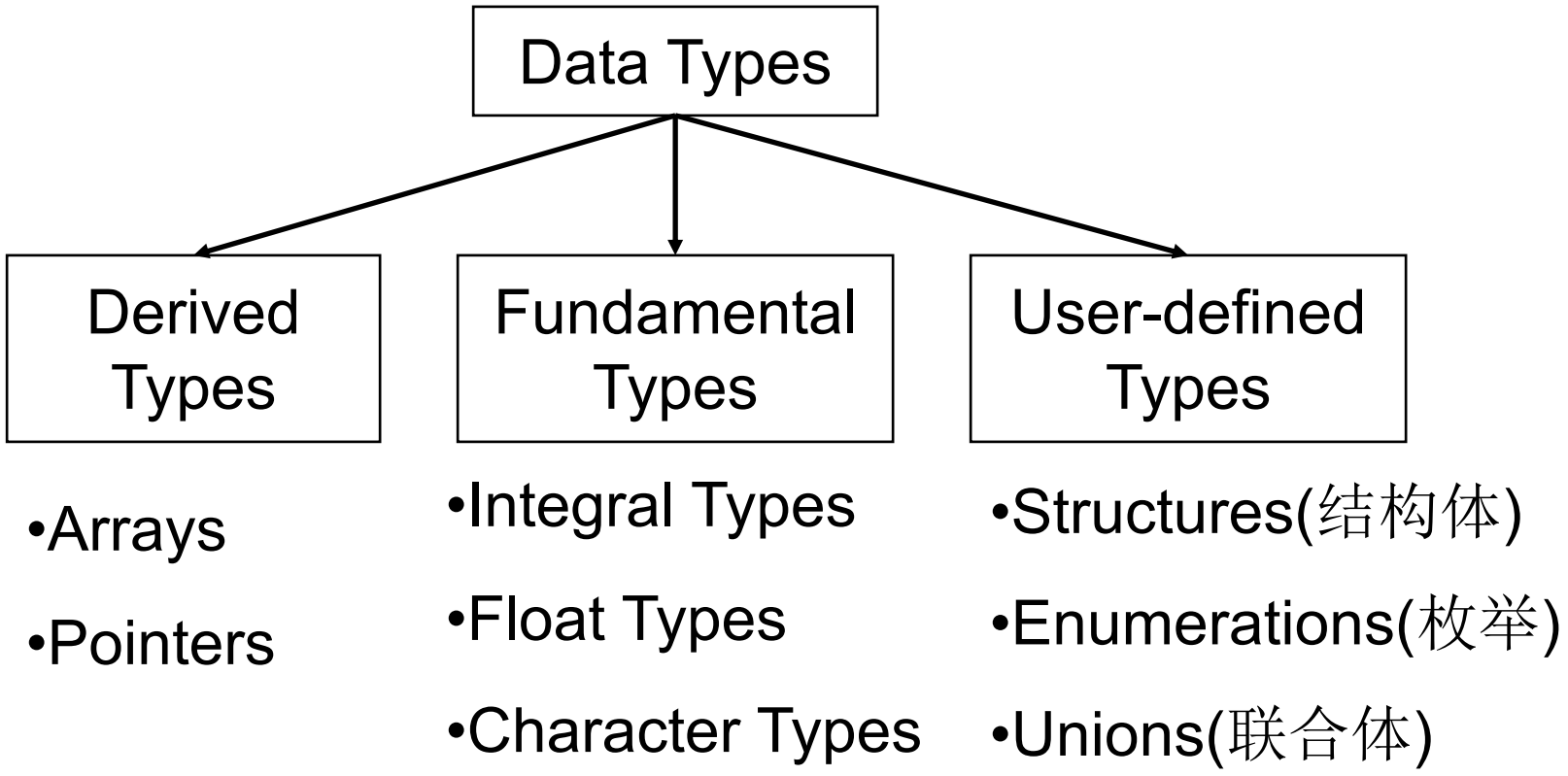
Class 11 Structure(结构体)

- Program = Algorithm(算法) + Data structure(数据结构)

  - Subsequent, selection and loop blocks of statements.
  - Function: a group of statements achieves a specific task.

