I117（3）文字、文字列処理（その1）

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

char

char は文字向けの型でサイズは 1バイト

型	最小	最大
signed char	-128	127
unsigned char	0	255

- 小さな数字の型として使うこともある
  - 画像処理等
- 文字定数は ’ (シングル・クォート; single-quote 0x27) で囲む
- 多くの処理系では ASCII 文字を想定している

ASCII 文字コード

マニュアルで確認できる

```
man ascii
```

代表的な値

	SP	0	9	A	Z	a	z
Oct(8進)	040	060	071	101	132	141	172
Hex(16進)	20	30	39	41	5A	61	7A
Dec(10進)	20	48	57	65	90	97	122

文字列

- C 言語では文字列は文字の配列
- 0 が終了の印で '\0' と表記
- 文字列定数は " (ダブル・クォート; double-quote 0x22) で囲む
- 配列へのポインタとして登場することが多い

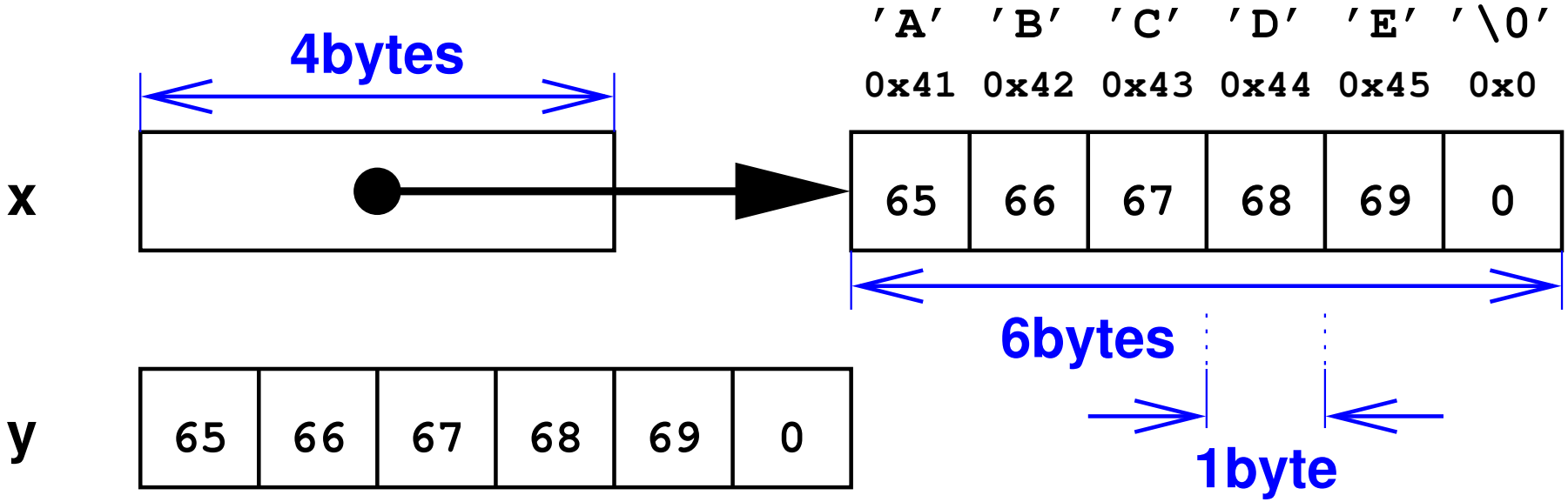
```
char *x="ABCDEF";

strcpy(dst, "foo.tar.gz");
```

「配列へのポインタ」と「配列」の違い

```
char *x="ABCDE";
char y[]={'A','B','C','D','E','\0'};
```

