Recitation 03

Arrays – Strings

George Mappouras

9/22/2017

Arrays & Dimensions

- Allocate memory of certain size
- How many dimensions? From 1 to ~30! As many as it makes sense.
- Why may I need more than 3 dimensions?

  Example: result = f(t,p,Q,T,A)
- Arrays of any type or even structs!

Initializing an Array

```
int array[5] = {1 2 3 4 5};

int array[5] = {0};



for (int i=0; i<5; i++)
  array[i]=i+1;
```

Array & Pointers

int array[5] = {0};

int *p = array

4 ways to access the second element of array?

Array & Pointers

int array[5] = {0};

int *p = array

4 ways to access the second element of array?

Answer:

1. array[1]
2. *(array+1)
3.
4.

Array & Pointers

int array[5] = {0};

int *p = array

4 ways to access the second element of array?

Answer:

1. array[1]
2. *(array+1)
3. *(p+1)
4. p[1]

Array & Pointers

```
int array[5] = {0};
int *p = array
```

Is that valid?

```
array = p;
```

Array & Pointers

int array[5] = {0};

int *p = array

Is that valid?

array = p; NO!

Array & Pointers

int array[5] = {0};

int *p = array

Is that valid?

array = p; NO!

Why?

array is not an lvalue (there is no box called array)

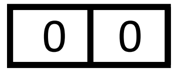
Arrays & Memory

Instruction		What I actually store
• int array[2] = {0};	=>	Continues space for 2 integers The number of dimensions = 1
• int array[2][2] = {{0,0},{1,1}};	=>	Continues space for 4 integers The number of dimensions = 2

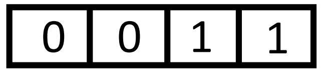
Arrays & Memory

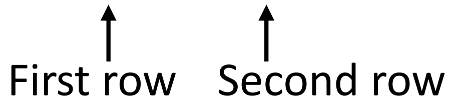
Instruction   Memory

- int array[2] = {0};   =>



- int array[2][2] = {{0,0},{1,1}};   =>





Example Problem

```
#include <stdlib.h>
#include <stdio.h>
int main(){
  int array[2][2] = {{1,2},{3,4}};
  int *ptr = &array[0][0];
  printf("%d\n",array[0][3]);
  for (int i =0; i<4; i++)
    printf("%d\n",*(ptr+i));
  return 0;
}
```

Example Problem

```
#include <stdlib.h>
#include <stdio.h>
int main(){
  int array[2][2] = {{1,2},{3,4}};
  int *ptr = &array[0][0];
  printf("%d\n",array[0][3]);
  for (int i =0; i<4; i++)
    printf("%d\n",*(ptr+i));
  return 0;
}
```

Answer

```
4
1
2
3
4
```

Example Problem 2

```
int main(){
  int array[2][2] = {{1,2},{3,4}};
  int *ptr = array[1];
  printf("%d\n",*(array[1]));
  for (int i =0; i<2; i++)
    printf("%d\n",*(ptr+i));
  return 0;
}
```

Example Problem 2

```
int main(){
  int array[2][2] = {{1,2},{3,4}};
  int *ptr = array[1];
  printf("%d\n",*(array[1]));
  for (int i =0; i<2; i++)
    printf("%d\n",*(ptr+i));
  return 0;
}
```

Answer

```
3
3
4
```

Array & Pointers 2 – Pass array to a function

```
int* function1 (int * array, size_t size){
        array[size-1] = 10;
        return array
}

void main(){
        int my_array[10];
        int *b = function1(my_array,10);
}
```

Example

```
#include <stdlib.h>
#include <stdio.h>
void func_pointer (int * array){
  array[0]=10;
  array[2]=12;
}

void func_array (int array[][2]){
 array[0][1]=11;
 array[1][1]=13;
}
```

```
int main(){
  int array[2][2] = {{1,2},{3,4}};
  int *ptr = array[0];
  func_pointer(ptr);
  func_array(array);

  for (int i =0; i<4; i++)
    printf("%d\n",*(array[0]+i));
  return 0;
}
```

Example

```
#include <stdlib.h>
#include <stdio.h>
void func_pointer (int * array){
  array[0]=10;
  array[2]=12;
}

void func_array (int array[][2]){
 array[0][1]=11;
 array[1][1]=13;
}
```

```
int main(){
  int array[2][2] = {{1,2},{3,4}};
  int *ptr = array[0];
  func_pointer(ptr);
  func_array(array);

  for (int i =0; i<4; i++)
    printf("%d\n",*(array[0]+i));
  return 0;
}
```

Answer

```
10
11
12
13
```

Array & Pointers 2 – Dangling Pointer

```
int* function2 (size_t size){
        int array[size] = {0};
        return array
}

void main(){
        int *b = function2(10);
}
```

Where is b pointing?