- Data structure



- Data structures enable us to access and manipulate data in a relatively easy manner, and thus generate efficient programs.

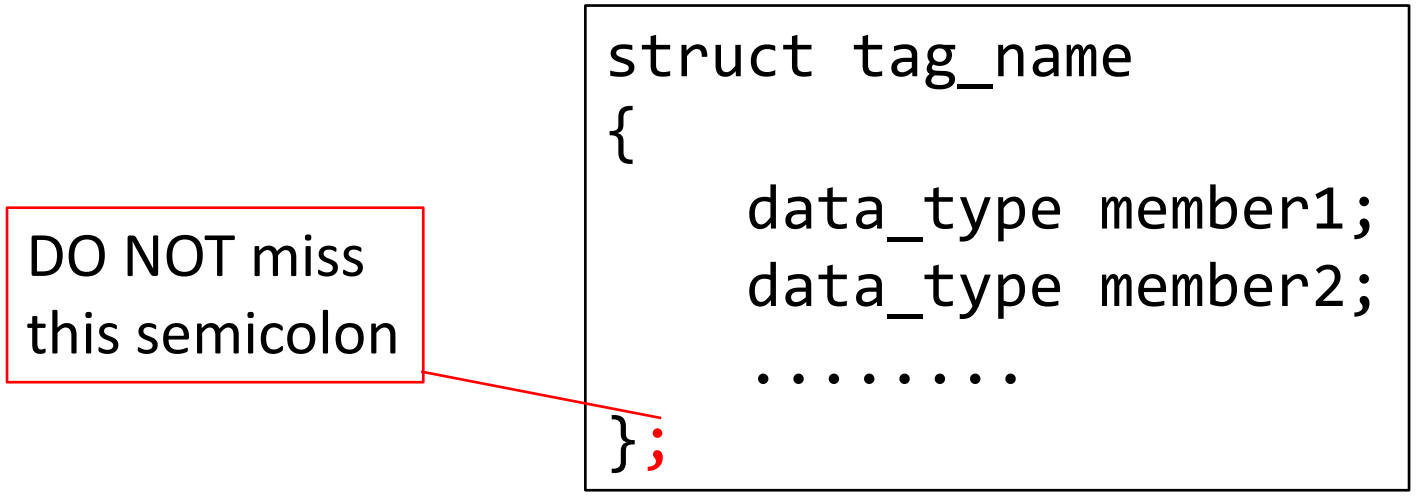
- An array is a fixed-size sequenced collection of elements of the same data type.

  Examples: float height[50];
  int number[5];
  double scores[STU_NUM]; /*#define STU_NUM 60*/

- A structure: a convenient tool for organizing a group of logically related data items with different types.

time	:	seconds, minutes, hours
date	:	day, month, year
book	:	author, title, price, year
address	:	name, door-number, street
point	:	x, y coordinates
rectangle	:	coordinates of vertices

Structure Declaration (结构体声明)



- Not like derived types, the formats or members of a structure have to be declared first and then structure variables can defined later.
- The *tag_name* is not a variable
- No memory is allocated in structure declaration

Example of Structure Declaration

```
struct book
{
    char title[30];
    char author[15];
    int pages;
    float price;
};
```

- It declares a structure to hold the details of four data fields, namely title, author, page, price. These fields are called structure elements or members.
- Structure members may belong to a different type of data.

title	array of 30 characters
author	array of 15 characters
pages	integer
price	float

Define a Structure Variable

```
struct tag_name variable_name;
```

- Memory is allocated when a structure variable is defined

- Declare a structure before a structure variable is defined

Example:

```
//declaration of the structure book
struct book
{
    char title[30];
    char author[15];
    int pages;
    float price;
};

//definition of two variables
struct book b1, b2;
```

DO NOT miss the tag_name

Combing Declaration and Definition

- Declaration and Definition with a *tag_name*

```
//structure declaration and
//structure definition are
//done at the same time
struct book
{
    char title[30];
    char author[15];
    int pages;
    float price;
}b1, b2;

//other variables can be
//defined by using tag_name
struct book b3, b4;
```

