The C Language

Week 03 – Lecture

Arrays, Structures and Function pointers

Team

- Instructor
  - Giancarlo Succi
- Teaching Assistants
  - Nikita Lozhnikov (also Tutorial Instructor)
  - Manuel Rodriguez
  - Shokhista Ergasheva

Sources

- These slides have been adapted from the original slides of the adopted book:
  - Tanenbaum & Bos, Modern Operating Systems: 4th edition, 2013
    Prentice-Hall, Inc.

  and customized for the needs of this course.
- Additional input for the slides are detailed later

Arrays (1)

- An array is a collection of data elements that are of the same type (e.g., a collection of integers, collection of characters, collection of doubles).

Arrays (2)

Currency Last Trade	U.S. $ N/A	Aust $ Oct 14	U.K. £ Oct 14	Can $ Oct 14	DMark Oct 14	FFranc Oct 14	¥en Oct 14	SFranc Oct 14	Euro 12:39AM
U.S. $	1	0.6493	1.663	0.675	0.5513	0.1644	0.009316	0.6784	1.082
Aust $	1.54	1	2.562	1.04	0.8491	0.2532	0.01435	1.045	1.666
U.K. £	0.6012	0.3904	1	0.4058	0.3314	0.09883	0.005601	0.4079	0.6505
Can $	1.481	0.9619	2.464	1	0.8167	0.2435	0.0138	1.005	1.603
DMark	1.814	1.178	3.017	1.224	1	0.2982	0.0169	1.231	1.963
FFranc	6.083	3.95	10.12	4.106	3.354	1	0.05667	4.127	6.582
¥en	107.3	69.7	178.5	72.46	59.18	17.64	1	72.82	116.1
SFranc	1.474	0.9571	2.452	0.995	0.8126	0.2423	0.01373	1	1.595
Euro	0.9242	0.6001	1.537	0.6238	0.5095	0.1519	0.00861	0.627	1

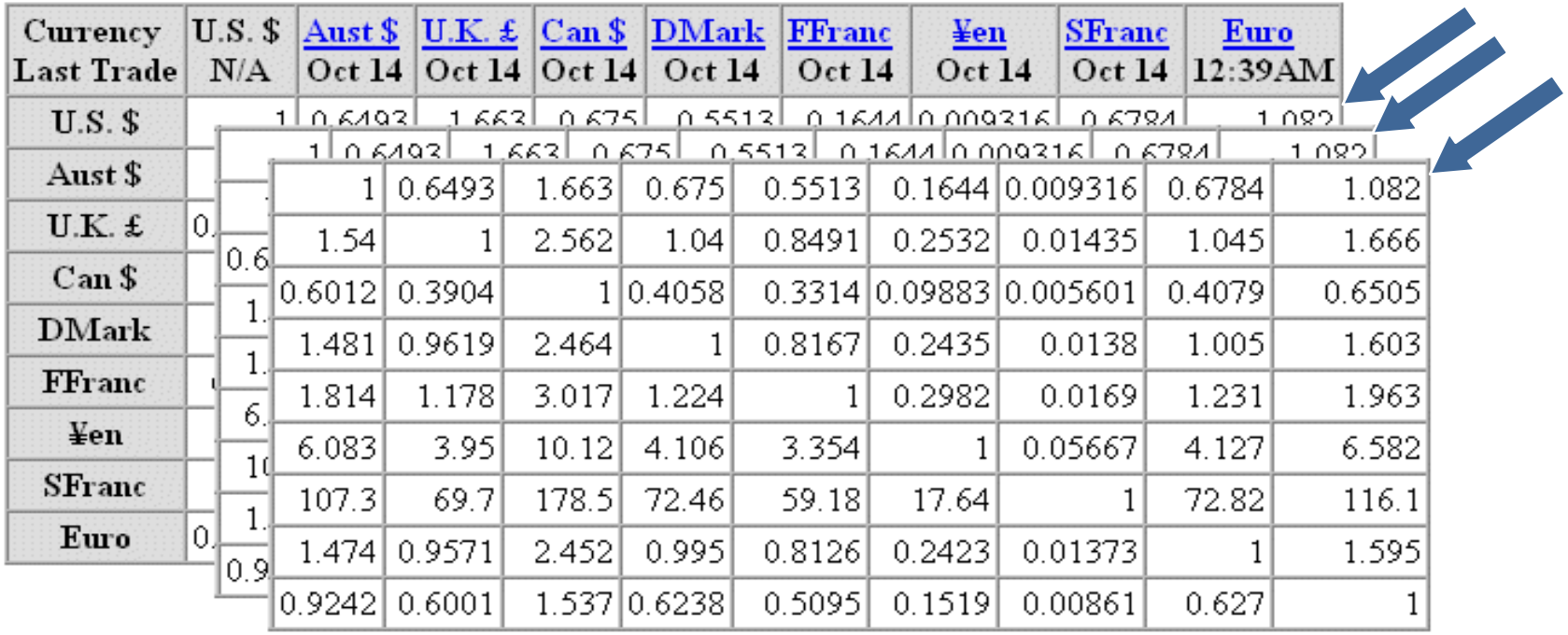
A table is an example of two-dimensional array

Arrays (3)

