

I117 (4) 文字、文字列処理 (その2)

知念

北陸先端科学技術大学院大学 情報科学研究科
School of Information Science,
Japan Advanced Institute of Science and Technology

左／右詰め

多くの場合はこれで足りる

```
sprintf(msg, "%-15s", "abc");  
printf("|%s|\n", msg);  
sprintf(msg, "%15s", "xyz");  
printf("|%s|\n", msg);
```

結果

```
|abc|  
|          xyz|
```

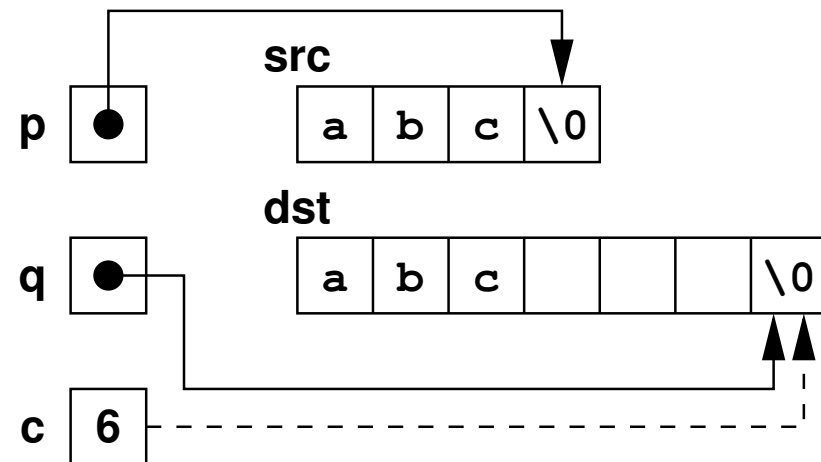
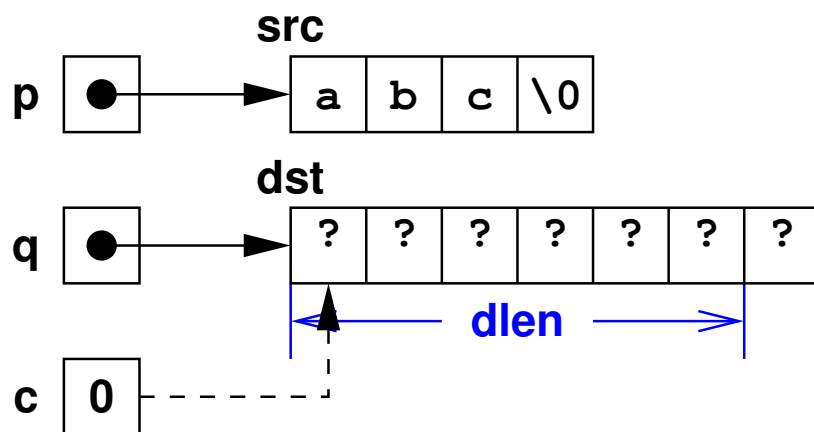
仕組みを考えてみよう

左詰め（その1） — 長さで繰り返す

```
int leftjust(char *dst, int dlen, char *src) {
    char *p, *q;
    int c;
    p = src; q = dst; c = 0;
    while(c<dlen && *p) {
        *q++ = *p++; c++;
    }
    while(c<dlen) {
        *q++ = ' '; c++;
    }
    *q = '\\0';
}
```

左詰め（その1） — 長さで繰り返す (*cont.*)