Array & Pointers 2 – Dangling Pointer

```
int* function2 (size_t size){
        int array[size] = {0};
        return array
}

void main(){
        int *b = function2(10);
}
```

Where is b pointing?
dangling pointer

Question 9.13 – AOP page 158

- int array[3];
- int a;
- int * p = &array[1];
- int * q = &a;
- int ** r = &p;

Group the following names:
a, p,*p, p[1], array[0], array[1], array[2], q, *q, **r, *r

Question 9.13 – AOP page 158

- int array[3];
- int a;
- int * p = &array[1];
- int * q = &a;
- int ** r = &p;

(a,*q)
(*p, array[1],**r)
(p[1],array[2])
(*r, p)
array[0], p, q do not group

Strings

```
const char * str = “hello world\n”;

char str[] = “hello world\n”;

char str[] = “ ‘h’, ‘e’, ‘l’, ‘l’, ‘o’, ‘ ’, ‘w’, ‘o’, ‘r’, ‘l’, ‘d’, ‘\n’, ‘\0’”;
```

Strings

```
const char * str = “hello world\n”;

char str[] = “hello world\n”;

char str[] = “ ‘h’, ‘e’, ‘l’, ‘l’, ‘o’, ‘ ’, ‘w’, ‘o’, ‘r’, ‘l’, ‘d’, ‘\n’, ‘\0’”;
```

Null Terminator

Comparing and Copying Strings

char str1[8] = “ece551\n”;

char str2[8] = “ece551\n”;

char * str3 = str1;

What do the follow expressions return (False or True):

str1 == str2

str1 == str3

Comparing and Copying Strings

```
char str1[8] = “ece551\n”;
char str2[8] = “ece551\n”;
char * str3 = str1;
```

What do the follow expressions return (False or True):

str1 == str2	False
str1 == str3	True

Comparing and Copying Strings

```
char str1[8] = “ece551\n”;
char str2[8] = “ece590\n”;
char * str3 = str2;
str2[0] = ‘E’ ;
```

Where do they point?

*str2

*str3

*str1

ece551\n\0

ece590\n\0

Ece590\n\0

Comparing and Copying Strings

```
char str1[8] = “ece551\n”;
char str2[8] = “ece590\n”;
char * str3 = str2;
str2[0] = ‘E’ ;
```

Where do they point?

$$\begin{tikzcd}
{*\text{str2}} \arrow[ddr] & \text{ece551\textbackslash n\textbackslash 0} \\
{*\text{str3}} \arrow[dr] & \text{ece590\textbackslash n\textbackslash 0} \\
{*\text{str1}} \arrow[uur] & \text{Ece590\textbackslash n\textbackslash 0}
\end{tikzcd}$$

Comparing and Copying Strings

```
char str1[8] = “ece551\n”;
char str2[8] = “ece590\n”;
char * str3 = str2;
str2[0] = ‘E’ ;
```

What if I try to execute “str2 = str1;”?

Where do they point?

$$\begin{tikzcd}
\text{*str2} \arrow[rr, dd] & & \text{ece551\textbackslash n\textbackslash 0} \\
\text{*str3} \arrow[rrd] & & \text{ece590\textbackslash n\textbackslash 0} \\
\text{*str1} \arrow[rruu] & & \text{Ece590\textbackslash n\textbackslash 0}
\end{tikzcd}$$

Example

```
#include <stdlib.h>
#include <stdio.h>
int main(){
  const char * str = "Hello World\n";
  char str2[] = "Hello World\n";
  char  * const str3 = str2;
  char str4[] = "Hi there!\n";
  printf("%s",str);
  printf("%s",str2);
  printf("%s", str4);
  str4[0] = 'P';
  str = str3;
  printf("%s",str);
  return 0;
}
```

Example

```
#include <stdlib.h>
#include <stdio.h>
int main(){
  const char * str = "Hello World\n";
  char str2[] = "Hello World\n";
  char  * const str3 = str2;
  char str4[] = "Hi there!\n";
  printf("%s",str);
  printf("%s",str2);
  printf("%s", str4);
  str4[0] = 'P';
  str = str3;
  printf("%s",str);
  return 0;
}
```

What is the difference?