格納の形、サイズが異なる



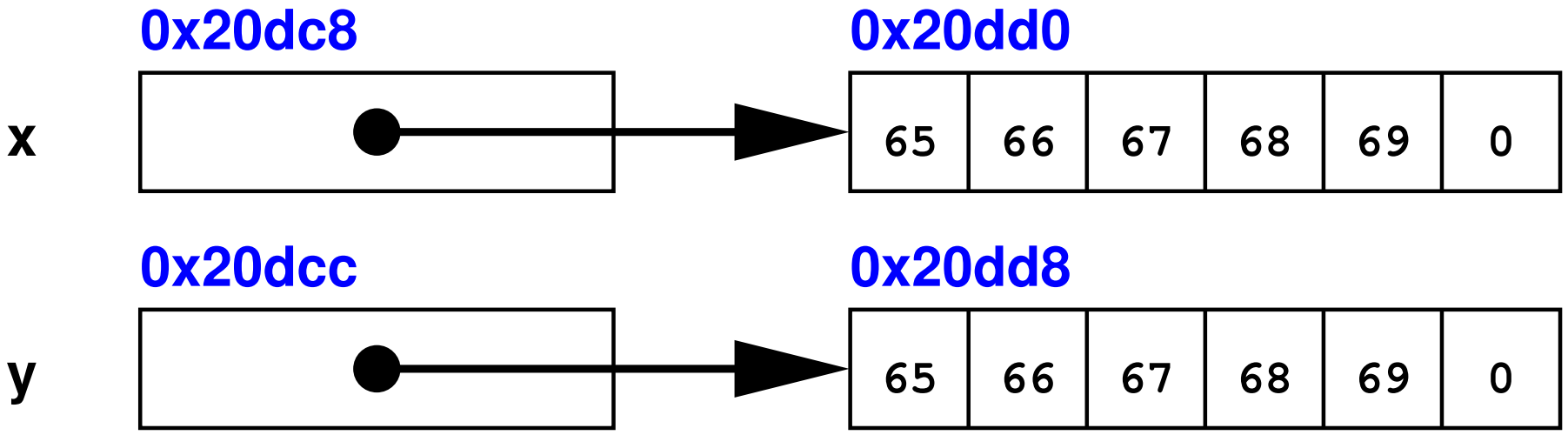
格納方法

配列へのポインタの最適化の例

```
#include <stdio.h>
char *x="ABCDE";
char *y="ABCDE";
int main() {
    printf("&x %p x %p size %d\n",
        &x, x, sizeof(x));
    printf("&y %p y %p size %d\n",
        &y, y, sizeof(y));
}
```

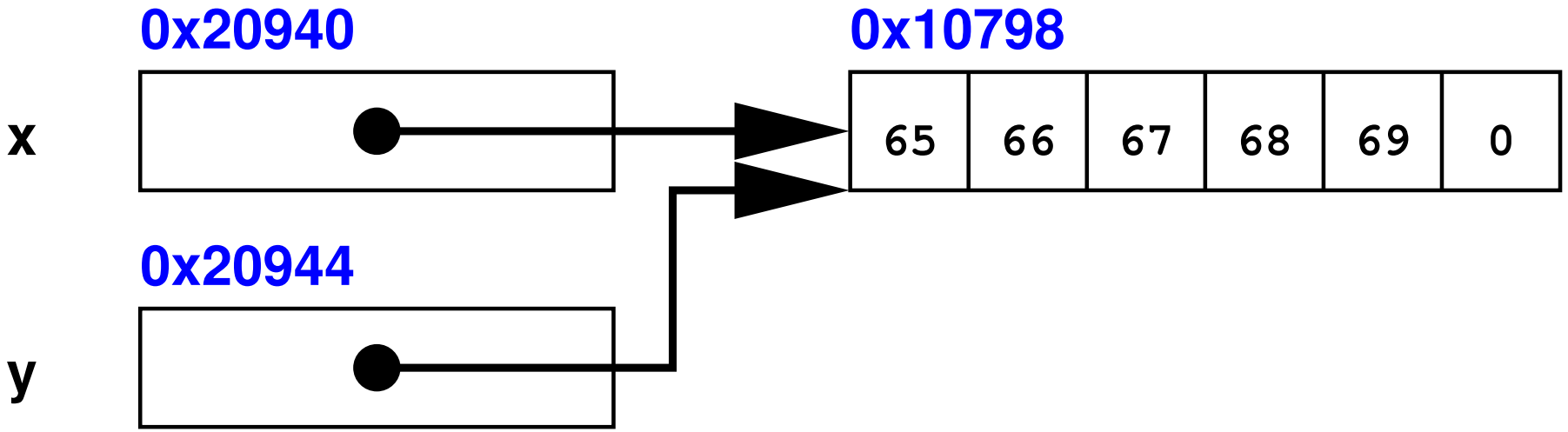
格納方法 *(cont.)*

```
% cc ls3.c
% ./a.out
&x 20dc8 x 20dd0 size 4
&y 20dcc y 20dd8 size 4
```



格納方法 — 最適化して共有

```
% gcc ls3.c
% ./a.out
&x 20940 x 10798 size 4
&y 20944 y 10798 size 4
```