Currency	U.S. $	Aust $	U.K. £	Can $	DMark	FFranc	¥en	SFranc	Euro
Last Trade	N/A	Oct 14	Oct 14	Oct 14	Oct 14	Oct 14	Oct 14	Oct 14	12:39AM
U.S. $	1	0.6493	1.663	0.675	0.5513	0.1644	0.009316	0.6784	1.082

One-dimensional array

Arrays (4)



Currency Last Trade	U.S. $ N/A	Aust $ Oct 14	U.K. £ Oct 14	Can $ Oct 14	DMark Oct 14	FFranc Oct 14	¥en Oct 14	SFranc Oct 14	Euro 12:39AM
U.S. $	1	0.6493	1.663	0.675	0.5513	0.1644	0.009316	0.6784	1.082
Aust $	1.54	1	2.562	1.04	0.8491	0.2532	0.01435	1.045	1.666
U.K. £	0.6012	0.3904	1	0.4058	0.3314	0.09883	0.005601	0.4079	0.6505
Can $	1.481	0.9619	2.464	1	0.8167	0.2435	0.0138	1.005	1.603
DMark	1.814	1.178	3.017	1.224	1	0.2982	0.0169	1.231	1.963
FFranc	6.083	3.95	10.12	4.106	3.354	1	0.05667	4.127	6.582
¥en	107.3	69.7	178.5	72.46	59.18	17.64	1	72.82	116.1
SFranc	1.474	0.9571	2.452	0.995	0.8126	0.2423	0.01373	1	1.595
Euro	0.9242	0.6001	1.537	0.6238	0.5095	0.1519	0.00861	0.627	1

Three-dimensional array (3rd dimension is the day)

Array Applications

- Given a list of test scores, determine the maximum and minimum scores
- Read in a list of student names and rearrange them in alphabetical order (sorting)
- Given the height measurements of students in a class, output the names of those students who are taller than average

Array Declaration (1)

- Syntax:

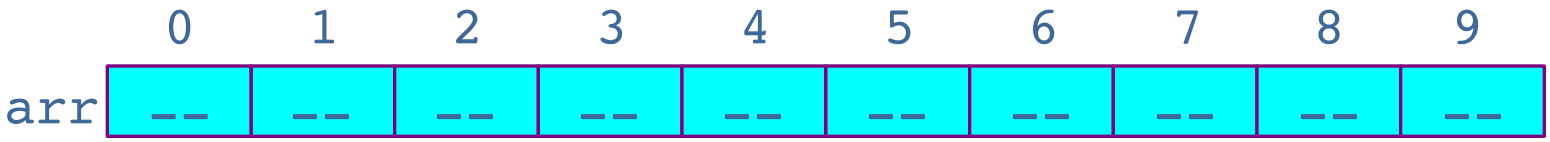
  <type> <arrayName>[<array_size>]

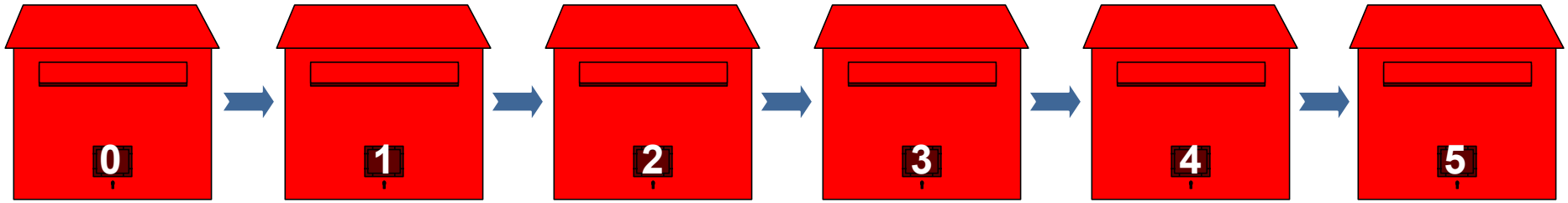
  Ex. int arr[10];

- The array elements are all values of the type **<type>**
- The size of the array is indicated by **<array_size>**, the number of elements in the array
- <array_size> must be an int constant or a constant expression. Note that an array can have multiple dimensions

Array Declaration (2)

```
// array of 10 uninitialized ints
int arr[10];
```





Subscripting (1)