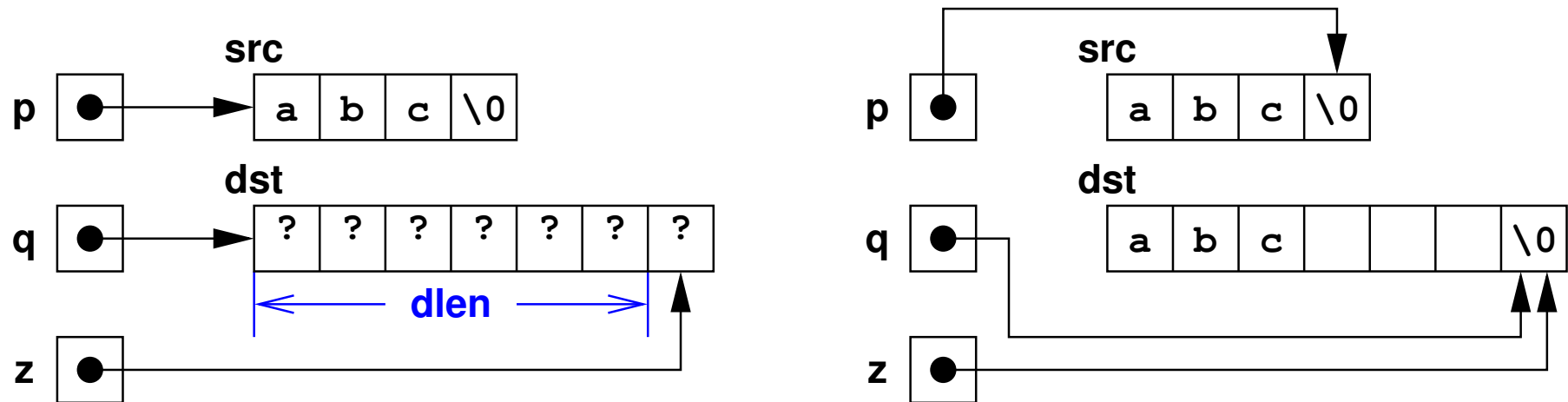
- 長さで繰り返す
 - ◇ 長さを数える
 - ◇ src を複写、残りは ' '(空白) を複写
- dst は $dlen+1$ の長さが必要



左詰め（その2） — 場所で繰り返す

```
int leftjust(char *dst, int dlen, char *src) {
    char *p, *q, *z;
    p = src; q = dst; z = dst+dlen;
    while(q<z && *p) {
        *q++ = *p++;
    }
    while(q<z) {
        *q++ = ' ';
    }
    *q = '\0';
}
```

左詰め（その2） — 場所で繰り返す (*cont.*)



- 長さではなく場所でループを表現
- ループ中の演算が少ない

コンマ（位どり）

入力	出力	コンマの数
1	1	0
12	12	0
123	123	0
1234	1,234	1
12345	12,345	1
123456	123,456	1
1234567	1,234,567	2

コンマの数 = (入力長 - 1) / 3

出力長 = 入力長 + (入力長 - 1) / 3

コンマ (位どり) (*cont.*)

```
int addcomma(char *dst, int dlen, char *src)
{
    char *p, *q;
    int iw, ow;
    iw = 0;
    p = src;
    while(*p) {
        p++; iw++;
    }
    ow = iw+(iw-1)/3;
    if(ow>dlen-1)
        return -1;
}
```


コンマ (位どり) (*cont.*)

```
p = src; q = dst;
while(iw>0) {
    if(iw%3==0 && q!=dst) {
        *q++ = ',';
    }
    *q++ = *p++;
    iw--;
}
*q = '\0';

return 0;
}
```

範囲指定 ハイフン展開

ハイフン (0x2d) で連結した数字の表記を展開

3-6 => 3, 4, 5, 6

- 「数字」「ハイフン」「数字」の順に登場する
- 「ハイフン」がなければそこで打ちきり
- 二つ目の数字がなければ一つ目と同じとする

3 => 3

3- => 3

数字を取り出す処理をマクロに、p は展開場所依存

```
#define takenum(dst) { \
    char  tmp[BUFSIZ]; \
    char  *q=tmp; \
    int    c=0; \
    while(c<BUFSIZ && *p && *p>='0' && *p<='9') {\
        *q++ = *p++; \
    } \
    *q = '\0'; \
    if(*tmp) { \
        dst = atoi(tmp); \
    } \
}
```

```
int hexp(char *src) {  
    char *p = src;  
    int    benum = -1, ennum = -1;  
    int    i;  
  
    takenum(benum);          /* beginning number */  
    if(*p=='-') {  
        p++;  
        takenum(ennum); /* ending number */  
    }  
    if(ennum<=0) {  
        ennum = benum;  
    }  
}
```

```
    for(i=benum;i<=ennum;i++) {  
        printf("%d", i);  
        if(i!=ennum) {  
            printf(",");  
        }  
    }  
    return 0;  
}
```

