ECE 220: Computer Systems & Programming

Lecture 10: Strings and Multi-dimensional Arrays

Thomas Moon

February 20, 2024

Strings

- Allocate space for a string just like any other **array**

```
char buf[200];
```

- Space for string must contain room for terminating zero.
- Null terminating strings- '\0' special sequence that corresponds to the null character.
- **Special syntax for initializing a string:**

```
char buf[200] = "abc";
buf[0] = 'a';
buf[1] = 'b';
buf[2] = 'c';
buf[3] = '\0';
```

How about this?

```
char buf[200];
buf = "abc";
```

err

Exercise: Copy a string from source to destination

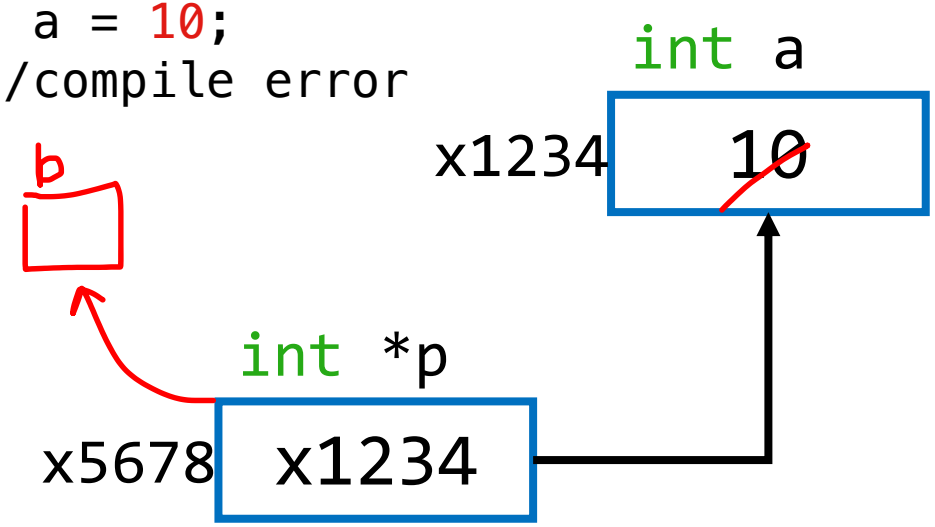
```
int main(){
    char buf[200];
    //buf = "ABC"; // compile error
    string_copy(buf,"ABC");
}


void string_copy(char des[] ,_char src[]){
    int i=0;
    while( src[i] ! = ‘\0’   _){
        des[i] = src[i];
        i++;
    }
    des[i] = '\0’;
    src[2] = ‘b’;//
}
```

pointer to const

```
int a = 10;
int b = 30;
const int *p = &a;
*p = 20;          ← Compile error
p = &b;           ← OK
a = 20;           ← OK
```

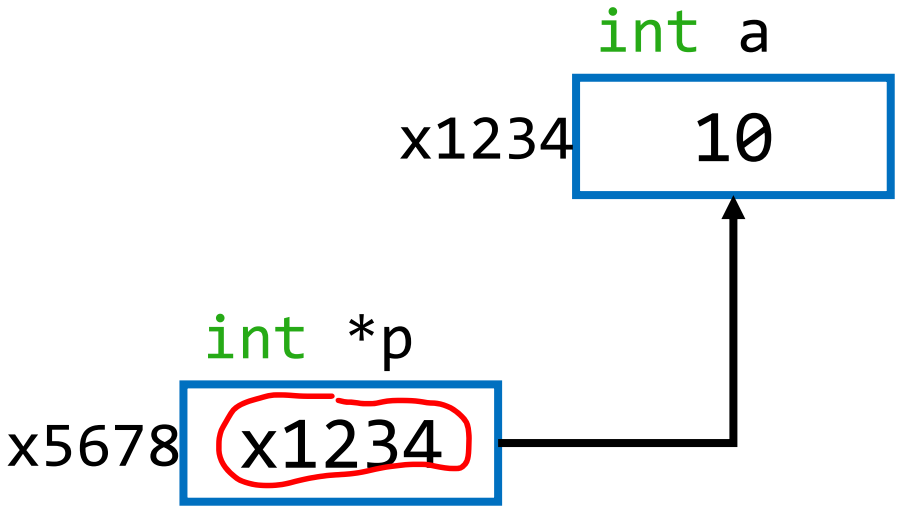
```
const int a = 10;
a = 20; //compile error
```



"Pointer to const" DOES NOT allow the pointed object to be changed by the pointer.

const pointer

```
int a = 10;
int b = 30;
int * const ptr = &a;
*ptr = 20;        ← OK
ptr = &b;         ← Compile error
a = 20;           ← OK
```



"Const pointer" to DOES NOT allow to change the pointer.

Exercise: Copy a string from source to destination

```
int main(){
    char buf[200];
    //buf = "ABC"; // compile error
    string_copy(buf,"ABC");
}
```

```
void string_copy(char des[], const char src[]){
    int i=0;
    while( src[i] != '\0' ){
        des[i] = src[i];
        i++;
    }
    des[i] = '\0';
    src[0] = 'z';
}
```

← const will prevent any changes in src

← Compile error

I/O with Strings

printf and scanf use “%s” format character for string.

