で入力文字列は変わらない。

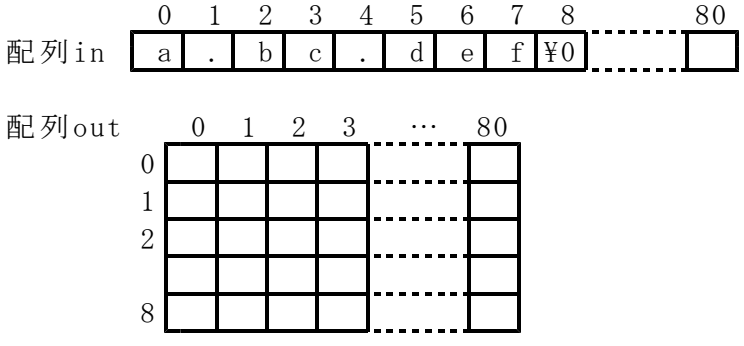
●プログラム（ptr821.c）

```
 1  /* << ptr821.c >> */
 2  #include <stdio.h>
 3  #include <string.h>
 4  int main() {
 5    int i,j,k,
 6        len,          /* 入力した文字列の長さ */
 7        m;            /* 分離した文字列の個数 */
 8    char in[81],      /* 入力用配列の宣言 */
 9         out[9][81],  /* 出力用配列の宣言 */
10         del;         /* 分離文字 */
11
12    /* 文字列の入力 */
13    scanf("%s",in);
14    printf("処理前文字列(in) : %s ¥n",in);
15    del = '.';
16    printf("分離文字 : %c ¥n",del);
17
18    /* 初期設定。*/
19    len = strlen(in);
20    j = 0;
21    m = 0;
22
23    /* 分離処理。 */
24    for( i=0; i<len; i++ ) {
25      if( in[i] != del ) {
26        /* 文字in[i]が分離文字でない場合。*/
27        out[m][j] = ③          ;
28        j = j + 1;
29      } else {
30        /* 文字in[i]が分離文字である場合。*/
31        out[m][j] = ④          ;
32        j = 0;
33        m = m + 1;
34      }
35    }
36
37    /* 最後の文字がピリオドでない場合、ヌル文字を追加する。*/
38    if( in[len-1] != del ) {
39      out[m][j] = '¥0';
40      m = m + 1;
41    }
42
43    /* 分離された文字列の表示。*/
44    for( k=0; k<m; k++ ) {
45      printf("分離された文字列(%d) : %s ¥n",k,out[k]);
46    }
47    printf("処理後文字列(in) : %s ¥n",in);
48  }
```

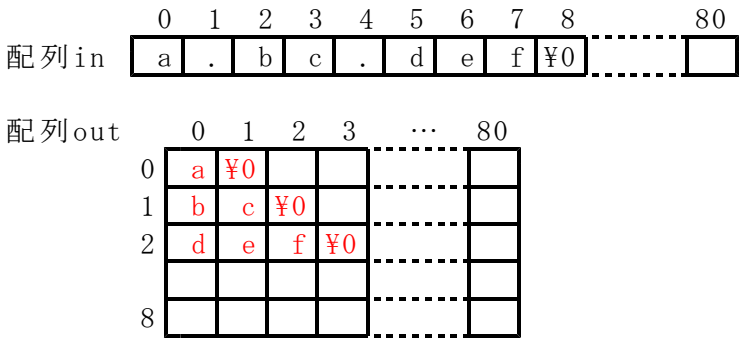
実行結果

```
% cc ptr821.c
% ./a.out
a.bc
処理前文字列(in) : a.bc
分離文字 : .
分離された文字列(0) : a
分離された文字列(1) : bc
処理後文字列(in) : a.bc
% ./a.out
a.bc.
処理前文字列(in) : a.bc.
分離文字 : .
分離された文字列(0) : a
分離された文字列(1) : bc
処理後文字列(in) : a.bc.
% ./a.out
.a.bc.
処理前文字列(in) : .a.b
分離文字 : .
分離された文字列(0) :
分離された文字列(1) : a
分離された文字列(2) : b
処理後文字列(in) : .a.b
```

処理前



処理後



（２）入力文字列をピリオドで分離するプログラムを完成せよ。
ただし、処理後、入力文字列は変わる。

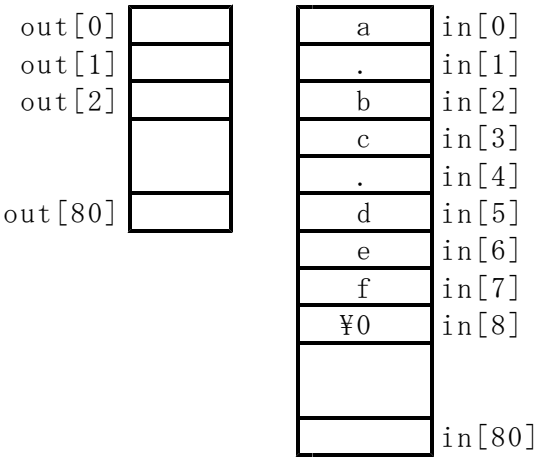
●プログラム（ptr822.c）

```
1  /* << ptr822.c >> */
2  #include <stdio.h>
3  #include <string.h>
4  int main() {
5    int i,k,
6        len,        /* 文字列の長さ */
7        m;          /* 分離した文字列の個数 */
8    char in[81],    /* 入力用配列の宣言 */
9         *out[81], /* 出力用配列の宣言 */
10        del;       /* 分離文字 */
11
12   /* 文字列の入力 */
13   scanf("%s",in);
14   printf("処理前文字列(in) : %s ¥n",in);
15   del = '.';
16   printf("分離文字 : %c ¥n",del);
17
18   /* 初期設定。*/
19   len = strlen(in);
20   m = 0;
21   out[0] = in;
22
23   /* 分離処理。*/
24   for( i=0; i<len; i++ ) {
25     if( in[i] == del ) {
26       in[i] = ⑤          ;
27       m = m + 1;
28       out[m] = ⑥          ;
29     }
30   }
31
32   /* 最後の文字がピリオドでない場合、ヌル文字を追加する。*/
33   if( in[len-1] != '¥0' ) {
34     in[len] = '¥0';
35     m = m + 1;
36   }
37
38   /* 分離された文字列の表示。*/
39   for( k=0; k<m; k++ ) {
40     printf("分離された文字列(%d) : %s ¥n",k,out[k]);
41   }
42   printf("処理後文字列(in) : %s ¥n",in);
43 }
```

実行結果

```
% cc ptr822.c
% ./a.out
a.bc
処理前文字列(in) : a.bc
分離文字 : .
分離された文字列(0) : a
分離された文字列(1) : bc
処理後文字列(in) : a
% a.out
a.bc.
処理前文字列(in) : a.bc.
分離文字 : .
分離された文字列(0) : a
分離された文字列(1) : bc
処理後文字列(in) : a
% a.out
.a.bc.
処理前文字列(in) : .a.bc
分離文字 : .
分離された文字列(0) :
分離された文字列(1) : a
分離された文字列(2) : bc
処理後文字列(in) :
```

処理前



処理後