C Programming Language

(5th class)

Dohyung Kim

Assistant Professor @ Department of Computer Science

Today …

- Array
- String
- Pointer

Why do we need arrays in C?

- **Consider that you would calculate manage 100 students’ mid-term scores.**
- **How would you add them?**
  - total = first + second + third + … + hundred;
- **What if you are supposed to give grades to 100 students?**

```
int first, second, …, hundred;
char grade1, grade2, …, grade100;

if (first > 90) {
    grade1 = ‘A’;
}elseif (first > 80){
…

if (second > 90){
     grade1 = ‘A’;
} elseif (first > 80){
…
```

Arrays in C

- **Arrays act to store related data under a single variable name with an index, also know as a subscript**

```
char grade[100];
int    score[100];
for (i=0; i<100; i++) {
   if (score[i] > 90){ grade[i] = ‘A’;}
   else if(score[i] > 80){ …;}
      …
}
```

- **Declaration of arrays**
  - int scores[100];
  - char name[100][20];
- **Initialization while declaring arrays**
  - int numbers[7] = {0, 0, 0, 1, 1, 1, 1};
  - int numbers[ ] = {0, 0, 0, 1, 1, 1, 1};
  - int numbers[7] = {255};   // int numbers[7] = {255, 0, 0, 0, 0, 0, 0}

Indexing

- Arrays in C are indexed starting at 0.

```
int numbers[ ] = {0, 1, 2, 3, 4};
x = numbers[2];
printf (“the number in x: %d\n”, x);
```

- An out of bound access does not always cause a runtime error (Of course, it’s a semantic error.).

```
y = numbers[5];
```

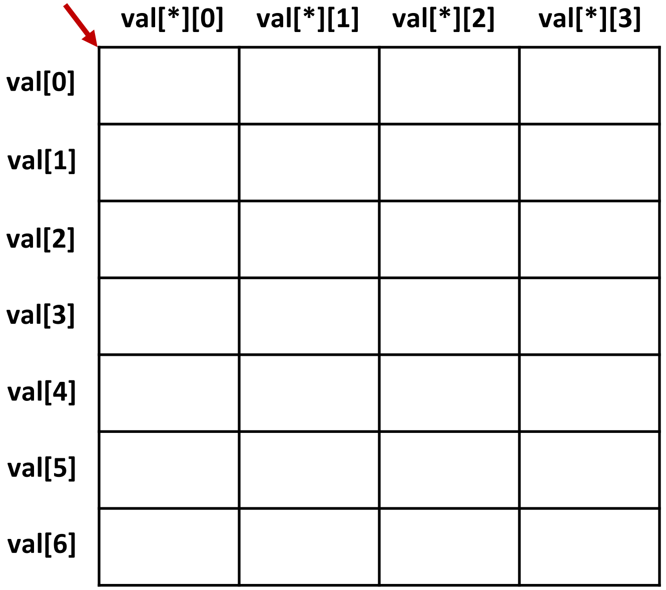
- To alleviate indexing problem, the sizeof() expression is commonly used.

```
for (i = 0;  i < sizeof(numbers)/sizeof(int);  i++){
   printf (“%d”, numbers[i]);
}
```

Multi-dimensional arrays

- Consider that seven students have scores in four different subjects (math, physics, English, history).
- How to store them in your program?
- A possible approach is to use multi-dimensional arrays

```
int val[7][4];
```



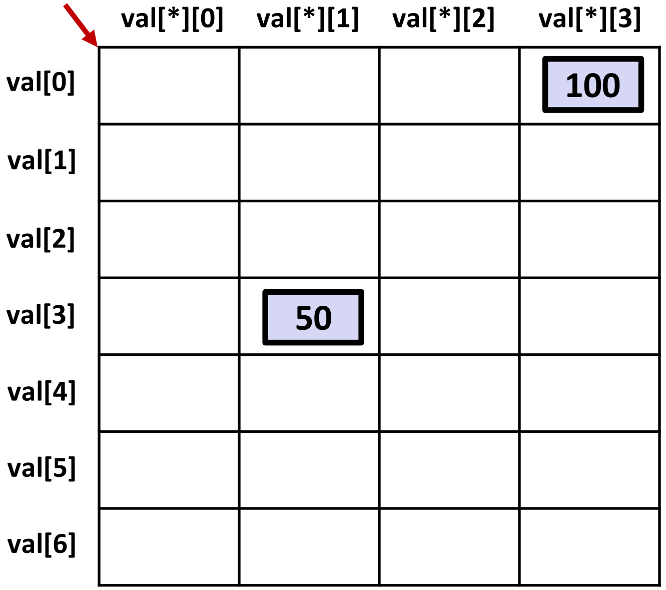
	val[*][0]	val[*][1]	val[*][2]	val[*][3]
val[0]				
val[1]				
val[2]				
val[3]				
val[4]				
val[5]				
val[6]				