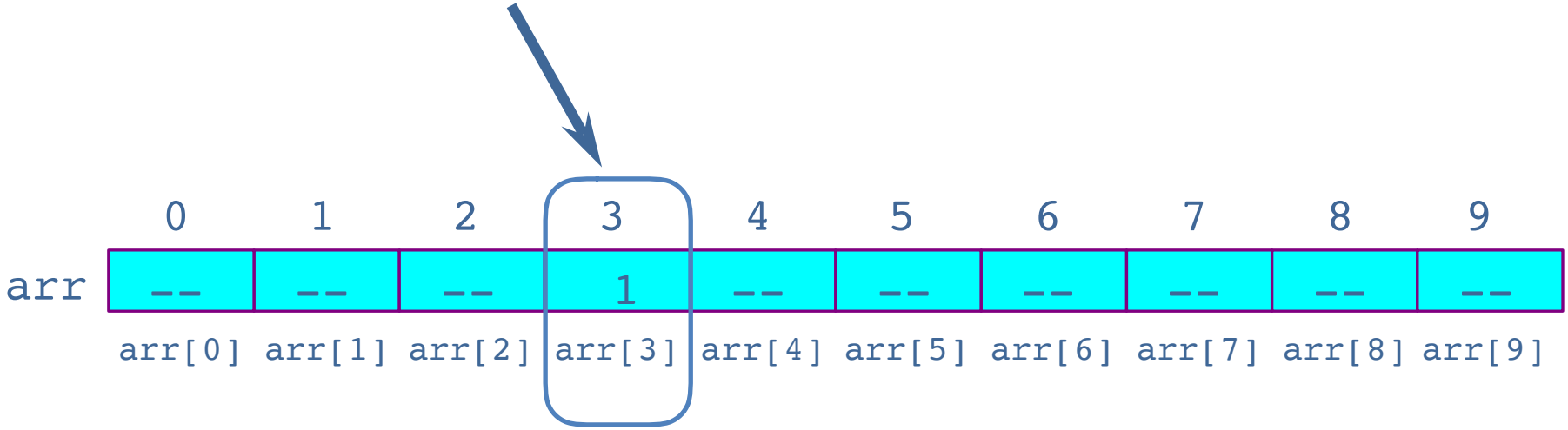
- Let's consider an example of an array of 10 integers:

```
int arr[10];  // array of 10 ints
```

- To access an individual element we must apply a subscript to array named **`arr`**
  - A subscript is a bracketed expression
  - The expression in the brackets is known as the index
  - First element of array has index 0: arr[0]
  - Second element of array has index 1, and so on: arr[1], arr[2], …
  - Last element's index is one less than the size of the array: arr[9]

Subscripting (2)

```
// array of 10 uninitialized ints
int arr[10];

arr[3] = 1;
int x = arr[3];
```



Arrays Example 1: Subscripting

```c
#include <stdio.h>
#include <stdlib.h>

int main(void) {
    int index, arr[10];  // Array of 10 integers

    // Read in 10 elements
    printf("Enter 10 integers: \n");
    for(index = 0; index < 10; index ++)
        scanf("%d", &arr[index]);

    // Print the elements
    printf("The integers are: ");
    for(index = 0; index < 10; index ++)
        printf("%d ", arr[index]);

    return EXIT_SUCCESS;
}
```

Arrays Example 1: Sorting (1)

```
void swap(int *first, int *second);

int main(void) {

    int arr[3];  // input integers
    // Read in three elements.
    printf("Enter three integers: \n");
    scanf("%d", &arr[0]);
    scanf("%d", &arr[1]);
    scanf("%d", &arr[2]);
    if (arr[0] > arr[1]) swap (&arr[0], &arr[1]);
    if (arr[1] > arr[2]) swap (&arr[1], &arr[2]);
    if (arr[0] > arr[1]) swap (&arr[0], &arr[1]);
    printf("The sorted integers are: %d, %d, %d",
              arr[0], arr[1], arr[2]);

    return EXIT_SUCCESS;
}
```

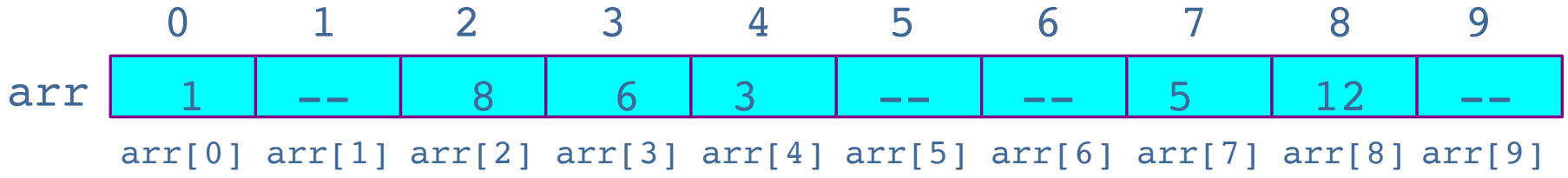
Arrays Example 1: Sorting (2)

```
// Function for swapping two integers

void swap(int *first, int *second) {
   int temp;
   temp = *first;
   *first = *second;
   *second = temp;
}
```

Array Elements Manipulation

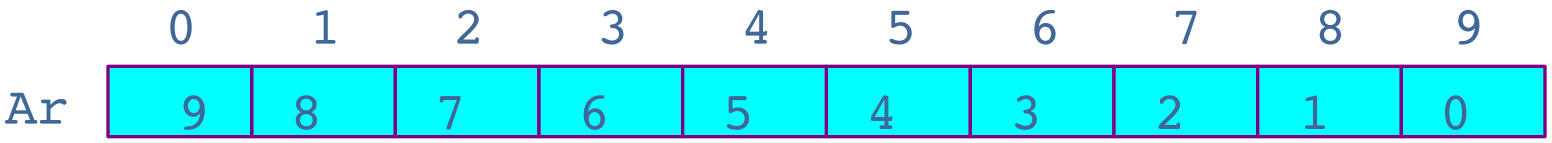
- Consider

```
int arr[10], i = 7, j = 2, k = 4;
arr[0] = 1;
arr[i] = 5;
arr[j] = arr[i] + 3;
arr[j+1] = arr[i] + arr[0];
arr[arr[j]] = 12;
scanf(arr[k]); // where the next input
value is 3
```

