DASF004
Basic and Practice in Programming

Lecture 12

Introduction to Data Structure

COVID-19 Situation …

- Vaccination is the key for current situation
- About 70% immunity in order to achieve herd immunity
    - UK: about 45%
    - US: about 40%
    - Israel: about 60%
- Two shots per person
    - 28 days separation
    - 14 days after the 2nd shot to achieve max immunity
- Paradigm shift in human behavior

Administration

- Final exam
  - The Final Assignment will be similar to the Midterm Assignment
  - It will be available on icampus
    - **Tuesday** 1 June 2021, 12:00 pm (noon)
    - Due date: **Wednesday** 2 June 2021 18:00 pm

Food for your MIND

- On Research Creativity
- What is Academic Research?
  - Ask new questions
  - Find out the answer for the question you asked
    - systematic investigation
    - in order to establish new facts and reach new conclusions

- Two major components to judge if a research topic is good
  - Research Novelty – What is new?
  - Research Importance – Why is it important?

- Start with an intense research about "what is done" in your research topic

Agenda

Introduction to Data Structure

- struct
- enum

Data Structures

Ways to organize information in order to enable efficient computation

- e.g. Represent a students score in an array
  - StudentA's score in course1 is 89; course2 is 92; course3 is 91.
  - `int StudentA[3] = {89, 92, 91};`
  - Course029 has 80 students, each student has a mid-term and a final score
  - `int Course029[80][2];`

How do you represent data structure with mixed types information

» Name, Student ID, Scores and Attendance

» String, String, Integers, Binary(True/False)

How do you represent data structure that is more complex or abstract

» Graph (e.g. the Subway Map)

» Priority (e.g. Queue)

» Procedure (e.g. Flowchart)

<u>Can I create my own Abstract Data Type?</u>

Picking the best Data Structure for the job

When you design your data structure:

- The data structure you pick needs to *support* the operations you need
- i.e. You need to understand what are the operations you will be performing

Ideally it supports the operations you will use most often in an *efficient* manner

Examples of operations:

Student score: average, max, min, count, histogram

Priority Queue: Insert, dispatch, reschedule, is_empty

Graph: shortest path, reachability

Some Common Data Structures

- List
  - Linked List
  - Sorted List
  - Double Linked List
  - Circular Linked List
- Stack (or Queue)
- Tree
  - Binary Tree
  - B-Tree
- Graph
  - Directed Graph
  - Weighted Graph

Question

How do you implement a data structure in C?

`struct`

- `struct` hold data in a group together
- Collection of related variables together
- `struct` are commonly used to define *records* to be stored in files (last lecture, File Processing).
- Pointers and structures facilitate the formation of more complex data structures
- Building block for database applications
  - Entry for storing information

struct basics

Definition of a structure:

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

Each identifier defines a member of the structure.

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

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

Example:

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

Date Birthday;
Birthday.day = 25;
Birthday.month = 12;
Birthday.year = 1980;
```

struct examples

Example:

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

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.

struct examples

Example:

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

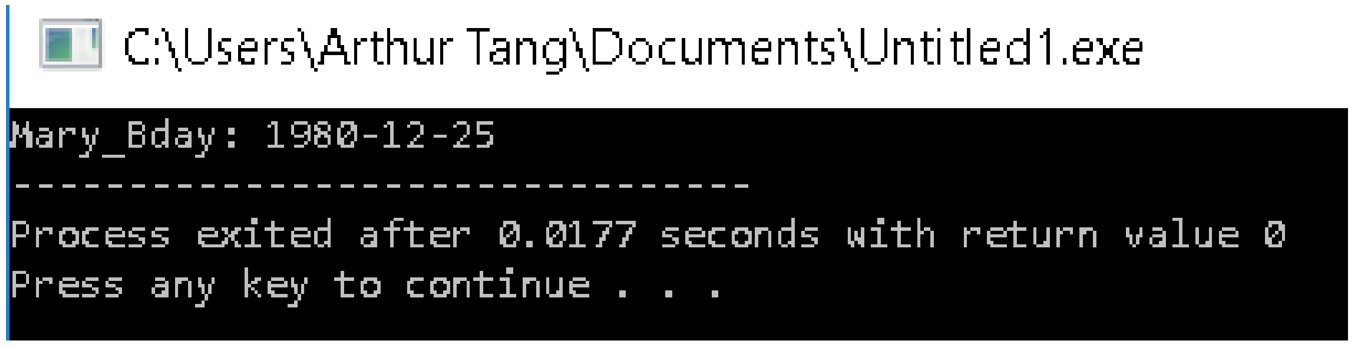
The “StudentRecord” structure has 4 members.

Assignment operator on `struct`

- Assignment operator = can be performed on `struct`

Example:

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

Date John_Bday, Mary_Bday;
John_Bday.day = 25;
John_Bday.month = 12;
John_Bday.year = 1980;
Mary_Bday = John_Bday;
printf("Mary_Bday: %d-%d-%d",
        Mary_Bday.year,Mary_Bday.month,Mary_Bday.day);
```



```
C:\Users\Arthur Tang\Documents\Untitled1.exe
Mary_Bday: 1980-12-25
--------------------------------
Process exited after 0.0177 seconds with return value 0
Press any key to continue . . .
```

Assignment operator on `struct`

- Note: if variable x is a char[] (string), assignment cannot be performed using the = operator

Example:

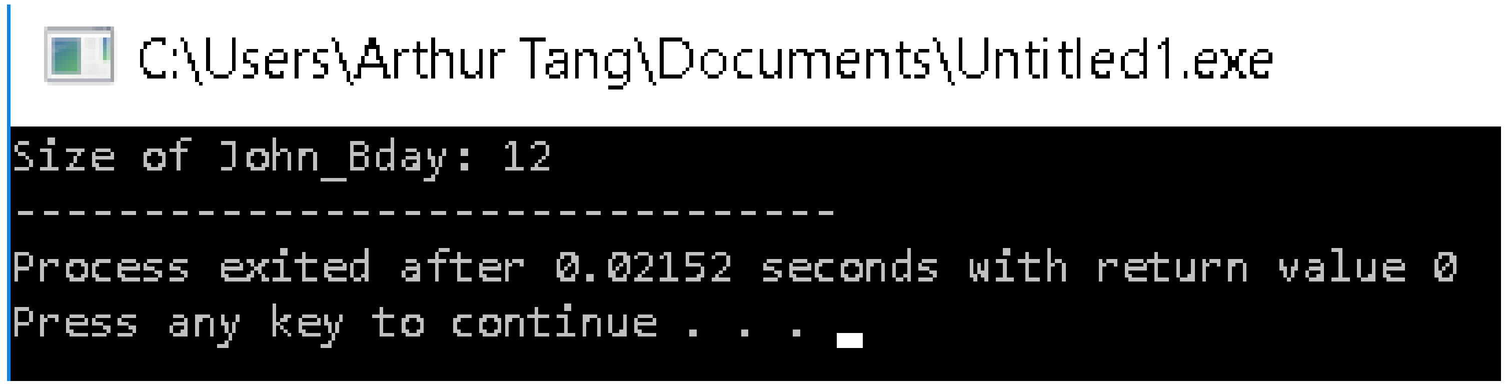
```
struct Student {
   char Name[15];
   int Id;
   char Gender;
} ;

Student StudentA;
strcpy(StudentA.Name,"John Kim");
StudentA.Id = 1357;
StudentA.Gender = 'M';
if(StudentA.Gender == 'M')
  printf("Mr. %s",StudentA.Name);
else if (StudentA.Gender == 'F')
  printf("Ms. %s",StudentA.Name);

Note: StudentA.Name = "John Kim" is invalid!
```

Using `sizeof()` function on `struct`

- You can use the `sizeof()` function to get the size of a `struct` variable

Example:

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

Date John_Bday;
John_Bday.day = 25;
John_Bday.month = 12;
John_Bday.year = 1980;

printf("Size of John_Bday: %d",sizeof(John_Bday));
```



C:\Users\Arthur Tang\Documents\Untitled1.exe

```
Size of John_Bday: 12
--------------------------------
Process exited after 0.02152 seconds with return value 0
Press any key to continue . . .
```

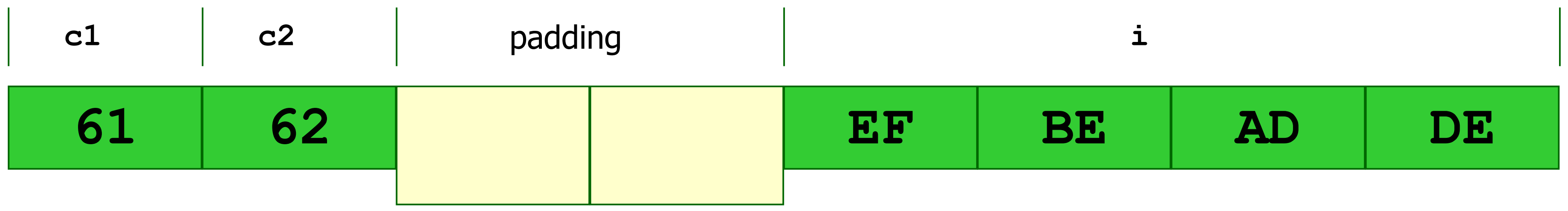
Using `sizeof()` function on `struct`

The size of a struct variable is not always equal to the sum of the size of its members.

For Example:
`sizeof(foo)` is 8 byte.

`sizeof(struct …)` =
sum of `sizeof(field)`
+ alignment padding
Processor- and compiler-specific

```
struct CharCharInt {
   char  c1;
   char  c2;
   int   i;
} foo;

foo.c1 = 'a';
foo.c2 = 'b';
foo.i  = 0xDEADBEEF;
```



x86 uses "little-endian" representation

Array of `struct`

- You can declare an array of a `struct` variable

Example:

```
struct Student {
   int StuID;
   int Exam[2];
} ;

Student GEDB029_50[80];
GEDB029_50[0].StuID = 0001;
GEDB029_50[0].Exam[0] = 92;
GEDB029_50[0].Exam[1] = 84;
```

… … … …

Pointer of `struct`

- You can declare pointer variable of a `struct` variable

Example:

```
struct Student {
   int StuID;
   int Exam[2];
} ;

Student GEDB029_50[80];
GEDB029_50[0].StuID = 0001;
GEDB029_50[0].Exam[0] = 92;
GEDB029_50[0].Exam[1] = 84;
GEDB029_50[1].StuID = 0002;
GEDB029_50[1].Exam[0] = 86;
GEDB029_50[1].Exam[1] = 74;
… … … …
Student * CurrentStudentPtr = &GEDB029_50[1];
printf("%d %d %d", (*CurrentStudentPtr).StuID,
(*CurrentStudentPtr).Exam[0], (*CurrentStudentPtr).Exam[1]);
```

Pointer of `struct`

- Note: . Has a higher precedence than *, therefore, () is required to enforce the * operator is calculated first.

Example:

```
struct Student {
   int StuID;
   int Exam[2];
} ;