Example

```
#include <stdlib.h>
#include <stdio.h>
int main(){
  const char * str = "Hello World\n";
  char str2[] = "Hello World\n";
  char  * const str3 = str2;
  char str4[] = "Hi there!\n";
  printf("%s",str);
  printf("%s",str2);
  printf("%s", str4);
  str4[0] = 'P';
  str = str3;
  printf("%s",str);
  return 0;
}
```

Anything wrong here?

Some useful string functions

- size_t strlen(const char * str): get length of a string
- int strcmp(const char* str1, const char* str2): compare two strings for equality/ordering
- char* strncpy(char* dest, const char * source, size_t n): copy string from one location to another
- int atoi(const char * str): convert string to integer
- char * strcat(char*dest, char*source): append two strings
- char * strchr(char * str, int character): locate the first occurrence of a specific character in a string

Writing our own atoi

ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	*[NULL]*	32	20	*[SPACE]*	64	40	**@**	96	60	`
1	1	*[START OF HEADING]*	33	21	**!**	65	41	**A**	97	61	**a**
2	2	*[START OF TEXT]*	34	22	**"**	66	42	**B**	98	62	**b**
3	3	*[END OF TEXT]*	35	23	**#**	67	43	**C**	99	63	**c**
4	4	*[END OF TRANSMISSION]*	36	24	**$**	68	44	**D**	100	64	**d**
5	5	*[ENQUIRY]*	37	25	**%**	69	45	**E**	101	65	**e**
6	6	*[ACKNOWLEDGE]*	38	26	**&**	70	46	**F**	102	66	**f**
7	7	*[BELL]*	39	27	**'**	71	47	**G**	103	67	**g**
8	8	*[BACKSPACE]*	40	28	**(**	72	48	**H**	104	68	**h**
9	9	*[HORIZONTAL TAB]*	41	29	**)**	73	49	**I**	105	69	**i**
10	A	*[LINE FEED]*	42	2A	*	74	4A	**J**	106	6A	**j**
11	B	*[VERTICAL TAB]*	43	2B	**+**	75	4B	**K**	107	6B	**k**
12	C	*[FORM FEED]*	44	2C	**,**	76	4C	**L**	108	6C	**l**
13	D	*[CARRIAGE RETURN]*	45	2D	**-**	77	4D	**M**	109	6D	**m**
14	E	*[SHIFT OUT]*	46	2E	**.**	78	4E	**N**	110	6E	**n**
15	F	*[SHIFT IN]*	47	2F	**/**	79	4F	**O**	111	6F	**o**
16	10	*[DATA LINK ESCAPE]*	48	30	**0**	80	50	**P**	112	70	**p**
17	11	*[DEVICE CONTROL 1]*	49	31	**1**	81	51	**Q**	113	71	**q**
18	12	*[DEVICE CONTROL 2]*	50	32	**2**	82	52	**R**	114	72	**r**
19	13	*[DEVICE CONTROL 3]*	51	33	**3**	83	53	**S**	115	73	**s**
20	14	*[DEVICE CONTROL 4]*	52	34	**4**	84	54	**T**	116	74	**t**
21	15	*[NEGATIVE ACKNOWLEDGE]*	53	35	**5**	85	55	**U**	117	75	**u**
22	16	*[SYNCHRONOUS IDLE]*	54	36	**6**	86	56	**V**	118	76	**v**
23	17	*[ENG OF TRANS. BLOCK]*	55	37	**7**	87	57	**W**	119	77	**w**
24	18	*[CANCEL]*	56	38	**8**	88	58	**X**	120	78	**x**
25	19	*[END OF MEDIUM]*	57	39	**9**	89	59	**Y**	121	79	**y**
26	1A	*[SUBSTITUTE]*	58	3A	**:**	90	5A	**Z**	122	7A	**z**
27	1B	*[ESCAPE]*	59	3B	**;**	91	5B	**[**	123	7B	**{**
28	1C	*[FILE SEPARATOR]*	60	3C	**<**	92	5C	\	124	7C	**|**
29	1D	*[GROUP SEPARATOR]*	61	3D	**=**	93	5D	**]**	125	7D	**}**
30	1E	*[RECORD SEPARATOR]*	62	3E	**>**	94	5E	**^**	126	7E	**~**
31	1F	*[UNIT SEPARATOR]*	63	3F	**?**	95	5F	_	127	7F	*[DEL]*

Writing our own atoi

```
int myAtoi(char * str){







}
```

Writing our own atoi

```
int myAtoi(char * str){
    int ans = 0, i=0;
    while (str[i] != ‘\0’){
        ans = ans*10 + (str[i] – 48);
        i++;
    }
    return ans;
}
```