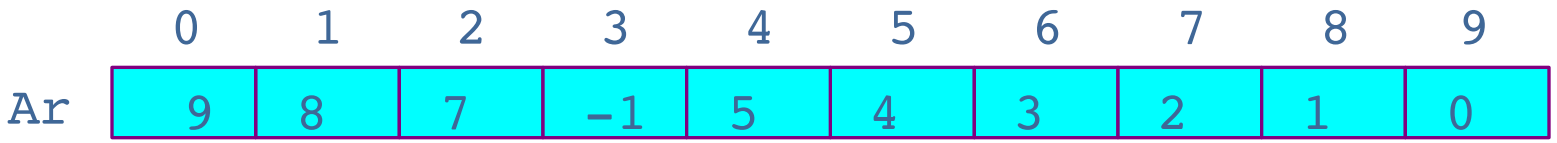


Array Initialization

```
int arr[10] = {9, 8, 7, 6, 5, 4, 3, 2, 1, 0};
```




```
arr[3] = -1;
```



Initializing Arrays With Random Values

- The following loop initializes the array myList with random values between 0 and 99:

```
#define ARRAY_SIZE 100
…
for (int i = 0; i < ARRAY_SIZE; i++)
{
 myList[i] = rand() % 100;
}
```

Printing Arrays

- To print an array, you have to print each element in the array using a loop like the following:

```
#define ARRAY_SIZE 100
…
for (int i = 0; i < ARRAY_SIZE; i++)
{
 printf("%d", myList[i]);
}
```

Copying Arrays

- Can you copy array using a syntax like this?

```
list = myList;
```

- This is not allowed in C. You have to copy individual elements from one array to the other as follows:

```
for (int i = 0; i < ARRAY_SIZE; i++)
{
  list[i] = myList[i];
}
```

Summing All Elements

- Use a variable named ***total*** to store the sum. Initially *total* is 0. Add each element in the array to *total* using a loop like this:

```
double total = 0;
for (int i = 0; i < ARRAY_SIZE; i++)
{
  total += myList[i];
}
```

Finding the Largest Element

- Use a variable named ***max*** to store the largest element. Initially *max* is *myList[0]*. To find the largest element in the array *myList*, compare each element in *myList* with *max*, update *max* if the element is greater than max:

```
double max = myList[0];
for (int i = 1; i < ARRAY_SIZE; i++)
{
  if (myList[i] > max)
    max = myList[i];
}
```

Finding the Smallest Index of the Largest Element

```
double max = myList[0];
int indexOfMax = 0;

for (int i = 1; i < ARRAY_SIZE; i++)
{
  if (myList[i] > max)
  {
    max = myList[i];
    indexOfMax = i;
  }
}
```

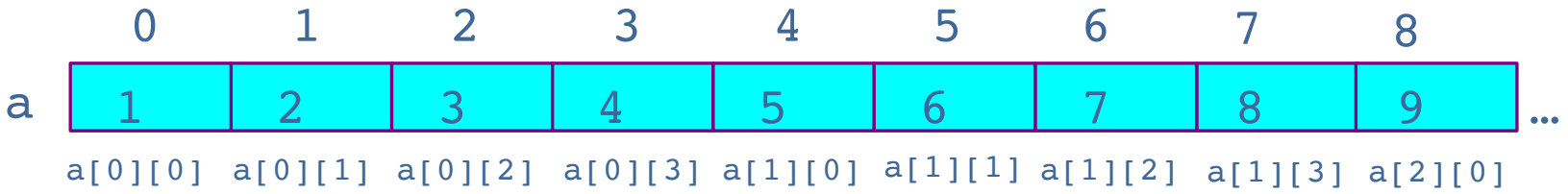
Shifting Elements

```
// Retain the first element
double temp = myList[0];

// Shift elements left
for (int i = 1; i < ARRAY_SIZE; i++)
{
  myList[i - 1] = myList[i];
}

/* Move the first element
to fill in the last position */
myList[ARRAY_SIZE - 1] = temp;
```

Multidimensional Array

```
int a[3][4] =
{
      {1,2,3,4},
      {5,6,7,8},
      {9,10,11,12}
};
```

In memory:



Structures

- A **structure** is a collection of related data items, possibly of different types
- A structure type in C is called **struct**
- A struct is heterogeneous in that it can be composed of data of different types
- In contrast, **array** is homogeneous since it can contain only data of the same type

Structures

- Structures hold data that belong **together**
- Examples:
  - Student record: student id, name, major, gender, start year, …
  - Bank account: account number, name, currency, balance, …
  - Address book: name, address, telephone number, …
- In database applications, structures are called records

Structures

- Individual components of a struct type are called **members** (or **fields**)
- Members can be of different types (simple, array or struct)
- A struct is named as a **whole** while individual members are named using **field identifiers**
- Complex data structures can be formed by defining arrays of structs

Structure Basics

- Definition of a structure:

```
struct <struct-type>{
  <type> <identifier_list>;
  <type> <identifier_list>;
  ...
} ;
```

Each identifier defines a *member* of the structure

- Example:

```
struct Date {
  int day;
  int month;
  int year;
};
```

The Date structure has 3 members: day, month & year.