Student GEDB029_50[80];
GEDB029_50[0].StuID = 0001;
GEDB029_50[0].Exam[0] = 92;
GEDB029_50[0].Exam[1] = 84;
GEDB029_50[1].StuID = 0002;
GEDB029_50[1].Exam[0] = 86;
GEDB029_50[1].Exam[1] = 74;
… … … …
Student * CurrentStudentPtr = &GEDB029_50[1];
printf("%d %d %d", (*CurrentStudentPtr).StuID,
 (*CurrentStudentPtr).Exam[0], (*CurrentStudentPtr).Exam[1]);
```

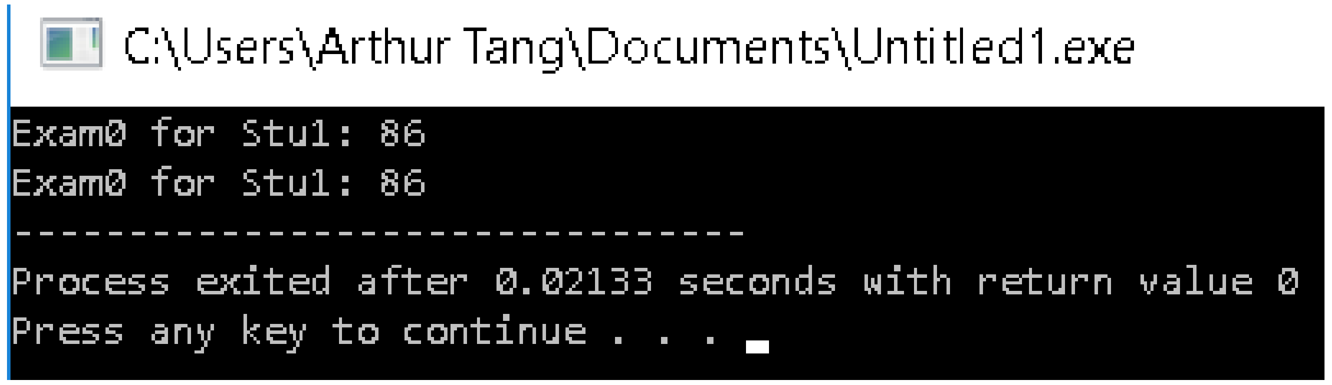
`struct` Pointer Operator `->`

The structure pointer operator (`->`) —accesses a structure member via a pointer to the structure.

`VarPtr->member` is a shorthand for `(*VarPtr).member`

Example:

```
struct Student {
   int StuID;
   int Exam[2];
} ;

Student GEDB029_50[80];
GEDB029_50[0].StuID = 0001;
GEDB029_50[0].Exam[0] = 92;
GEDB029_50[0].Exam[1] = 84;
GEDB029_50[1].StuID = 0002;
GEDB029_50[1].Exam[0] = 86;
GEDB029_50[1].Exam[1] = 74;
…  …  …  …
Student * CurrentStudentPtr = &GEDB029_50[1];
printf("Exam0 for Stu1: %d", (*CurrentStudentPtr).Exam[0]);
printf("Exam0 for Stu1: %d", CurrentStudentPtr->Exam[0]);
```



C:\Users\Arthur Tang\Documents\Untitled1.exe

```
Exam0 for Stu1: 86
Exam0 for Stu1: 86
--------------------------------
Process exited after 0.02133 seconds with return value 0
Press any key to continue . . .
```

```c
// Fig. 10.2: fig10_02.c
// Structure member operator and
// structure pointer operator
#include <stdio.h>

// card structure definition
struct card {
   char *face; // define pointer face
   char *suit; // define pointer suit
}; // end structure card

int main( void )
{
   struct card aCard; // define one struct card variable
   struct card *cardPtr; // define a pointer to a struct card

   // place strings into aCard
   aCard.face = "Ace";
   aCard.suit = "Spades";

   cardPtr = &aCard; // assign address of aCard to cardPtr

   printf( "%s%s%s\n%s%s%s\n%s%s%s\n", aCard.face, " of ", aCard.suit,
      cardPtr->face, " of ", cardPtr->suit,
      ( *cardPtr ).face, " of ", ( *cardPtr ).suit );
} // end main
```