- Declaration and Definition without a *tag_name*

```
//structure declaration and
//structure definition are
//done at the same time,
//but without a tag_name
struct
{
    char title[30];
    char author[15];
    int pages;
    float price;
}b1, b2;

//Without a tag_name
//No other variables
//can be defined
```

Using the keyword of typedef

- Using typedef to declare a type name which can be used as any other fundamental and derived types.
  - define a structure variable without the keyword struct
  - tag_name can be omitted

```
typedef struct
{
    data_type member1;
    data_type member2;
    ........
}type_name;

type_name variable_name1;
type_name variable_name2;
```

Structure Initialization

- Structure variables can be initialized in the definition

```
struct book
{
    char title[30];
    char author[15];
    int pages;
    float price;
}b1 = {“C Language”, “John Smith”, 345, 56.0f};

struct book b2 =
    {“C++”, “Alen Green”, 421, 73.5f};
```

Accessing Members of a Structure

```
variable_name.member_name
```

- By using the . operator, values of each member can be accessed.
  - reading values of members
  - changing values of members

```
struct book
{
    char *title; //why we can not use a character array?
    char *author;//why we can not use a character array?
    int pages;
    float price;
}b1 = {“C Language”, “John Smith”, 345, 56.0f};
printf(“%s %s %d %f\n”, b1.title, b1.author, b1.pages, b1.price);

b1.title = “C++”;  b1.author = “Alen Green”;
b1.pages = 421;  b1.price = 73.5f;
printf(“%s %s %d %f\n”, b1.title, b1.author, b1.pages, b1.price);
```

Exercise

- Design a structure and use it to store all scores for each student, and print out the stored information.

	Math	C	English
Jack	80	61	59
Tom	75	65	63
Lucy	92	71	70

Hint : A structure includes four members that stores the student name, scores of the three courses

```c
#include <stdio.h>

int main()
{
	typedef struct
	{
		char *name;
		int math;
		int c;
		int english;
	}RecordType;

	RecordType r1 = {"Jack", 80, 61, 59};
	RecordType r2 = {"Tom", 75, 65, 63};
	RecordType r3 = {"Lucy", 92, 71, 70};

	printf("%s %d %d %d\n", r1.name, r1.math, r1.c, r1.english);
	printf("%s %d %d %d\n", r2.name, r2.math, r2.c, r2.english);
	printf("%s %d %d %d\n", r3.name, r3.math, r3.c, r3.english);

	return 0;
}
```

Operation on Structure Variables

- Assignment operator can be applied on two variables of the same structure type
  - Bitwise copy is operated in assignment of two variables of structures

```
struct book b1 = {“C Language”, “John Smith”, 345, 56.0f};
struct book b2 = b1;
```

- Arithmetic and logical operators are not permitted.

```
//The following operations are not allowed
b1+b2;
b1<b2;
b1 == b2;
scanf("  ",&b1);
```

Structure Array

```
struct tag_name
{
    data_type member1;
    data_type member2;
    ........
};
struct tag_name
    array_name[size];
```

```
struct tag_name
{
    data_type member1;
    data_type member2;
    ........
}array_name[size];
```

- The type of Elements of a structure array is structure
- Each element of a structure array can be used as a structure variable

```c
#include <stdio.h>

int main()
{
        struct RecordType
        {
                char *name;
                int math;
                int c;
                int english;
        };

        int i;
        struct RecordType r[3] = { {"Jack", 80, 61, 59},
                                   {"Tom", 75, 65, 63},
                                   {"Lucy", 92, 71, 70} };

        for ( i = 0 ; i < 3 ; i++ )
              printf("%s %d %d %d\n", r[i].name, r[i].math,
                                      r[i].c, r[i].english);

        return 0;
}
```

Pointer to a Structure

- Pointer variable that points to a structure variable.

```
struct tag_name *pointer_name;
```

- The initial address of a structure variable can be assigned to the pointer to a structure

```
struct book b1;
struct book *p = &b1;
```