Structure Examples

- Example:

```
struct StudentInfo{
  int Id;
  int age;
  char Gender;
  double CGA;
};
```

The *StudentInfo* structure has 4 members of different types

- Example:

```
struct StudentGrade{
  char Name[15];
  char Course[9];
  int Lab[5];
  int Homework[3];
  int Exam[2];
};
```

The *StudentGrade* structure has 5 members of different array types

Structure Examples

- Example:

```
struct BankAccount{
  char Name[15];
  int AcountNo[10];
  double balance;
  Date Birthday;
};
```

The *BankAcount* structure has simple, array and structure types as members

- Example:

```
struct StudentRecord{
  char Name[15];
  int Id;
  char Dept[5];
  char Gender;
};
```

The StudentRecord structure has 4 members

Structure Declaration

- Declaration of a variable of struct type:

```
struct <struct-type> <identifier_list>;
```

- Example:

```
struct Date d1, d2;
```

- *d1* and *d2* are variables of *Date* type

Structure Members Access (1)

- The members of a struct type variable are accessed with the dot (.) operator:

```
<struct-variable>.<member_name>;
```

- Example:

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

struct Student {
  int id;
  char name[50];
  char dept[10];
  char gender;
};
```

Structure Members Access (2)

```
int main(void) {
    struct Student s1;

    s1.id = 811;
    strcpy(s1.name, "Stanislav Litvinov");
    strcpy(s1.dept, "MSIT-SE");
    s1.gender = 'M';
    printf("The student is ");
    switch (s1.gender) {
    case 'F': printf("Ms. "); break;
    case 'M': printf("Mr. "); break;
    }
    printf("%s", s1.name);

    return EXIT_SUCCESS;
}
```

Struct-to-Struct Assignment

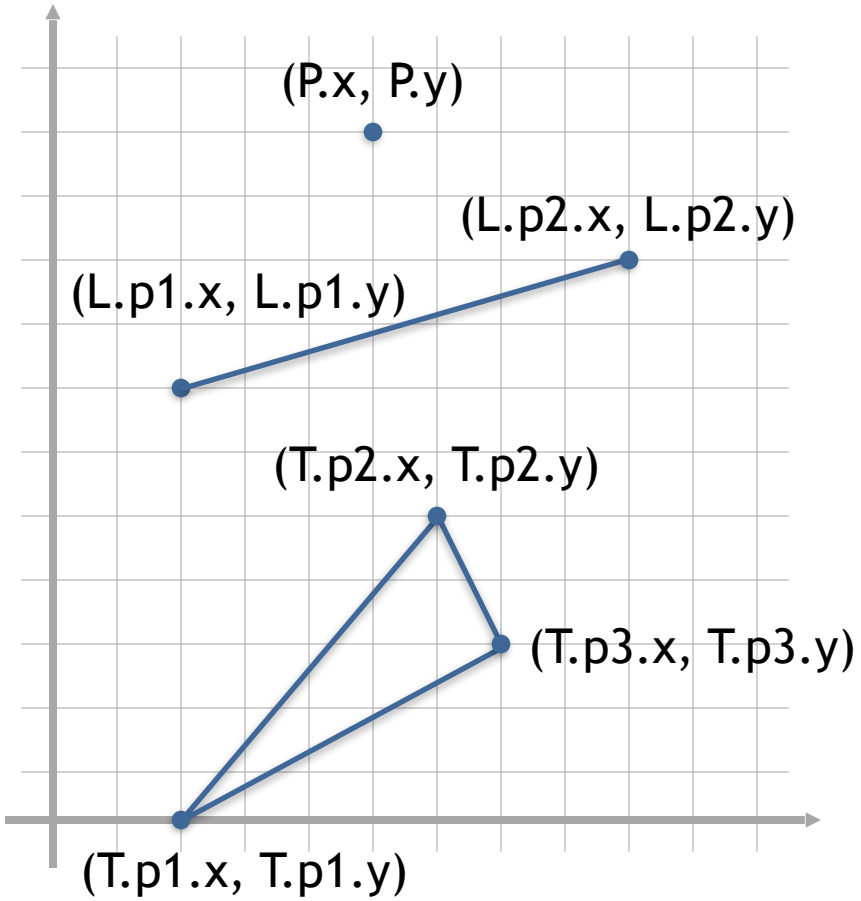
- The values contained in one *struct* type variable can be assigned to another variable of the same *struct* type
- Example:

```
…
struct Student s1, s2;
   s1.id = 811;
   strcpy(s1.name, "Stanislav Litvinov");
   strcpy(s1.dept, "MSIT-SE");
   s1.gender = 'M';

   s2 = s1;
…
```

Nested Structures (1)

- We can nest structures inside structures. Some examples are:

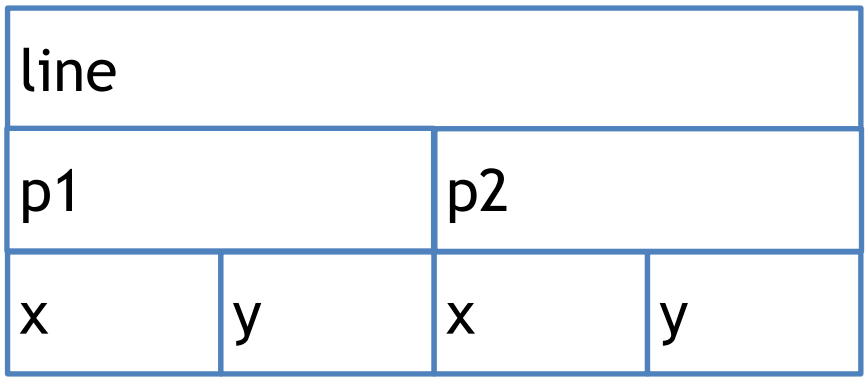
```
struct point {
  double x, y;
};