`printf`: print characters up to terminating zero

```
char buf[10]="abc";
printf("%s\n", buf);
```

`scanf`: read characters until whitespace, store result in string, and terminating with zero.

```
char input[10];
scanf("%s", input);
```

How about this?

```
char input[10];
scanf("%s", &input[0]);
```

Q1.

```
char temp[100] = "abcdef";
temp[3] = '\0';
printf("%s", temp);
```

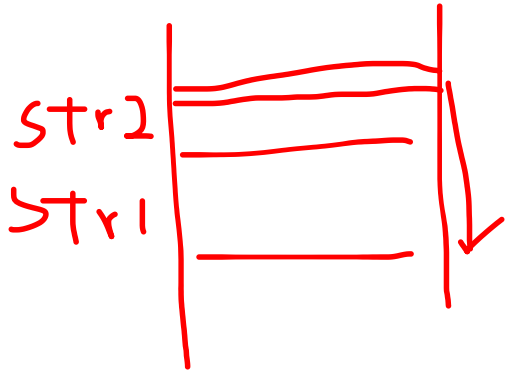
abc

Q2.

```
char temp[100] = "abcdef";
printf("%s", &temp[2]);
```

cdef

Read Strings



- fgets vs scanf

```
char buf[SIZE_BUF];
fgets(buf, SIZE_BUF, stdin);
scanf("%s",buf);
```

stdin→ keyboard
stdout→ monitor

fgets: Read characters from stdin(keyboard)
and store them into buf
until SIZE_BUF-1 characters
or a newline or the end-of-file

scanf: Read characters from stdin(keyboard)
and store them into buf
until whitespace characters

Read Strings

- `sscanf:` Read formatted data from string
  and retrun the number of items successfully read

```
char buf[SIZE_BUF]="12˅34˅abc";
int rc;
int num1, num2;
char str[20];
rc = sscanf(buf,"%d%d%s", &num1, &num2, str);
```

```
rc=3, num1=12, num2=34, str=abc
```

Multi-dimensional Arrays

```
int a[4];
```

	col0	col1	col2	col3
row0	a[0]	a[1]	a[2]	a[3]

```
int a[2][3];
```

	col0	col1	col2
row0	a[0][0]	a[0][1]	a[0][2]
row1	a[1][0]	a[1][1]	a[1][2]

In memory,

	offset
	offset
a[0]	0
a[1]	1
a[2]	2
a[3]	3

offset=
(row index) * (size of column) + column index

	offset
a[0][0]	0
a[0][1]	1
a[0][2]	2
a[1][0]	3
a[1][1]	4
a[1][2]	5

1×3 +1

Multi-dimensional array is stored in row-major order

Passing Array as Arguments

- C passes arrays by reference
  - The address of the array (*address of the first element*) is written to the function's activation record.

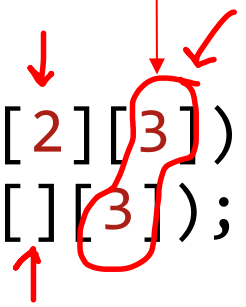
Must specify the **second** dimension

1D array

```
void func(int a[6]);
void func(int a[]);
void func(int *a);
```

2D array

```
void func(int a[2][3]);
void func(int a[][3]);
```



```c
#define ROW 2
#define COL 3

void print_2D(int a[][COL]){
    int i, j;
    for(i=0; i<ROW; i++){
        for(j=0; j<COL; j++){
            printf("%d ", a[i][j]);
        }
        printf("\n");
    }
    printf("\n");
}

void print_2D_ptr(int *a){
    int i, j;
    for(i=0; i<ROW; i++){
        for(j=0; j<COL; j++){
            printf("%d ", a[i*COL+ j]);
        }
        printf("\n");
    }
    printf("\n");
}
```

compiler needs to know the size of column to calculate the memory offset

offset=
(row index) * (size of column) + column index

```c
int main()
{
    int two[2][3] = {{1,2,3},{4,5,6}};
    print_2D(two);
    print_2D_ptr(&two[0][0]);
}
```

pass the array as a pointer

lose the 2D-array shape information

Initialize Multi-dimensional Array

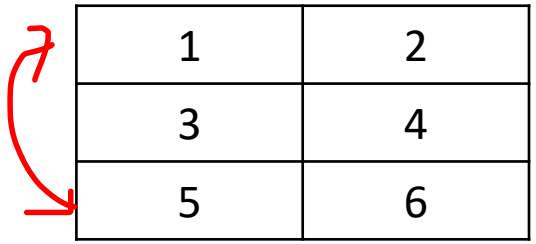
```
int a[2] = {1,2};
int a[] = {1,2};

int a[2][3] = {{1,2,3},{4,5,6}};
int a[2][3] = {1,2,3,4,5,6};
int a[][3] = {{1,2,3},{4,5,6}};
int a[2][] = {{1,2,3},{4,5,6}};
int a[][] = {{1,2,3},{4,5,6}};
```

Compile error!

Compile error!

Exercise: Exchange two rows in matrix




```
// Swap x-th row and y-th row
void matrix_change(int matrix[N][M], int x, int y)
{
    int temp, i;
    for(i=0_; i < M ____; i++_){
        temp = matrix[x][i]_____________________________;
        matrix[x][i] = matrix[y][i]_____________________;
        matrix[y][i] = temp_____________________________;
    }

}
```

x=0, y=2

5	6
3	4
1	2

Exercise: Transpose a matrix

1	2
3	4
5	6

```
// Transpose src matrix and store in des matrix
void matrix_tr(int des[M][N], const int src[N][M])
{
    int i,j;//index for destination

    for(i=0;i < M;i++){
        for(j=0;j<N;j++){
            des[i][j] = src[j][i];
        }
    }

or
    for(i=0;i < N;i++){
        for(j=0;j<M;j++){
            des[j][i] = src[i][j];
        }
    }
```

1	3	5
2	4	6