格納方法 — 最適化して共有 *(cont.)*

- 内容が同じだと共有できる
- 配列へのポインタでも最適化できる余地がある

共有すると動かなくなるプログラムもある

- 文字列を書き換える
  - 古いプログラムに多い
  - 単に初期値が同じなだけの場合も

※ 万全な策ではない、注意が必要

典型的な操作

- 新規作成 — malloc, strdup
- 転写、追加 — strcpy, strcat
- 削除
- 検索 — strchr, strstr
- 開放 — free

- 変換（次回）

新規作成

定数—コンパイル時に作成

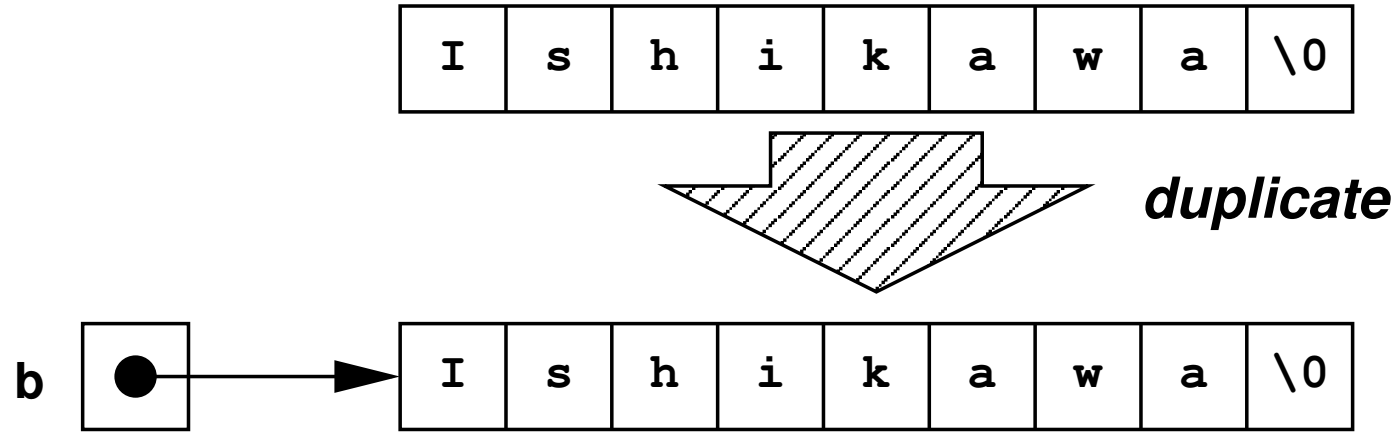
```
char *a = "Japan";
```

実行時に作成

```
char *b, *c;
...
b = strdup("Ishikawa");
c = (char*)malloc(9);
c[0] = 'A'; c[1] = '\0';
```

strdup

strdup は新しい領域を設けて文字列を複写する



以下の手順にほぼ等価

```
x = "Ishikawa";
b = malloc(strlen(x)+1);
strcpy(b, x);
```

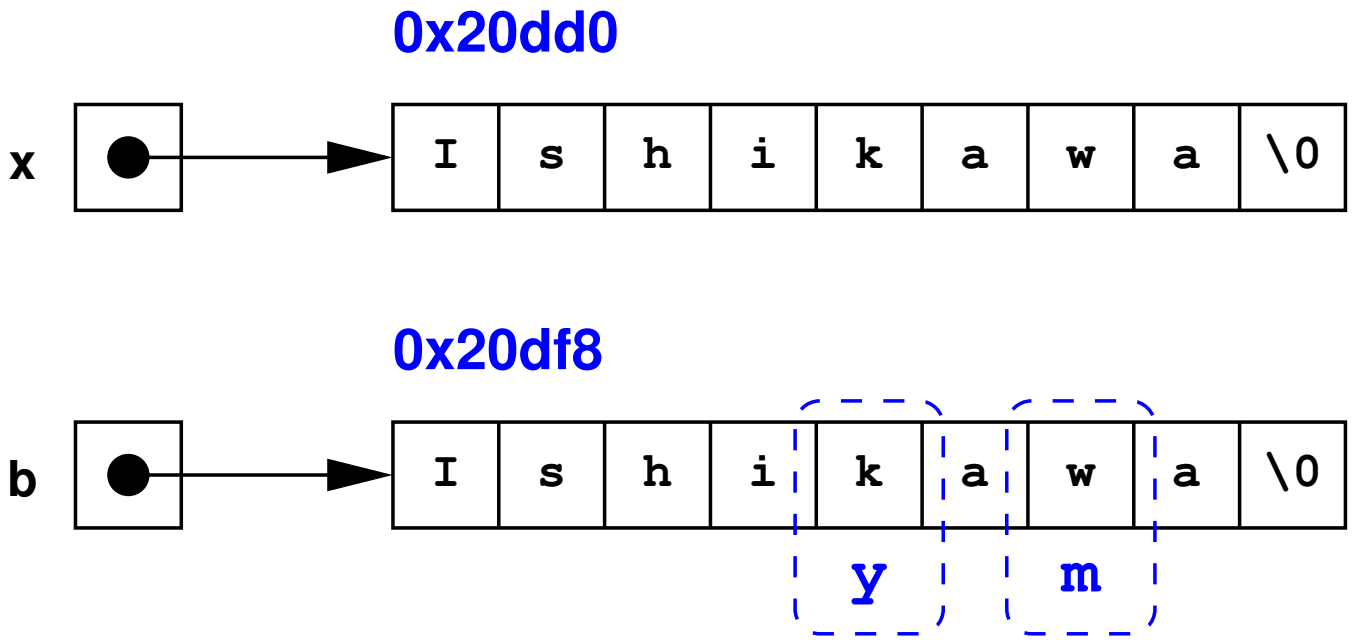
strdup *(cont.)*

```
char *x = "Ishikawa";
char *b = strdup(x);
b[4] = 'y';
b[6] = 'm';
printf("x %p, %s\n", x, x);
printf("b %p, %s\n", b, b);
```