struct line {
  point p1, p2;
};

struct triangle {
  point p1, p2, p3;
};

struct point P;
struct line L;
struct triangle T;
```

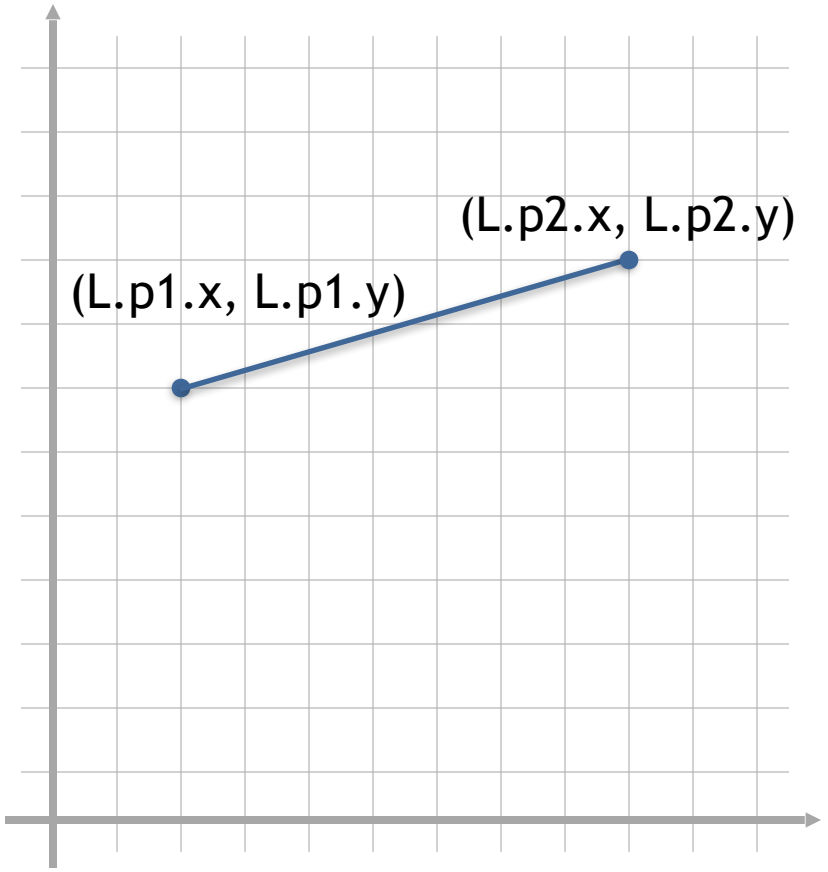


Nested Structures (2)

- We can nest structures inside structures