```
Ace of Spades
Ace of Spades
Ace of Spades
```

struct examples: Nested Structure

Example (nested structure):

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

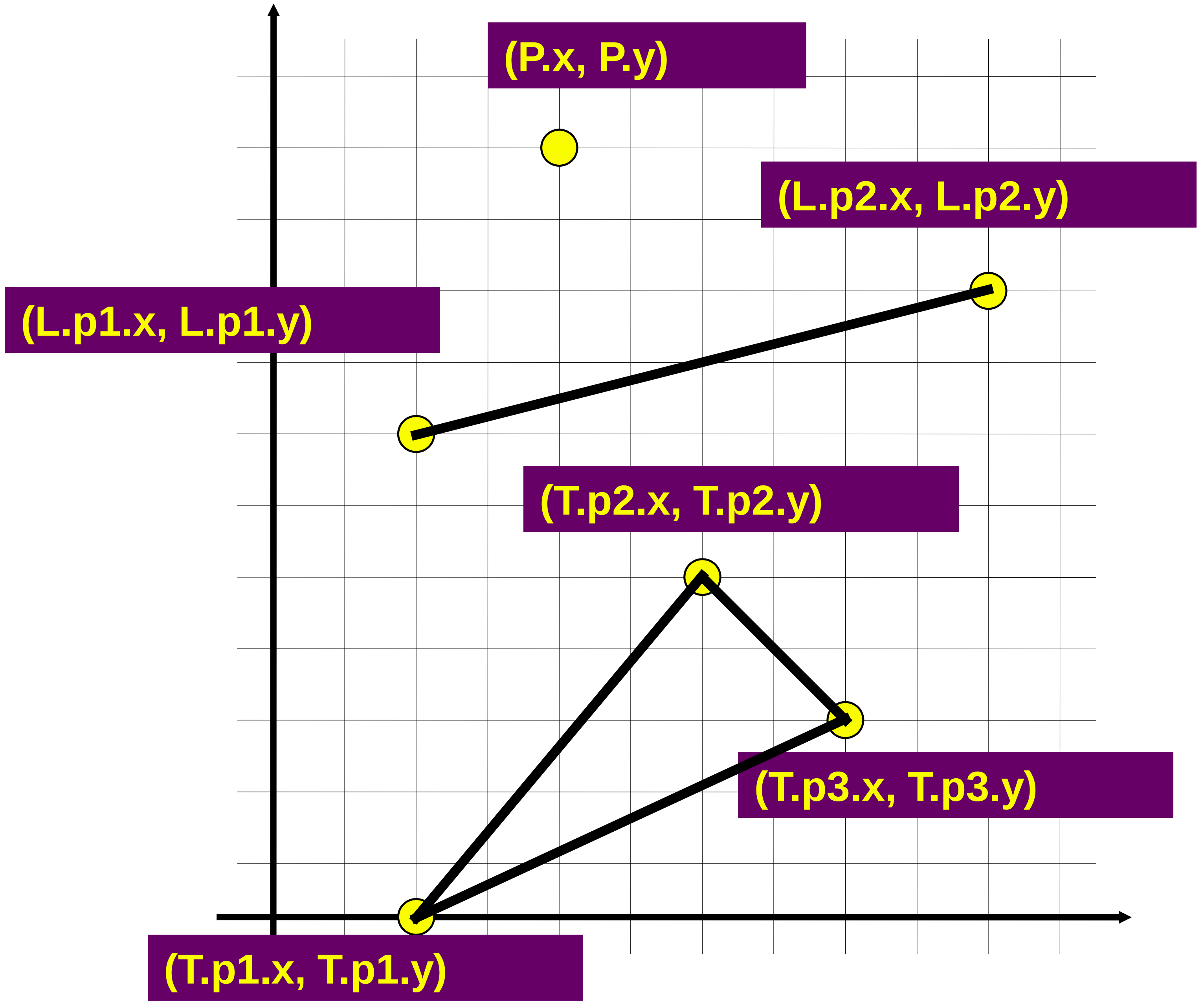
The “BankAcount” structure has simple, array and structure types as members.

Ex. 3-5: Nested structures

We can nest structures inside structures.

Examples:

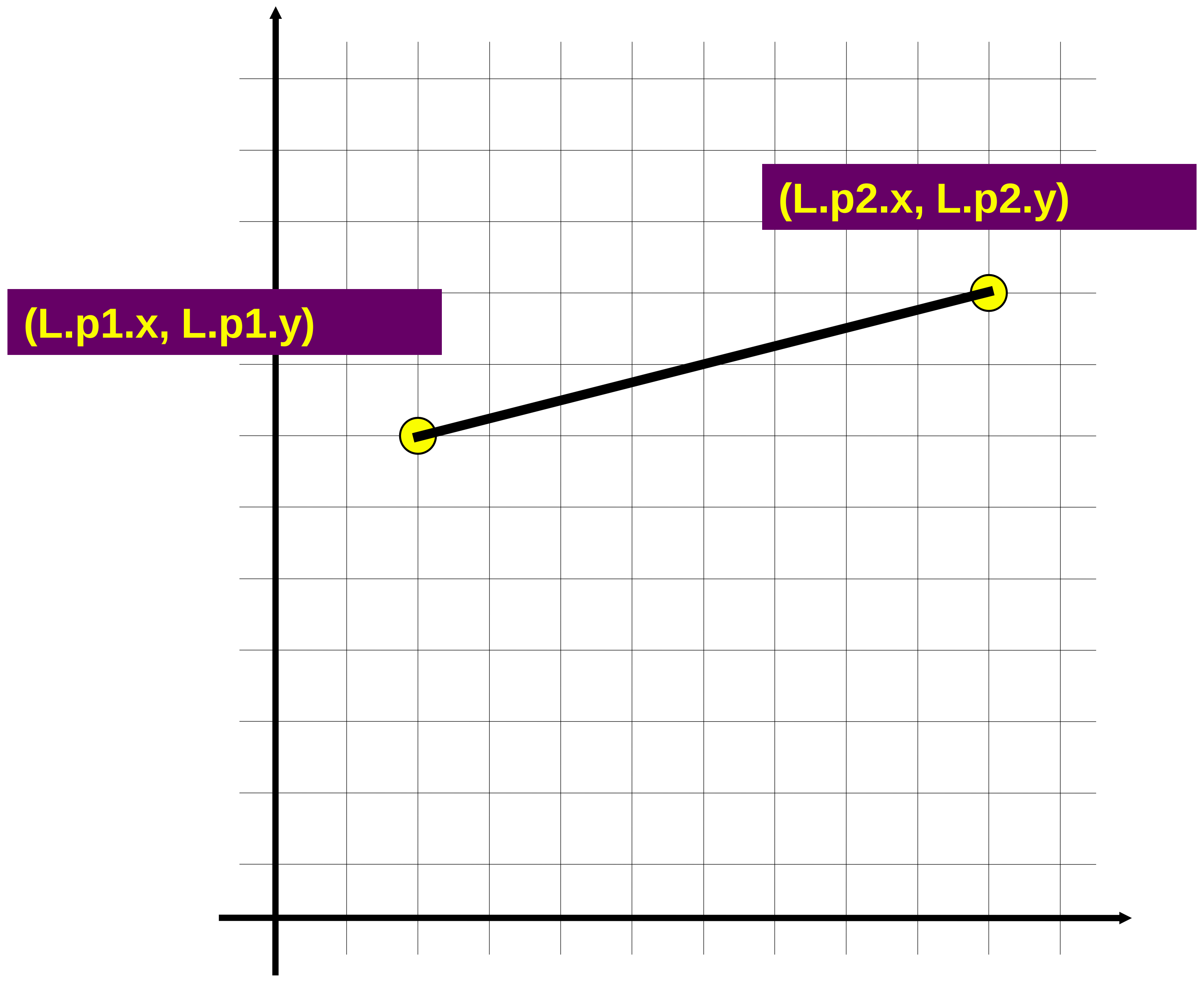
```
 struct point{
double x, y;
};
point P;