Multi-dimensional arrays

- Consider that seven students have scores in four different subjects (math, physics, English, history).
- How to store them in your program?
- A possible approach is to use multi-dimensional arrays

```
int val[7][4];

val[3][1] =50;
val[0][3] =100;
```



	val[*][0]	val[*][1]	val[*][2]	val[*][3]
val[0]				100
val[1]				
val[2]				
val[3]		50		
val[4]				
val[5]				
val[6]				

String as a Multi-dimensional array

- **char names[3][20] = { “Albert”, “John”, “Mary”};**

```
char names[3][20] = {“Albert”, “John”, “Mary”};
int i;
for (i=0; i<3; i++){
   printf(“person’s name : %s\n”, names[i]);
}

printf(“character : %c\n”, name[1][2]);
```

Strings

- C has no string handling facilities built in.
- Strings are defined as arrays of characters with the null terminating character automatically added to the end.
  - char name [] = “Albert”

In memory :

A	l	b	e	r	t	\0

  - char name [] = {‘A’, ‘l’, ‘b’, ‘e’, ‘r’, ‘t’, ‘\0’}

```
char string1[] = “this is a very, very long ”
                 “string that requires two lines.”;
char string2[] = “this is a very, very long \n string that requires
two lines.”

printf(“The first string: %s\n”, string1);
printf(“The first string: %s\n”, string2);
```

#include<string.h>

- **Library of string handling routine**
  - strcat - concatenate two strings
  - strcmp - compare two strings
  - strcpy - cpy a string
  - strlen - get string length
  - strchr - string scanning operation
  - strncat - concatenate one string with part of another
  - strcmp - compare parts of two string
  - …

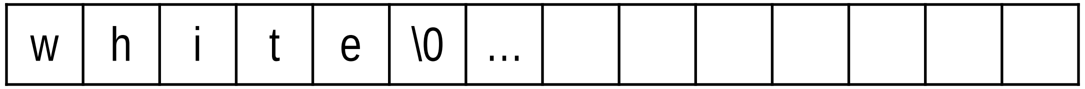
string.h

- **strcpy, strcat**

```
#include<stdio.h>
#include<string.h>

void main(){
   char colors[3][10] = {“red”, “blue”, “white”};
   char widths[3][10] = {“thin”, “medium”, “bold”};
   char myPen[20];
   strcpy(myPen, colors[2]);
   strcpy(myPen, widths[1]);
   printf(“My pen is : %s\n”, myPen);
   strcat(myPen, widths[1]);
   printf(“My pen is : %s\n”, myPen);
}
```

myPen



w	h	i	t	e	\0	…							

string.h

- **strcpy, strcat**

```
#include<stdio.h>
#include<string.h>

void main(){
   char colors[3][10] = {“red”, “blue”, “white”};
   char widths[3][10] = {“thin”, “medium”, “bold”};
   char myPen[20];
   strcpy(myPen, colors[2]);
   strcpy(myPen, widths[1]);
   printf(“My pen is : %s\n”, myPen);
   strcat(myPen, widths[1]);
   printf(“My pen is : %s\n”, myPen);
}
```

myPen

m	e	d	i	u	m	\0	…						

string.h

- **strcpy, strcat**

```
#include<stdio.h>
#include<string.h>

void main(){
   char colors[3][10] = {“red”, “blue”, “white”};
   char widths[3][10] = {“thin”, “medium”, “bold”};
   char myPen[20];
   strcpy(myPen, colors[2]);
   strcpy(myPen, widths[1]);
   printf(“My pen is : %s\n”, myPen);
   strcat(myPen, widths[1]);
   printf(“My pen is : %s\n”, myPen);
}
```