結果は b の内容を書き換えても x は変化なし

```
x 20dd0, Ishikawa
b 20df8, Ishiyama
```

strdup *(cont.)*



x と同じ長さしか複製されない点に注意

- 短縮は可、延長は不可

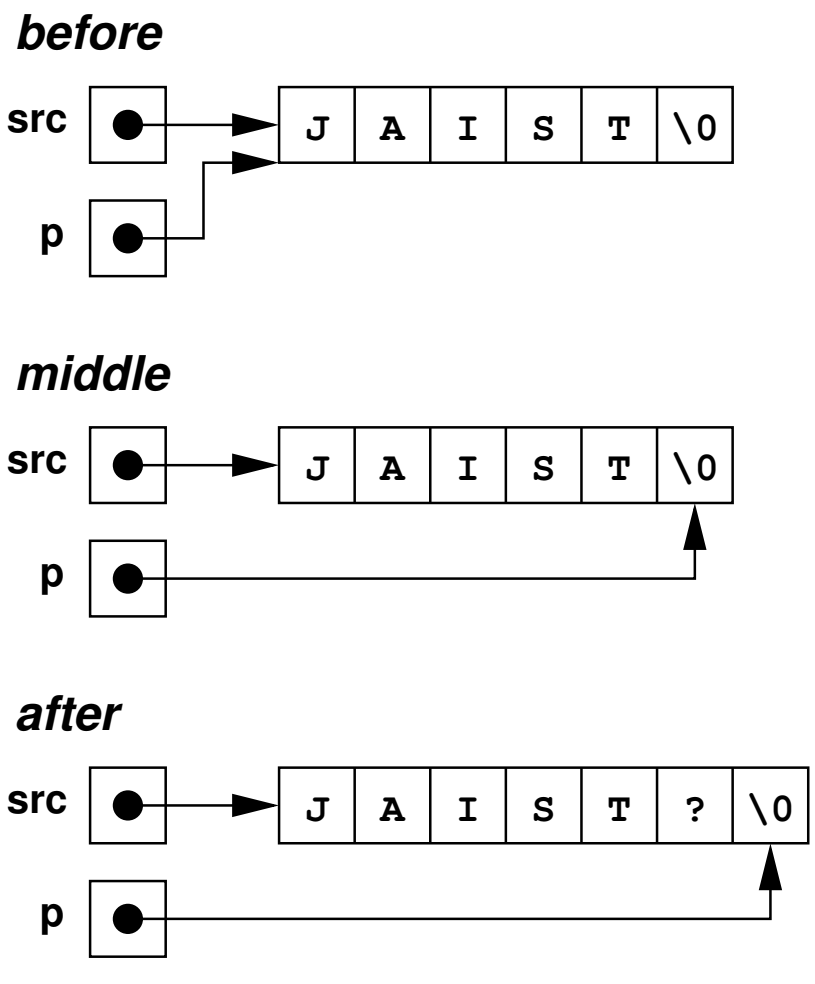
追加

末尾を探す

```
char *p=src;
while(*p) {
    p++;
}
```

p は終了端を指す

```
*p++ = '?';
*p = '\0';
```



追加 *(cont.)*

標準ライブラリ

- strcat — 追加

```
strcat(src,"?");          => "JAIST?"
```

- strcpy — 先頭から追加、上書き

```
strcpy(src,"AIST?");      => "AIST?"
```

- strncpy — 文字数指定、'\0' を書かない

```
strncpy(src,"AIST?", 2); => "AIIST"
```

追加 *(cont.)*

こう書くとわかりやすい

```
   ＪＡＩＳＴφ
＋ＡＩ
―――――――――
   ＡＩＩＳＴφ
```

- 類似関数 memcpy, memmove

これらは処理系向けに最適化されて高速

- アセンブラの実装が多い
- 安直に勝負しないこと
- 複雑な処理は自作した方が高速な場合もある

追加 *(cont.)*

src の大きさに注意

- 余裕が無いと他のデータを破壊する

```
char *src=strdup("JAIST");
```

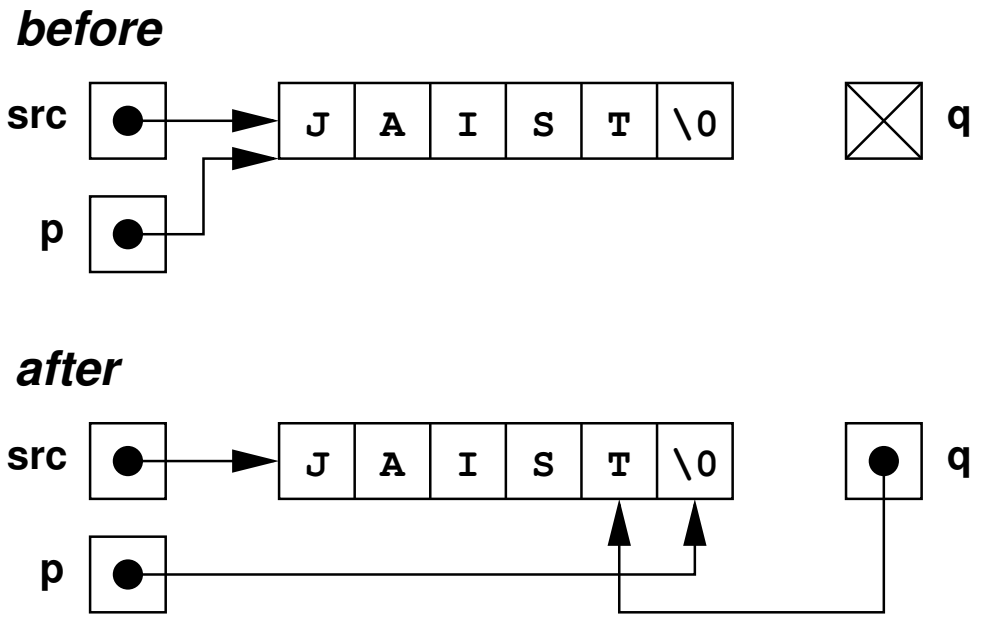
strdup() は元の文字列長以上の領域は未保証、危険

```
char buf[10];
char *src=buf;
```

余裕のある文字列なら安全
この場合、9文字まで格納可能

追加応用 — 最後の文字を書き換える

```
char *p=src;