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

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



Ex. 3-5: Nested structures

We can nest structures inside structures.

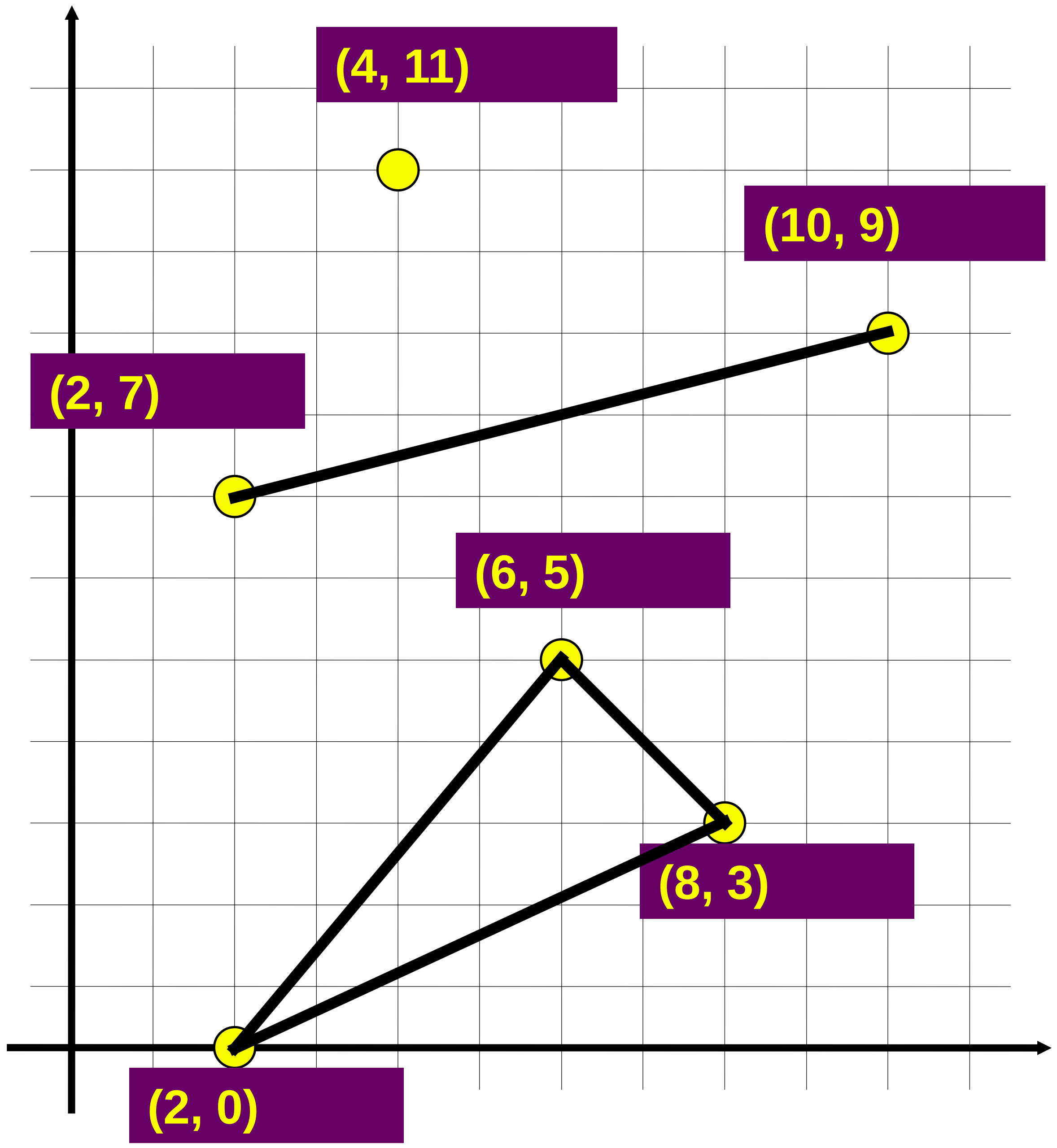
```
struct line{
point p1, p2;
};
line L;
```