- The pointer works the same as any other pointers that point to variables of fundamental and derived types.

```
struct book b[5];
struct book *p = &b[0];
//print b[0]
printf("%s %d %f\n", (*p).title, (*p).pages, (*p).price);
//print b[1]
p++; printf("%s %d %f\n", (*p).title, (*p).pages, (*p).price);
```

Pointer to a Structure

- Access members of a structure variable though a pointer by using -> operator
  - reading the values of members
  - changing the values of members

```
pointer_name->member
```

- We can allocate the memory for a structure and assign the initial address to a pointer

```
struct book *p = NULL;
p = (struct book *)malloc(sizeof(struct book));
.......
free(p);
```

```c
#include <stdio.h>
#include <stdlib.h>
int main()
{
    struct RecordType{
        char *name;
        int math;
        int c;
        int english;
    };
    struct RecordType *p = (struct RecordType *)malloc(
                        3*sizeof(struct RecordType));
    struct RecordType *pTem;
    p->name="Jack"; p->math=80; p->c=61; p->english=59;
    (*(p+1)).name="Tom"; (*(p+1)).math=75;
    (*(p+1)).c=65; (*(p+1)).english=63;
    p[2].name="Lucy"; p[2].math=80; p[2].c=71; p[2].english=70;

    for ( pTem = p ; pTem < p+3 ; pTem++ )
            printf("%s %d %d %d\n", (*pTem).name, (*pTem).math,
                    pTem->c, pTem->english);
    free(p);
    return 0;
}
```

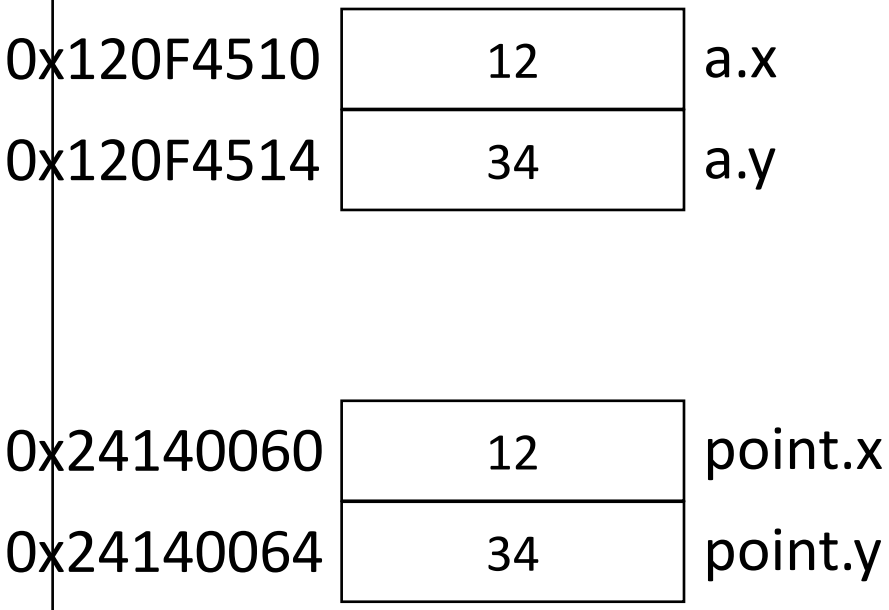
Structure & Function

- Structure Variables as Formal Parameters of a Function
  - We can read the values of structure members
  - No changes can be made for structure members inside the called function
  - Not efficient because all members are copied from the actual parameter to the formal parameter

- Structure Pointers as Formal Parameters of a Function
  - We can read and change the values of structure members inside the called function
  - Very efficient because no members are copied