char *q=NULL;
while(*p) {
     q = p;
     p++;
}
```



q は最後の要素を指す

- q が NULL なら文字列は空

```
if(q) *q = '#';
```

追加応用 — 最後の文字を書き換える *(cont.)*

結果

```
JAIS#
```

標準ライブラリを使うなら

```
int w;
...
w = strlen(src);
if(w>0) {
    src[w-1] = '#';
}
```

削除（初期化）

任意の場所から '\0' の代入で削除可能
先頭なら初期化となる

```
char *a=(char*)malloc(10);
a[0] = '\0';
...
a[4] = '\0';
```

検索 — 1文字

```
char *p=src;
char *x=NULL;
while(*p) {
    if(*p=='S') {
        x = p;
        break;
    }
    p++;
}
```

x が NULL でなければ見付かった場所

検索 — 文字列

```
char *p=src;
char *x=NULL;
while(*p) {
    if(*p=='A' && *(p+1)=='I') {
        x = p;
        break;
    }
    p++;
}
```

x が NULL でなければ見付かった場所

検索 — 文字列 *(cont.)*

標準ライブラリ

- strchr 文字列中の文字検索

```
char *x = strchr("JAIST", 'S');
```

- strstr 文字列中の文字列検索

```
char *x = strstr("JAIST", "AI");
```

検索アルゴリズムは広く研究されている
他の講義で紹介されるだろう

開放

strdup(), malloc() で作成するとヒープ領域

- free() で開放する

コンパイル時に作成した場合は開放できない