

I117 (5) 文字、文字列処理 (その3)

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

パス名操作 — 拡張子走査

- ピリオド（0x2e）を探す
- 複数存在する可能性も考慮

```
p = src;  q = NULL;
while(*p) {
    if(*p=='.') {
        q = p;
    }
    p++;
}
```

- q が NULL 以外なら最後のピリオドの場所

パス名操作 — 拡張子走査 (*cont.*)

実行結果例

```
% ./a.out /a/b/c.def  
.def  
% ./a.out /a/b/c.def.xyz  
.xyz
```

パス名操作 — 拡張子走査 (*cont.*)

標準ライブラリ利用

```
p = src;  q = NULL;
while(*p) {
    p = strchr(p, '.');
    if(!p) {
        break;
    }
    q = p;
    p++;
}
```

ディレクトリ名、ベース名

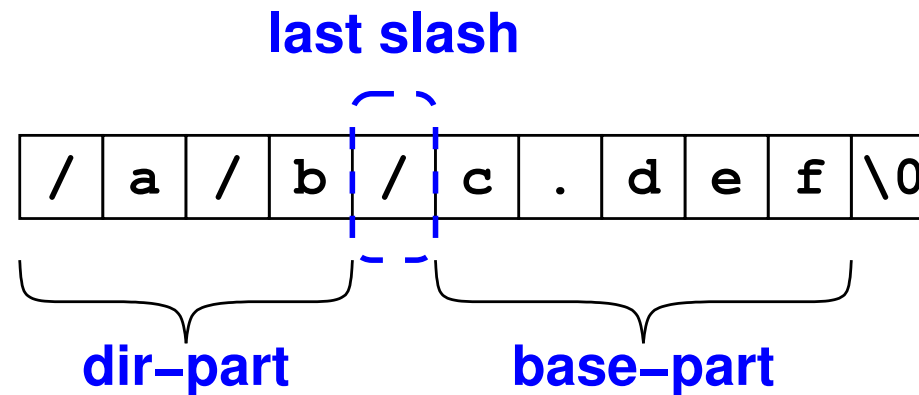
dirname, basename 相当のプログラムを作ってみる

```
% dirname /a/b/c.def  
/a/b  
% basename /a/b/c.def  
c.def
```

パスのディレクトリ部と残りを表示するプログラム

- 突き詰めると、最後のスラッシュ（0x2f）を探すことが肝心

ディレクトリ名、ベース名 (*cont.*)



(念のため) スラッシュが無い場合の挙動

```
% dirname a
.
% basename a
a
```

最後のスラッシュ探索

```
char* slashpos(char *src) {  
    char *p=src, *q=NULL;  
    while(*p) {  
        if(*p=='/') { q = p; }  
        p++;  
    }  
    if(q) { return q; }  
    return NULL;  
}
```

```
int printdir(char *src) {  
    char *p, *pos = slashpos(src);  
    if(pos) {  
        p = src;  
        while(p<pos) {  
            printf("%c", *p++);  
        }  
        printf("\n");  
    }  
    else {  
        printf(".\n");  
    }  
}
```



```
int printbase(char *src) {  
    char *pos = slashpos(src);  
    if(pos) {  
        pos++;  
        printf("%s\n", pos);  
    }  
    else {  
        printf("%s\n", src);  
    }  
}
```

ディレクトリ名、ベース名 (*cont.*)

まとめて出力する

```
% ./a.out /a/b/c.def  
/a/b  
c.def  
% ./a.out a  
.  
a
```

同等のプログラムができた

パス縮退

- 多くの OS では親ディレクトリへアクセス可能
- セキュリティの面から制限したい場面がある
 - ◇ パスワード等秘密のファイル
 - ◇ システムの挙動がわかるファイルを見せない

```
% cat ../../etc/passwd  
% cat ../../var/log/message
```