Ex. 3–5: Nested structures

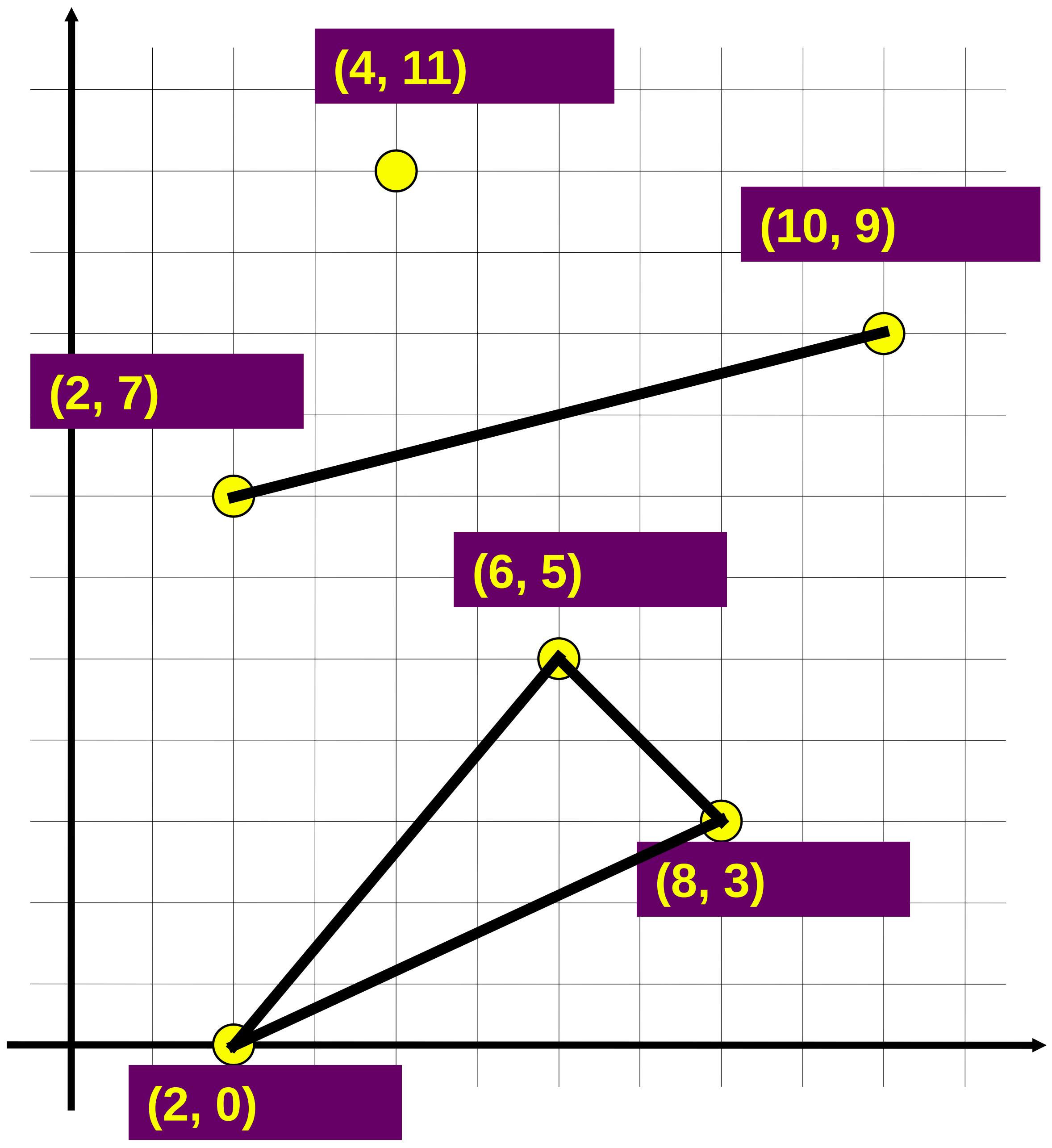
Assign values to the variables **P**, **L**, and **T** using the picture:

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



Ex. 3-5: Nested structures

```
point P;
line L;
triangle T;
P.x = 4;
P.y = 11;
L.p1.x = 2;
L.p1.y = 7;
L.p2.x = 10;
L.p2.y = 9;
T.p1.x = 2;
T.p1.y = 0;
T.p2.x = 6;
T.p2.y = 5;
T.p3.x = 8;
T.p3.y = 3;
```



struct examples: Nested Structure

- *A structure **cannot** contain an instance of itself.*
- For example, a variable of type `struct employee` cannot be declared in the definition for `struct employee`.
- A **pointer** to `struct employee`, however, may be included.

(This is useful!!! We’ll talk about it.)

- For example,

  -
    ```
    struct employee2 {
       char firstName[ 20 ];
       char lastName[ 20 ];
       unsigned int age;
       char gender;
       double hourlySalary;
       employee2 person; // This is ERROR!!!
       employee2 *ePtr; // This is valid
    }; // end struct employee2
    ```
- `struct employee2` contains an instance of itself (`person`), which is an error.
- ePtr is a pointer to type struct employee2, it is permitted in the definition.

10.2 Structure Definitions (Cont.)

- Because `ePtr` is a pointer (to type `struct employee2`), it’s permitted in the definition.
- A structure containing a member that’s a pointer to the *same* structure type is referred to as a self-referential structure.

10.2.4 Operations That Can Be Performed on Structures

The only valid operations that may be performed on structures are:

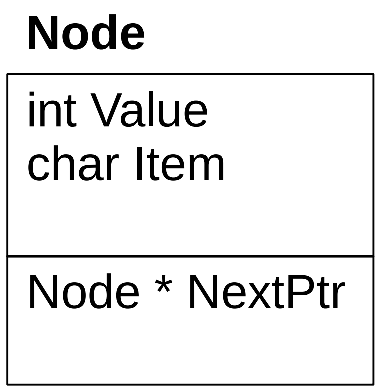
1. “`=`”: assigning structure variables to structure variables of the *same* type,
2. “`&`”: taking the address (&) of a structure variable,
3. “`.`” and “`->`”: accessing the members of a structure variable (see Section 10.4) and
4. `sizeof()`: using the `sizeof` operator to determine the size of a structure variable.

Application of `struct` Pointer

You can use the structure pointer to implement various generic data structure.

Example:

```
struct Node {
   int Value;
   char Item[10];
   Node * NextPtr;
} ;
```

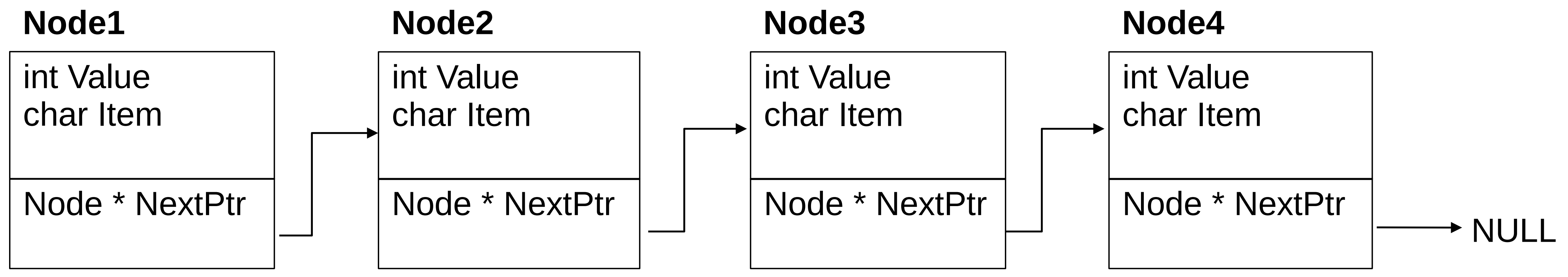


Application of `struct` Pointer

Then you can create a chain of data using the pointer.

Example:

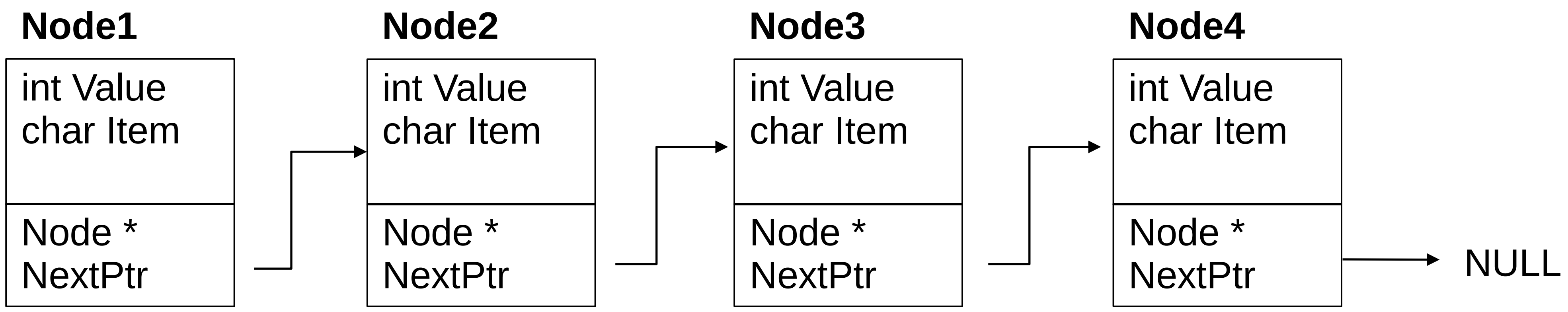
```
struct Node {
   int Value;
   char Item[10];
   Node * NextPtr;
} ;
```



Data Structure: Linked List

Similar to array, but

- Can store complex data
- Dynamic: Can grow and shrink in size
  - Don’t need to know how large the array is going to be
- Insertion and deletion is easy and fast
  - No need to move other data when adding or removing data in the middle



Application of `struct` Pointer

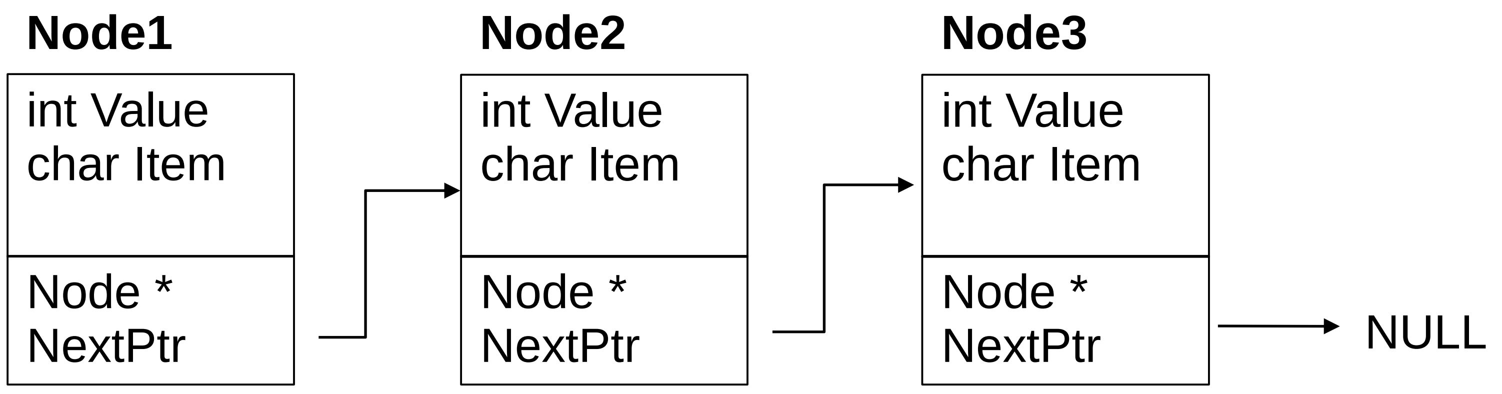
And you can use this to implement various data structures:

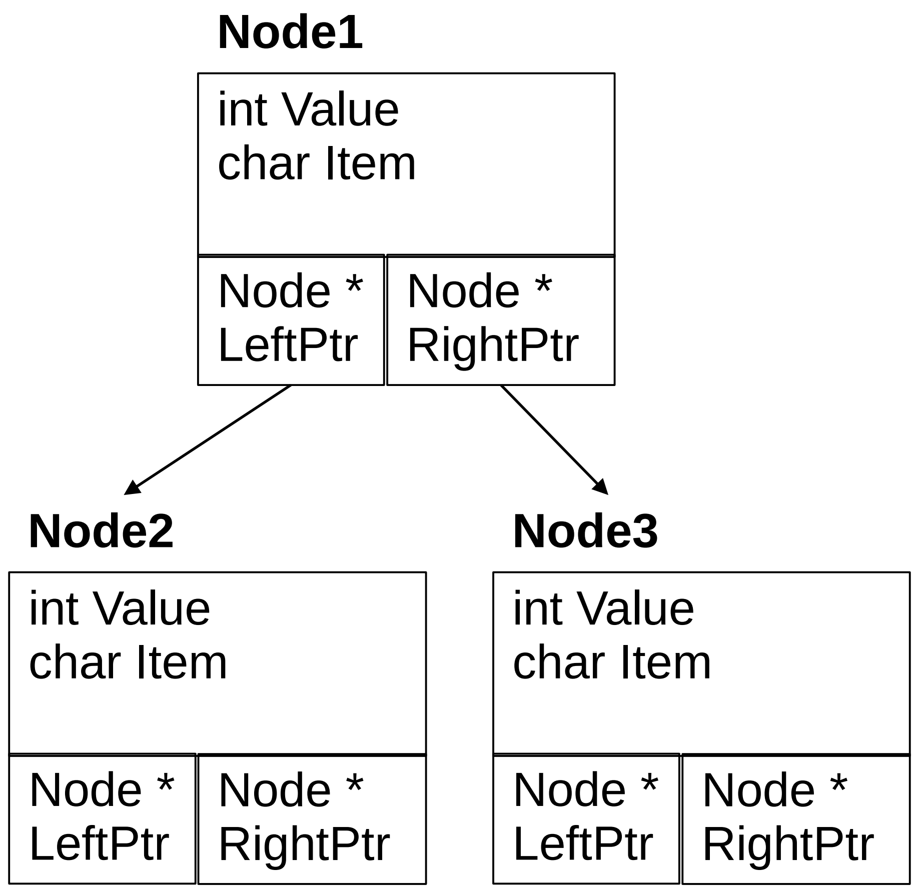
List
- Linked List
- Double Linked List
- Circular Linked List

Stack (or Queue)

Tree
- Binary Tree
- B-Tree

Graph
- Directed Graph
- Weighted Graph





10.11 Enumeration Constants

An enumeration (discussed briefly in Section 5.11), introduced by the keyword `enum`, is a set of integer enumeration constants represented by identifiers.

Values in an `enum` start with `0`, unless specified otherwise, and are incremented by `1`.

For example, the enumeration