Structure Variables as Formal Parameters

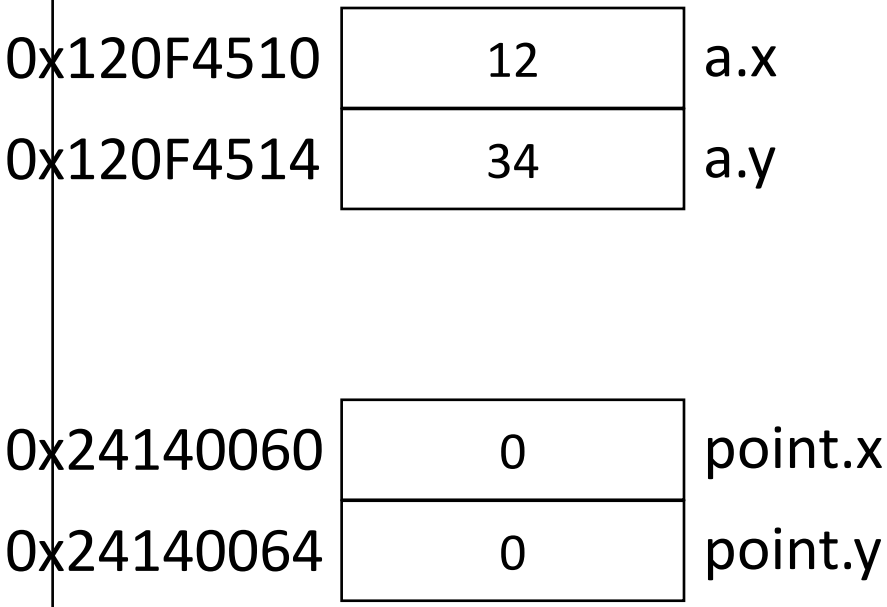
```
#include <stdio.h>

typedef struct
{
    int x;
    int y;
}PointType;

void setZero(PointType point)
{
    point.x = 0;
    point.y = 0;
}

int main()
{
    PointType a = {12, 34};
    printf("(%d, %d)\n", a.x, a.y);
    setZero(a);
    printf("(%d, %d)\n", a.x, a.y);
    return 0;
}
```



output:
(12, 34)
(12, 34)

Structure Variables as Formal Parameters

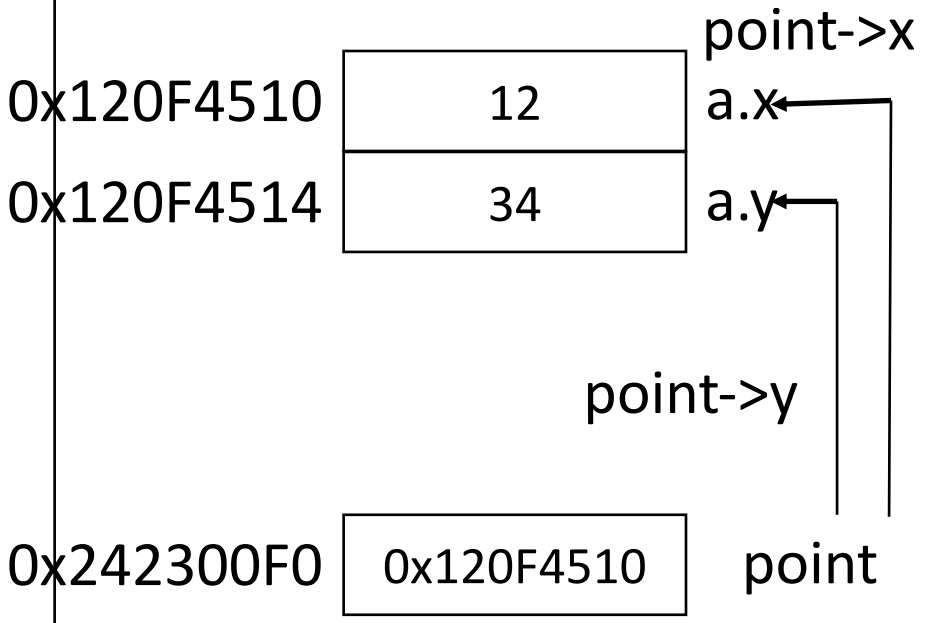
```
#include <stdio.h>

typedef struct
{
    int x;
    int y;
}PointType;

void setZero(PointType point)
{
    point.x = 0;
    point.y = 0;
}

int main()
{
    PointType a = {12, 34};
    printf("(%d, %d)\n", a.x, a.y);
    setZero(a);
    printf("(%d, %d)\n", a.x, a.y);
    return 0;
}
```



output:
(12, 34)
(12, 34)