Implement your own strncp()

Implement your own strncp()

```
char * strncpy(char * dest, char * source, size_t n){




}
```

Implement your own strncp()

```
char * strncpy(char * dest, char * source, size_t n){
  if (dest == NULL)
    return NULL;
  char* p = dest

 while (*source && n){
    *dest = *source;
    n--;
    dest++;
    source++;
  }

 *dest = '\0';
  return ptr;
}
```

Question 1 Multiple Choice Concepts [12 pts]

Q1.1 Consider the code on the left of the following figure, and the frame layout on the right (we assume that `sizeof(int)=4`, and `sizeof(int*)=4` on this system):

Code

```
int a = 9;
int * p = &a;
int ** q= &p;
int data[2][2];
data[0][0] = 35;
data[0][1] = 87;
data[1][0] = 12;
data[1][1] = 200;
```

Frame Layout

Name	Address	Value
a	484-487	
p	480-483	
q	476-479	
	472-475	
	468-471	
data	464-467	
	460-463	

1. In the diagram above, fill in each box in the frame with the **numerical** (not conceptually: write numbers, do not draw arrows for pointers) value that it contains when the code shown here finishes executing.

2. What is the **type** of `data[1]`?

3. What is the **numerical value** of `data[1]`?

4. What is the **type** of `data[1][0]`?

5. What is the **type** of `&data[1][1]`?

6. What is the **numerical value** of `&data[1][1]`?

Question 1 Multiple Choice Concepts [12 pts]

Q1.1 Consider the code on the left of the following figure, and the frame layout on the right (we assume that `sizeof(int)=4`, and `sizeof(int*)=4` on this system):

Code

```
int a = 9;
int * p = &a;
int ** q= &p;
int data[2][2];
data[0][0] = 35;
data[0][1] = 87;
data[1][0] = 12;
data[1][1] = 200;
```

Frame Layout

Name	Address	Value
a	484-487	9
p	480-483	484
q	476-479	480
data	472-475	200
	468-471	12
	464-467	87
	460-463	35

1. In the diagram above, fill in each box in the frame with the **numerical** (not conceptually: write numbers, do not draw arrows for pointers) value that it contains when the code shown here finishes executing.

2. What is the **type** of `data[1]`?

   int *

3. What is the **numerical value** of `data[1]`?

   468

4. What is the **type** of `data[1][0]`?

   int

5. What is the **type** of `&data[1][1]`?

   int *

6. What is the **numerical value** of `&data[1][1]`?

   472

Q1.2 Consider the following two declarations:

```
const char * s1 = "Hello";
char s2[] = "Hello";
```

For each of the following statements, select whether or not it is true of s1, s2, neither or both (place a check mark in the correct box)

Statement	**True of s1**	**True of s2**	**Both**	**Neither**
s1 and/or s2 is an lvalue				
&s1[3] and/or &s2[3] is valid				
s1[0] and/or s2[0] is in read only memory				
s1 and/or s2 occupies 6 bytes in the frame				
s1 and/or s2 point at a null terminated string				
s1 and/or s2 should be freed				

Q1.2 Consider the following two declarations:

```
const char * s1 = "Hello";
char s2[] = "Hello";
```

For each of the following statements, select whether or not it is true of s1, s2, neither or both (place a check mark in the correct box)

Statement	**True of s1**	**True of s2**	**Both**	**Neither**
s1 and/or s2 is an lvalue	X			
&s1[3] and/or &s2[3] is valid			X	
s1[0] and/or s2[0] is in read only memory	X			
s1 and/or s2 occupies 6 bytes in the frame		X		
s1 and/or s2 point at a null terminated string			X	
s1 and/or s2 should be freed				X

```c
#include <stdio.h>
#include <stdlib.h>
void f(int x, int * p, int ** q) {
  x = x + 7;
  *p = **q - x;
  *q = p;
}
```

```c
int main(void) {
  int a = 6;
  int b = 3;
  int c = 1;
  int * data[] = {&a, &b, &c};
  int ** q = &data[1];
  **q = 43;
  q[0] = q[1];
  printf("a=%d, b=%d, c=%d\n",a,b,c);
  for (int i = 0; i < 3; i++) {
    *data[i] = *data[i] + 11;
  }
  printf("a=%d, b=%d, c=%d\n",a,b,c);
  f(a, &b, &data[2]);
  *q[1] = *q[1]+2;
  printf("a=%d, b=%d, c=%d\n",a,b,c);
  return EXIT_SUCCESS;
}
```

Answer to the previous problem:

6	43	1
17	43	23
17	1	23