最後以外はコンマ (0x2c) を挿入

範囲指定（その2）コンマ展開

ハイフンより大きな範囲指定としてコンマを導入

3-6, 9-10 => 3, 4, 5, 6, 9, 10

コンマ以外を先の `hexp()` で処理

```
hexp("3-6");  
hexp("9-10");
```

```
while(*p) {  
    c = 0;    q = tmp;  
    while(c<BUFSIZ && *p && *p!=' , ' ) {  
        *q++ = *p++;  
    }  
    *q = '\0';  
    hexp(tmp);  
    if(*p==' , ') {  
        printf(",");  
        p++;  
    }  
}
```

範囲指定 圧縮

2, 3, 4, 9, 10 => 2-4, 9-10

数字を整数に変換した後、配列に蓄える

```
int isfirst = 1;
int benum = -7, ennum = -7;
int listuse = 0;
int numlist[MAXLIST], i;
for(i=0; i<MAXLIST; i++)
    numlist[i] = 0;
```



```

while(*p) {
    c = 0; q = tmp;
    while(c<BUFSIZ && *p && *p>='0' && *p<='9') {
        *q++ = *p++;
    }
    *q = '\0';
    if(*tmp) {
        curnum = atoi(tmp);
        numlist[listuse++]
            = curnum;
    }
    if(*p==',') { p++; }
    else { break; }
}

```

listuse

5

numlist

2	3	4	9	10				
---	---	---	---	----	--	--	--	--

benum

-7

ennum

-7

出力のマクロ

```
#define P \  
    if(benum>=0) { \  
        if(isfirst) { isfirst = 0; } \  
        else { printf(","); } \  
    } \  
    if(benum==ennum) { printf("%d", benum); } \  
    else { printf("%d-%d", benum, ennum); } \  
}
```

最初は出力マクロが実行されるが、benum の初期値を -7 にしてあるので実際には出力しない

出力ループ: 連続していないところで出力

```
for(i=0;i<listuse;i++) {  
    curnum = numlist[i];  
    if(curnum==ennum+1) {  
        ennum = curnum;  
    }  
    else {  
        P;  
        benum = curnum;  
        ennum = curnum;  
    }  
}  
P;
```

numlist

2	3	4	9	10					
---	---	---	---	----	--	--	--	--	--

▲
print

▲
print

▲
print

"2-4" "9-10"

範囲指定 圧縮 (*cont.*)

最後にも出力する点に注意
実践的には配列に格納する箇所はエラー処理が必要

配列に蓄えるため、その大きさが能力が決まる

- 100個の配列では内容 100個の範囲しか扱えない
- 一方、巨大配列を設けては効率が悪い
- 数字を直接蓄える点を変更すると効果が期待大
- ビットマップでは劇的な効果は期待薄

範囲指定 別の実装

より直観的な実装
同様に値を格納する配列を用意

```
int nummap[MAXNUM];
```

配列の添字にして蓄える

```
curnum = atoi(tmp);  
nummap[curnum]++;
```

入力順の影響がない

0	1	2	3	4	5	6	7	8	9	10	11	
0	0	1	1	1	0	0	0	0	1	1	0	●●●

範囲指定 別の実装 (*cont.*)

出力

```
for (i=0; i<MAXNUM; i++) {  
    if (!nummap[i])  
        continue;  
    curnum = i;  
    . . .
```

やはり配列の大きさが能力が決まる

- 100個の配列では 0-99の範囲しか扱えない