Structure Variables as Formal Parameters

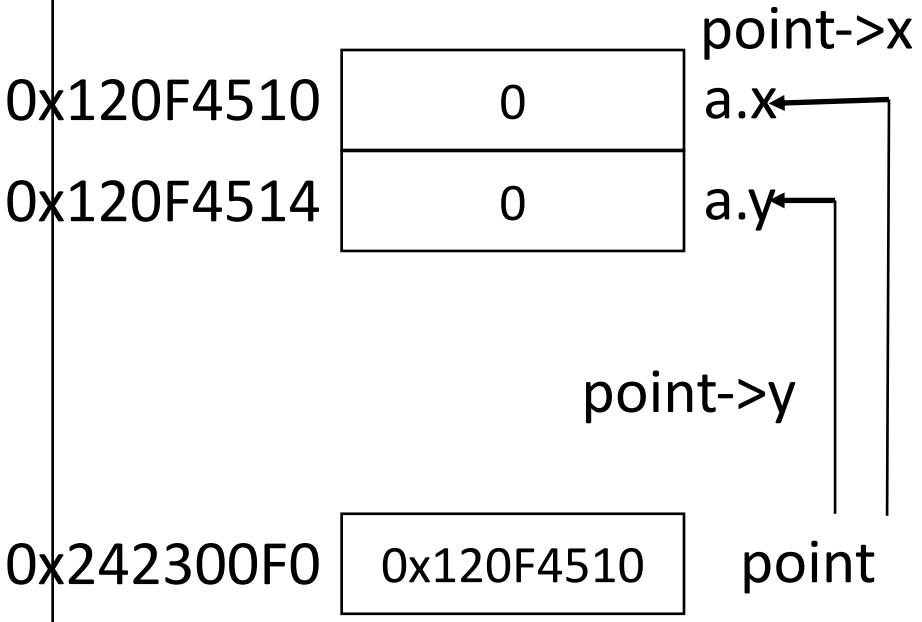
```
#include <stdio.h>

typedef struct
{
    int x;
    int y;
}PointType;

void setZero(PointType *point)
{
    point->x = 0;
    point->y = 0;
}

int main()
{
    PointType a = {12, 34};
    printf("(%d, %d)\n", a.x, a.y);
    setZero(&a);
    printf("(%d, %d)\n", a.x, a.y);
    return 0;
}
```



output: (12, 34)
(0, 0)

Structure Variables as Formal Parameters

```
#include <stdio.h>

typedef struct
{
    int x;
    int y;
}PointType;

void setZero(PointType *point)
{
    point->x = 0;
    point->y = 0;
}

int main()
{
    PointType a = {12, 34};
    printf("(%d, %d)\n", a.x, a.y);
    setZero(&a);
    printf("(%d, %d)\n", a.x, a.y);
    return 0;
}
```



output:
(12, 34)
(0, 0)

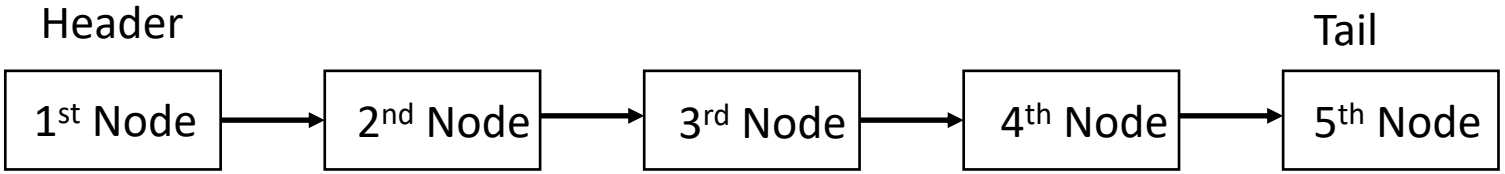
Introduction of List (链表)

- Array is a sequential data structure with a fixed size
  - Array size should be determined before the definition
  - If an array is defined, we can not change its size
  - It is difficult to add or delete an element from an array

- List is a sequential data structure either, but the size of a list is changeable.
  - We can change the size of a list any time.
  - It is easy to add or delete an element for a list.