```
struct line {
  point p1, p2;
};
line L;
```

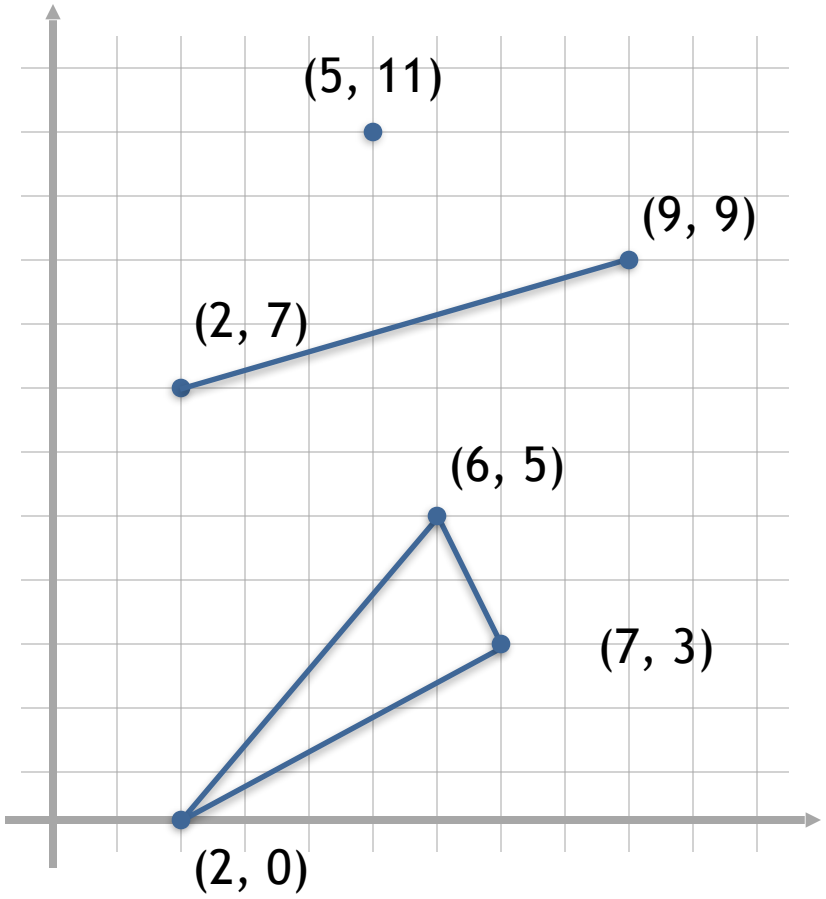




Exercise (1)

- Assign values to the variables P, L, and T using the picture:

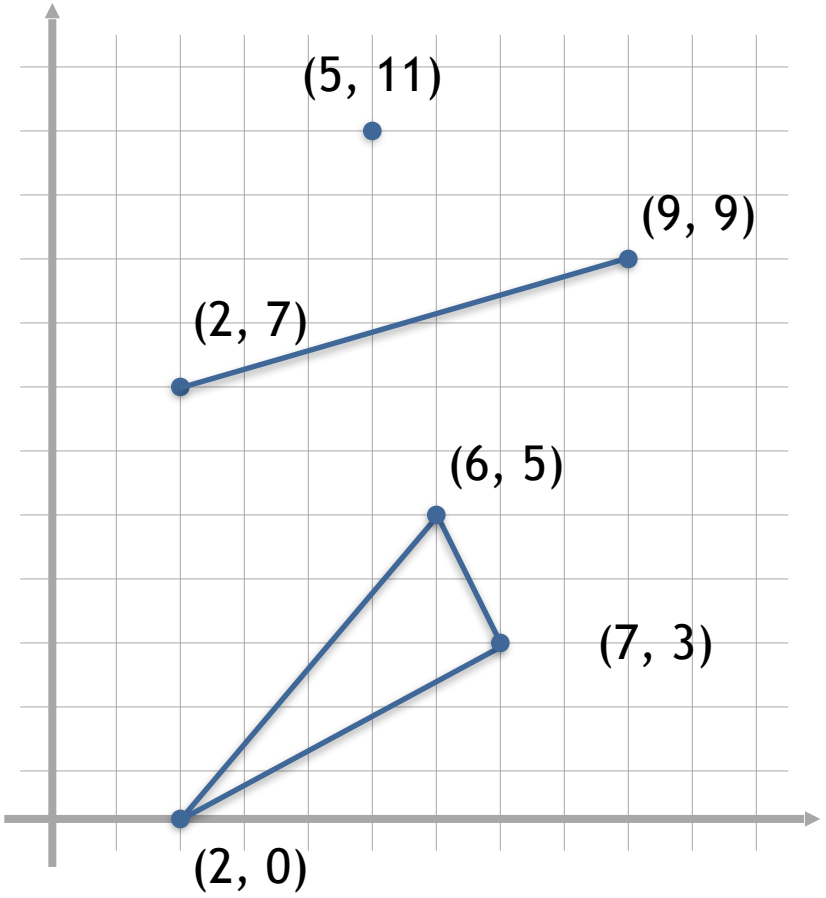
```
point P;
line L;
triangle T;
```



- Ex. 3: Graph a point
- Ex. 4: Graph a line
- Ex. 5: Graph a triangle

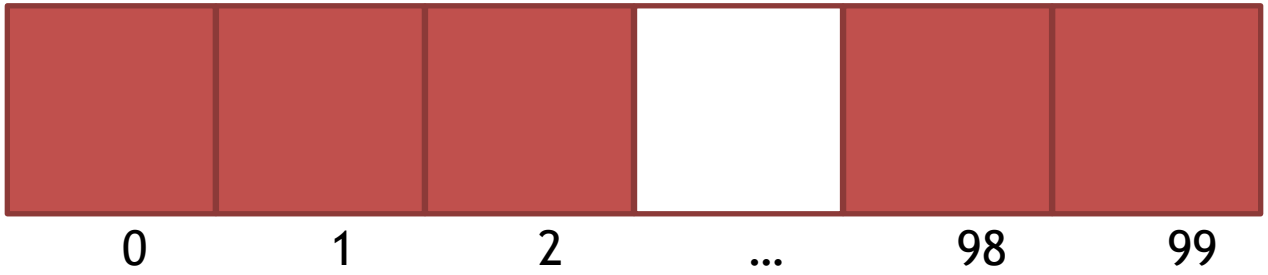
Exercise (2)

```
point P;
line L;
triangle T;
P.x = 5;
P.y = 11;

L.p1.x = 2;
L.p1.y = 7;
L.p2.x = 9;
L.p2.y = 9;