myPen

m	e	d	i	u	m	m	e	d	i	u	m	\0	…

string.h

- Compare two strings

```
#include<stdio.h>
#include<string.h>

void main(){
   char name1[] = “Albert”;
   char name2[] = “Albert”;
   if(name1 == name2){
       puts (“equal\n”);
   }else{
       puts (“not equal\n”);
   }
}
```

string.h

- strcmp

```
#include<stdio.h>
#include<string.h>

void main(){
   char name1[] = “Albert”;
   char name2[] = “Albert”;
   if(!strcmp(name1, name2)){
       puts (“equal\n”);
   }else{
       puts (“not equal\n”);
   }
}
```

* strcmp(name1, name2) returns

- 0 if name1 is equal to name2
- Negative value if name1 appears before name2 in lexicographical order
- Positive value if name1 appears after name2 in lexicographical order

string.h

- strncmp

```
#include<stdio.h>
#include<string.h>

void main(){
   char name1[] = “AlbertKim”;
   char name2[] = “AlbertLee”;
   if(!strncmp(name1, name2, 5)){
       puts (“equal\n”);
   }else{
       puts (“not equal\n”);
   }

   if(!strncmp(name1, name2, 9)){
       puts (“equal\n”);
   }else{
       puts (“not equal\n”);
   }
}
```

Pointers and arrays

- **A Pointer is a value that designates the address (i.e., the location in memory), of some value.**
  - How to declare them
  - How to assign to them
  - How to reference the value to which the pointer points (known as dereferencing)
  - How they relate to arrays
- **Dereferencing operator ‘*’**
- **Pointers can reference any data type, even functions**

Assigning values to pointers

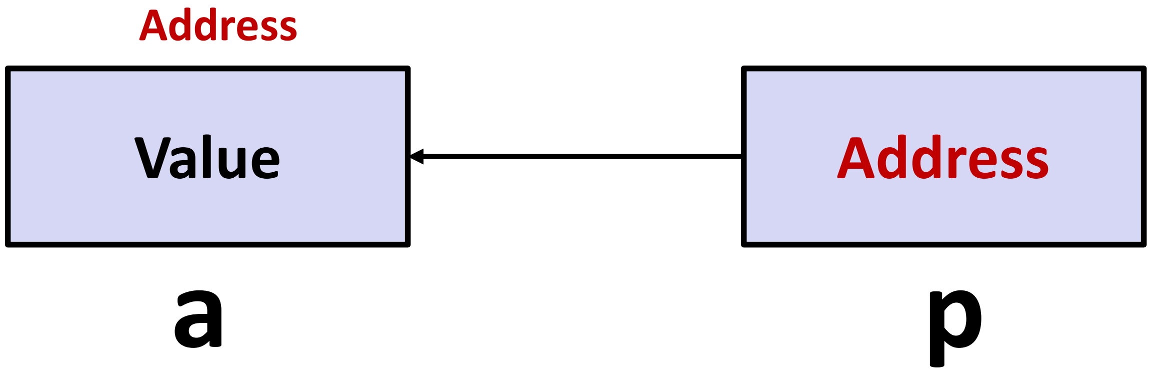
- **Pointer examples**

```
#include<stdio.h>

void main(){
   int a, b;
   double c;
   int *pA, *pB;
   double *pC;

   pA = &a;
   pB = pA;
   pC = &c;
   …
   …
}
```

Pointer dereferencing



Address

Value

Address

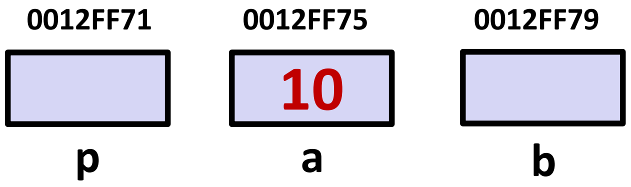
a

p

The pointer p points to the variable a

```
int *p;
int a, b;

a = 10;
p = &a;
b = *p;
```