- 最終的なパスを知るプログラムを作る

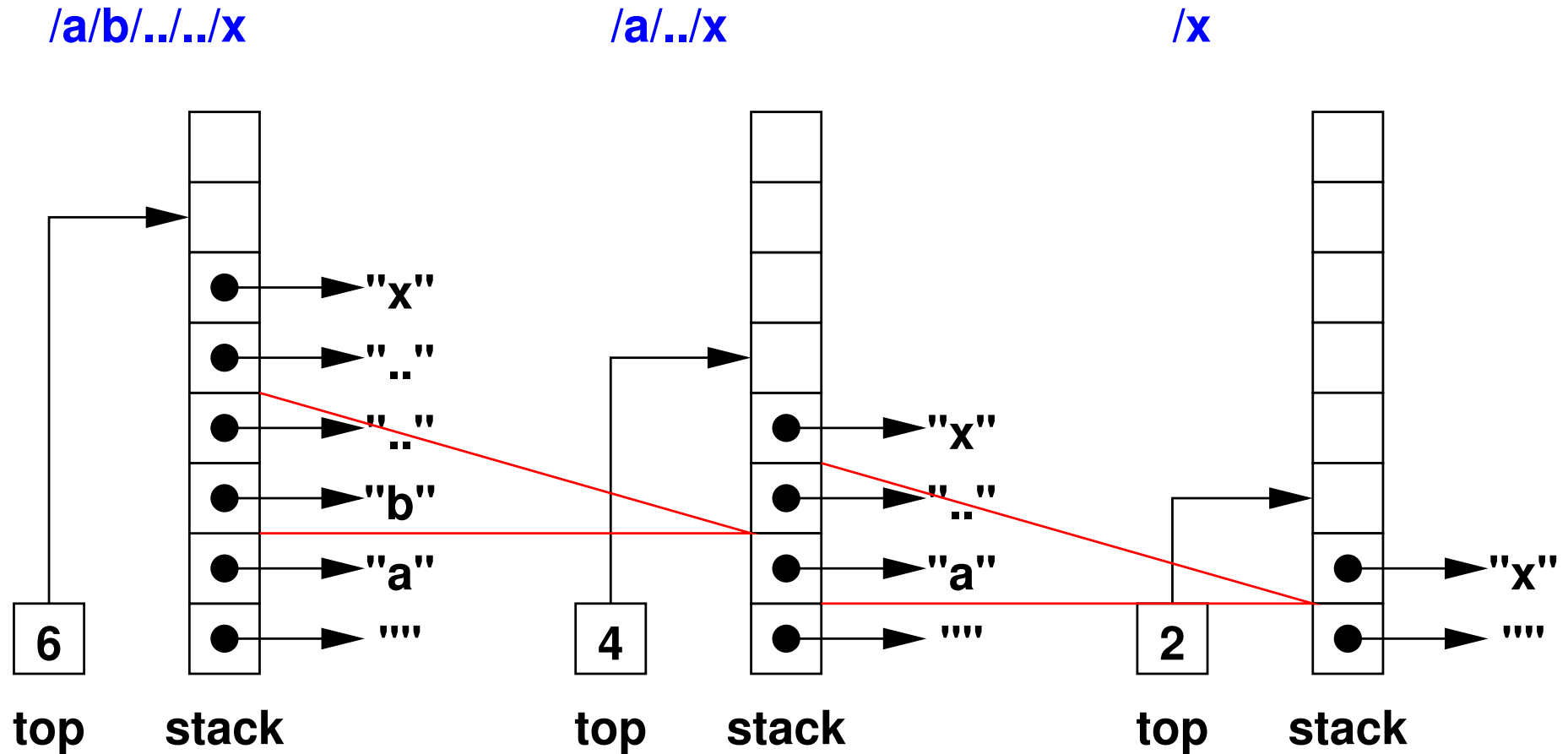
パス縮退 (*cont.*)

手順

- スラッシュで区切られた単語列を読む
- .. があればその場所と直前（計2つ）除去

/a/.. /x	=> /x
/a/b/.. /x	=> /a/x
/a/b/.. /.. /x	=> /x

パス縮退 (cont.)



パス縮退 (*cont.*)

スタックを設ける

```
int top;  
char *stack[STACKSIZE];
```

```
int stackclear() {  
    int i;  
    for(i=0; i<STACKSIZE; i++) {  
        stack[i] = NULL;  
    }  
    top = 0;  
}
```

パス縮退 (*cont.*)

```
int stackpush(char *s) {  
    if(top<STACKSIZE-1) {  
        stack[top] = strdup(s); top++;  
        return 0;  
    }  
    else {  
        printf("stack overflow\n");  
        return -1;  
    }  
}
```

```
void pathstore(char *src) {  
    char tmp[BUFSIZ]; char *p=src, *q=tmp;  
    while(*p) {  
        if(*p=='/') {  
            *q = '\\0'; stackpush(tmp);  
            q = tmp;  
        }  
        else {  
            *q++ = *p;  
        }  
        p++;  
    }  
    *q = '\\0'; stackpush(tmp);  
}
```



```
int stackreduce() {  
    int i, j;  
    for(i=1;i<top;i++) {  
        if(strcmp(stack[i], "..")==0) {  
            free(stack[i-1]); free(stack[i]);  
            for(j=i+1;j<top;j++) {  
                stack[j-2] = stack[j];  
            }  
            stack[top-2] = stack[top-1] = NULL;  
            top -= 2;  
            return 1;  
        }  
    }  
    return 0; }
```

パス縮退 (*cont.*)

全体を整理

- stackreduce は処理したら 1 を返す
- stackreduce が 0 を返すまで繰り返す
 - ◇ つまり、”..”があるだけ繰り返す

```
do {  
    n = stackreduce();  
} while(n>0);
```

パス縮退 (*cont.*)

最終的な出力

```
for(i=0;i<top;i++) {  
    if(i<top-1) {  
        printf("%s/", stack[i]);  
    }  
    else {  
        printf("%s", stack[i]);  
    }  
}  
printf("\n");
```

補足

- UNIX ではディレクトリのアクセス制限のために、`chroot()` という機構がある
- `chroot()` が扱えない場面では、今回のようなパス縮退も有用