T.p1.x = 2;
T.p1.y = 0;
T.p2.x = 6;
T.p2.y = 5;
T.p3.x = 7;
T.p3.y = 3;
```

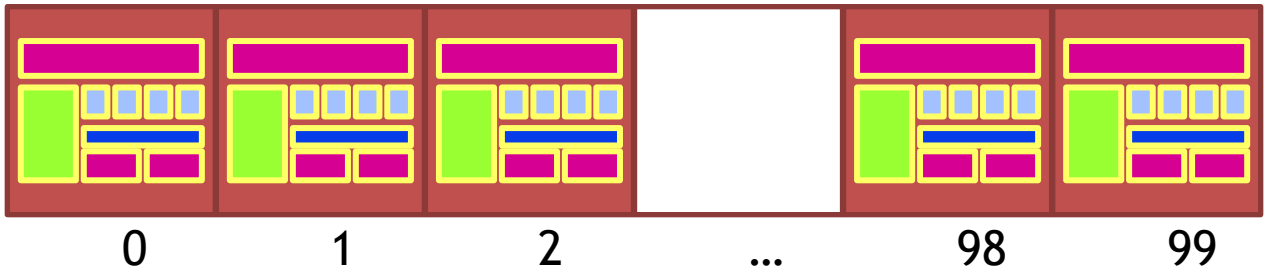


Arrays of Structures (1)

- An ordinary array: One type of data



- An array of structs: Multiple types of data in each array element



Arrays of Structures (2)

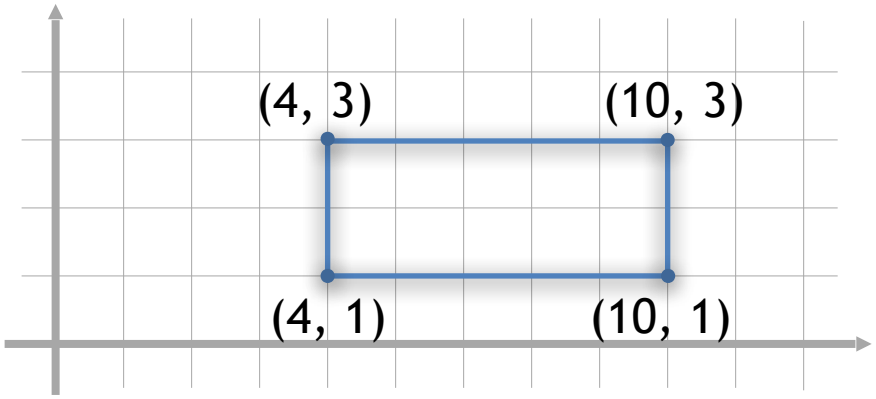
- We often use arrays of structures.
  Example:

```
struct Student class[100];

class[98].id = 811;
strcpy(class[98].name, "Stanislav Litvinov");
strcpy(class[98].dept, "MSIT-SE");
class[98].gender = 'M';

class[0] = class[98];
```

Arrays Inside Structures (1)

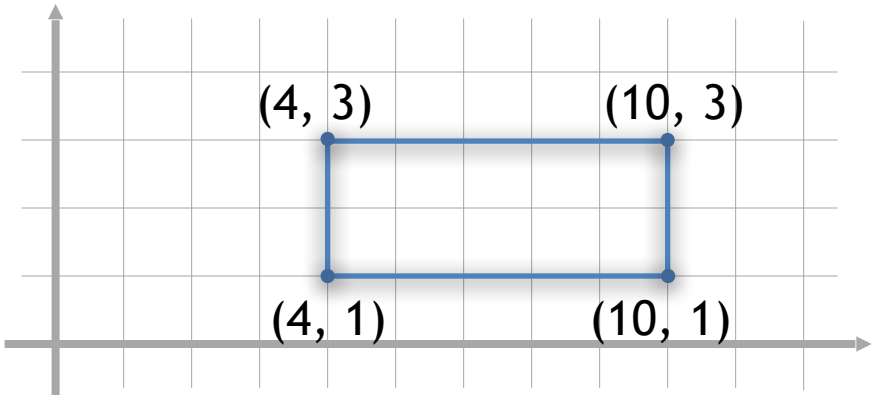
- We can use arrays inside structures. For example, let's assign values to *rect* using the given rectangle:

```
struct rectangle {
   struct point vertex[4];
};
struct rectangle rect;
```



Arrays Inside Structures (2)

```
struct rectangle rect;

rect.vertex[0].x = 4;
rect.vertex[0].y = 1;

rect.vertex[1].x = 4;
rect.vertex[1].y = 3;

rect.vertex[2].x = 10;
rect.vertex[2].y = 3;

rect.vertex[3].x = 10;
rect.vertex[3].y = 1;
```



Function pointers (1)

- Function pointer - a variable that holds the memory address of the callable object

Function pointers (2)

- Declaring

Use brackets () to declare function pointer:

`void (*foo)(int)` - function pointer which points to the function that takes int as parameter and returns void

`void *foo(int)` - function which returns void pointer

Function pointers (3)

- Initializing

```
void func(int x);

int main() {
void (*foo)(int);
foo = &func; // prefixing function name
with an ampersand
foo = func; // or simply by naming it
}
```

Function pointers (4)

- Invoking
  - Just as if you are calling a function:

```
foo(arg1, arg2);
```

  - Optionally dereference the function pointer before calling the function it points to:

```
(*foo)(arg1, arg2);
```

Function pointers (5)

- Function pointers can be passed to another function as an argument - callback:

```
void download_file(const char *file, void
(*callback_function)(int status_code));
```

Increasing flexibility of the functions and libraries!

Function pointers (6)

- qsort Example:

```
void qsort (void *base, size_t nmemb,
size_t size, int(*compar)(const void *,
const void *));
```

Function pointer allows anyone to specify how to sort the elements of the array without writing particular sorting algorithm.

Exercise: write your own **`sorter`** function and test it with **`qsort`**:

```
qsort (arg1, arg2, arg3, sorter);
```

Function pointers (7)

- Event programming

Function pointers provide a way of passing instructions on how to do smth when event happens:

```
struct Button;
void button_click(Button *button,
void (*on_click)(Button *button));
```

function sets up a callback that is called when a click is detected on a button

End

Week 03 – Lecture

References

- Tanenbaum & Bo, Modern Operating Systems: 4th edition, 2013 Prentice-Hall, Inc.
- http://www.cse.ust.hk/~liao/comp102/