0012FF71

0012FF75

0012FF79

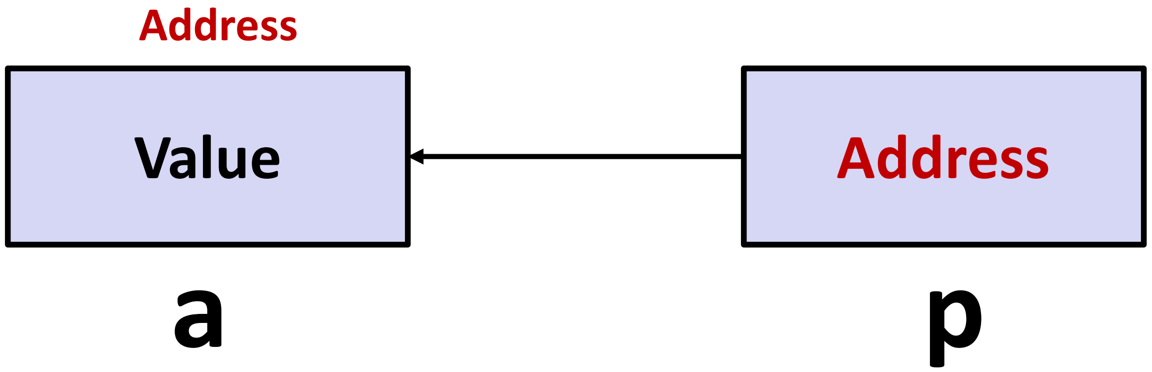
10

p

a

b

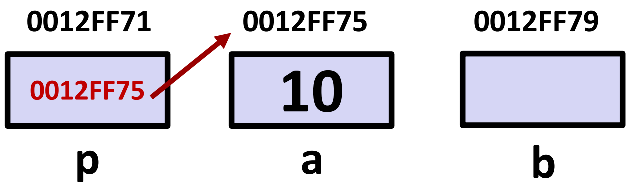
Pointer dereferencing



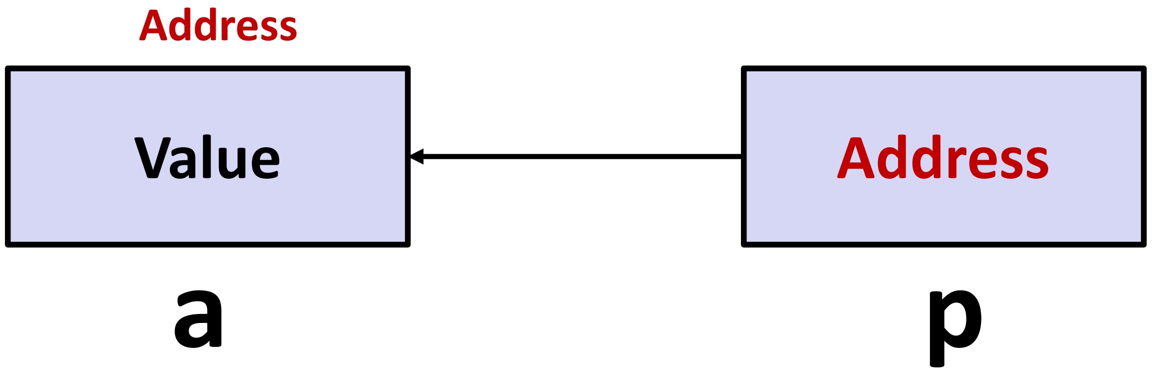
The pointer p points to the variable a

```
int *p;
int a, b;

a = 10;
p = &a;
b = *p;
```



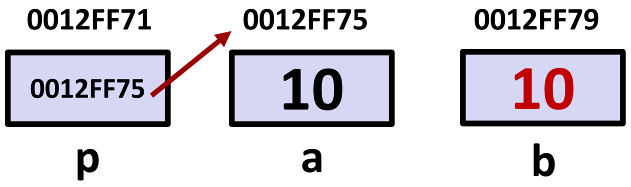
Pointer dereferencing



The pointer p points to the variable a

```
int *p;
int a, b;

a = 10;
p = &a;
b = *p;
```