Introduction of List (链表)

- Basic Concepts of a List
  - Header (头) : the 1st element of a list
  - Node (节点) : the element of a list
  - Tail (尾) : the last element of a list

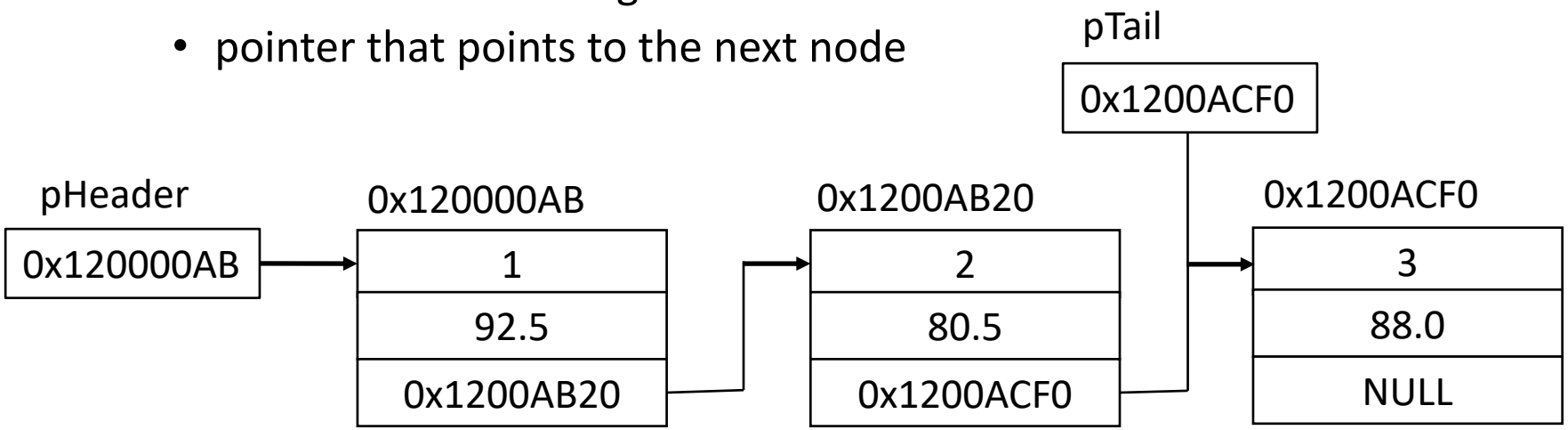


- List can be implemented by structure
  - Node : a structure defined by users
    - Each node should have a pointer that points to the next node
  - Header Pointer : a pointer that points to the 1st node
  - Tail Pointer: a pointer that points to the last node

Example of a List

- A list that stores the scores of all students
  - The elements of the Node
    - student ID : a integer number
    - score : a floating number
    - pointer that points to the next node

```
struct student
{
    int id;
    float score;
    struct student *pNext;
}
```



Manipulation of a List

- Create a List
- Traverse a List (遍历链表)
- Search a Node()
- Insert or Remove a Node
- Destroy a List