```
• enum months {
     JAN, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP,
     OCT, NOV, DEC }; // end enum months
```

creates a new type, `enum months`, in which the identifiers are set to the integers `0` to `11`, respectively.

10.11 Enumeration Constants (Cont.)

To number the months 1 to 12, use the following enumeration:

-
  ```
  enum months {
     JAN = 1, FEB, MAR, APR, MAY, JUN, JUL, AUG,
     SEP, OCT, NOV, DEC }; // end enum months
  ```

Because the first value in the preceding enumeration is explicitly set to `1`, the remaining values are incremented from `1`, resulting in the values `1` through `12`.

The identifiers in an enumeration must be unique.

The value of each enumeration constant of an enumeration can be set explicitly in the definition by assigning a value to the identifier.

10.11 Enumeration Constants (Cont.)

Multiple members of an enumeration can have the same constant value.

In the program of Fig. 10.18, the enumeration variable `month` is used in a `for` statement to print the months of the year from the array `monthName`.

We've made `monthName[0]` the empty string `""`.

You could set `monthName[0]` to a value such as `***ERROR***` to indicate that a logic error occurred.

```
// Fig. 10.18: fig10_18.c
// Using an enumeration
#include <stdio.h>

// enumeration constants represent months of the year
enum months {
   JAN = 1, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, DEC
}; // end enum months

int main( void )
{
   enum months month; // can contain any of the 12 months

   // initialize array of pointers
   const char *monthName[] = { "", "January", "February", "March",
      "April", "May", "June", "July", "August", "September", "October",
      "November", "December" };

   // loop through months
   for ( month = JAN; month <= DEC; ++month ) {
      printf( "%2d%11s\n", month, monthName[ month ] );
   } // end for
} // end main
```

Fig. 10.18 | Using an enumeration. (Part 1 of 2.)

```
 1    January
 2   February
 3      March
 4      April
 5        May
 6       June
 7       July
 8     August
 9  September
10    October
11   November
12   December
```

Fig. 10.18 | Using an enumeration. (Part 2 of 2.)

Q&A?

Last, but not least

Check that you have submit your attendance.

If you don’t have your phone or computer and could not submit your attendance, you should talk to me NOW!

Attendance cannot be made up.