Pointers and arrays

- A variable declared as an array of some type acts as a pointer to that type. When used by itself, it points to the first element of the array.

```
int checkNumbers(int inputNumber, int* numbers);
/* int checkNumbers(int inputNumber, int[] numbers);*/


void main(){
   int result[3];
   int inputNum[3], targetNum[3] = {1, 2, 3}; // in your code, numbers may be randomly generated.
   …                                          // take your inputs and store them in inputNum[3]
   for (i = 0; i<3; i++){
      result[i] = checkNumber(inputNumber[i], targetNum);
   }
   ….
}
```

Pointers and arrays

- **A pointer can be indexed like an array name.**

```
int val[3][4];
int *pf;

pf = &val[0][0];

*(pf+1) = 1.3;         /* assigns 1.3 to val[0][1] */
*(pf+8) = 2.3;         /* assigns 2.3 to val[2][0] */
```

Pointers and arrays

- A pointer can be indexed like an array name.

```
int val[3][4];
int *pf;

pf = &val[0][0];

*(pf+1) = 3;        /* assigns 3 to val[0][1] */
*(pf+8) = 2;        /* assigns 2 to val[2][0] */
```

What?

Revisit Multi-dimensional arrays

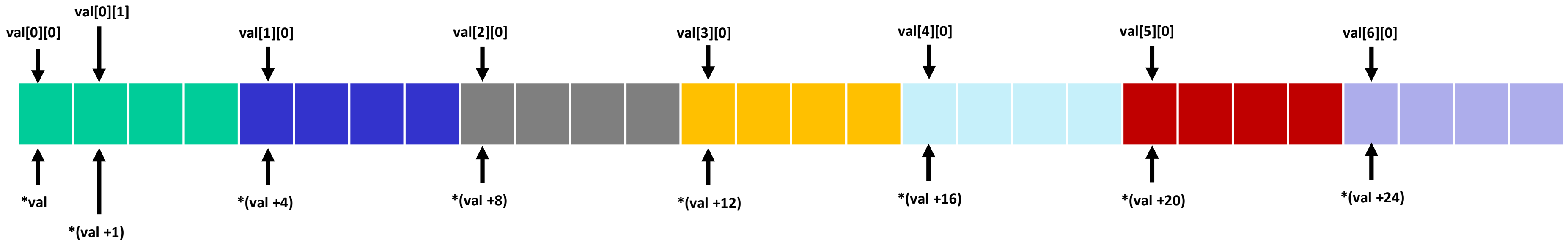
```
int val[7][4];
```

- Logical View

val

	val[*][0]	val[*][1]	val[*][2]	val[*][3]
val[0]				
val[1]				
val[2]				
val[3]				
val[4]				
val[5]				
val[6]				

- Physical View



Examples

```
double  linearM[30];
double  *M[6];                    /* Consider the statement int *p, p has an address.
                                     p is replaced by M[6] in the same statement */

M[0] = linearM;                   /* 5 – 0 = 5 elements in row */
M[1] = linearM + 5;               /* 11 - 5 = 6 elements in row */
M[2] = linearM + 11;              /* 15 - 11 = 4 elements in row */
M[3] = linearM + 15;              /* 21 - 15 = 6 elements in row */
M[4] = linearM + 21;              /* 25 - 11 = 4 elements in row */
M[5] = linearM + 25;              /* 30 - 25 = 5 elements in row */

M[3][2] = 3.66                    /* assigns 3.66 to linearM[17] */
M[3][-3] = 1.44                   /* refers to linearM[12], negative indices are sometimes
                                     useful. But avoid using them as much as possible */
```

Pointer and String

```
char * myString = “Pointer and String”;
/*char myString[]  = “Pointer and String” */
```

```
char myString[]  = {‘P’, ‘o’, ‘i’, ‘n’, ‘t’, ‘e’, ‘r’, ‘ ’, ‘a’, ‘n’, ‘d’, ‘ ’, ‘S’, ‘t’, ‘r’, ‘I’, ‘n’, ‘g’, ‘\0’}
```

```
char *myColors[] = {“red”, “blue”, “yellow”};
/* char *myColors[3] = {“red”, “blue”, “yellow”}; */
```

```
char myString[]  = “Pointer and String”
char *yourString;

yourString = myString;
```

What we have covered today

- **Array**
  - Multiple data could be stored using a single variable name along with an index
- **String**
  - An array of characters with the null terminating character
- **Pointer**
  - A Pointer is a value that designates the address
  - Address is corresponding to the location in memory

Q and A