Create a List

Create the header and
tail pointer and set to NULL

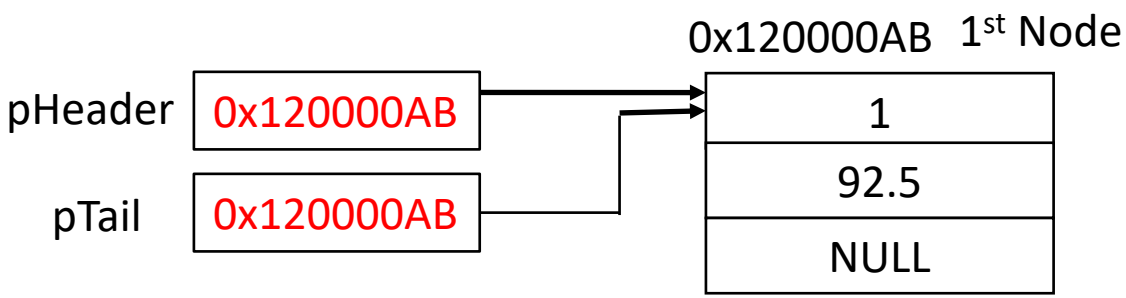
pHeader	NULL
pTail	NULL

Create a New Node

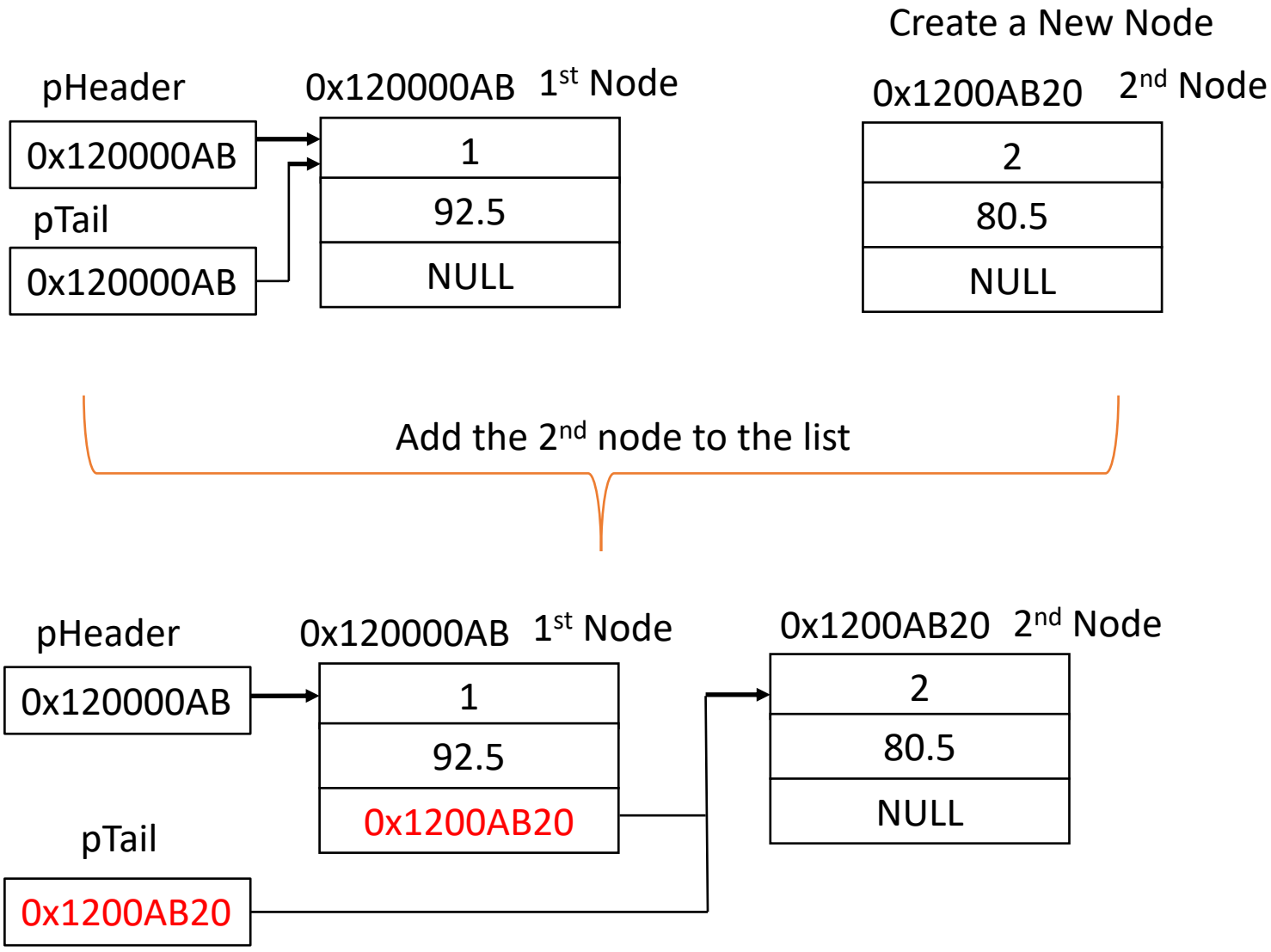
0x120000AB 1st Node

1
92.5
NULL

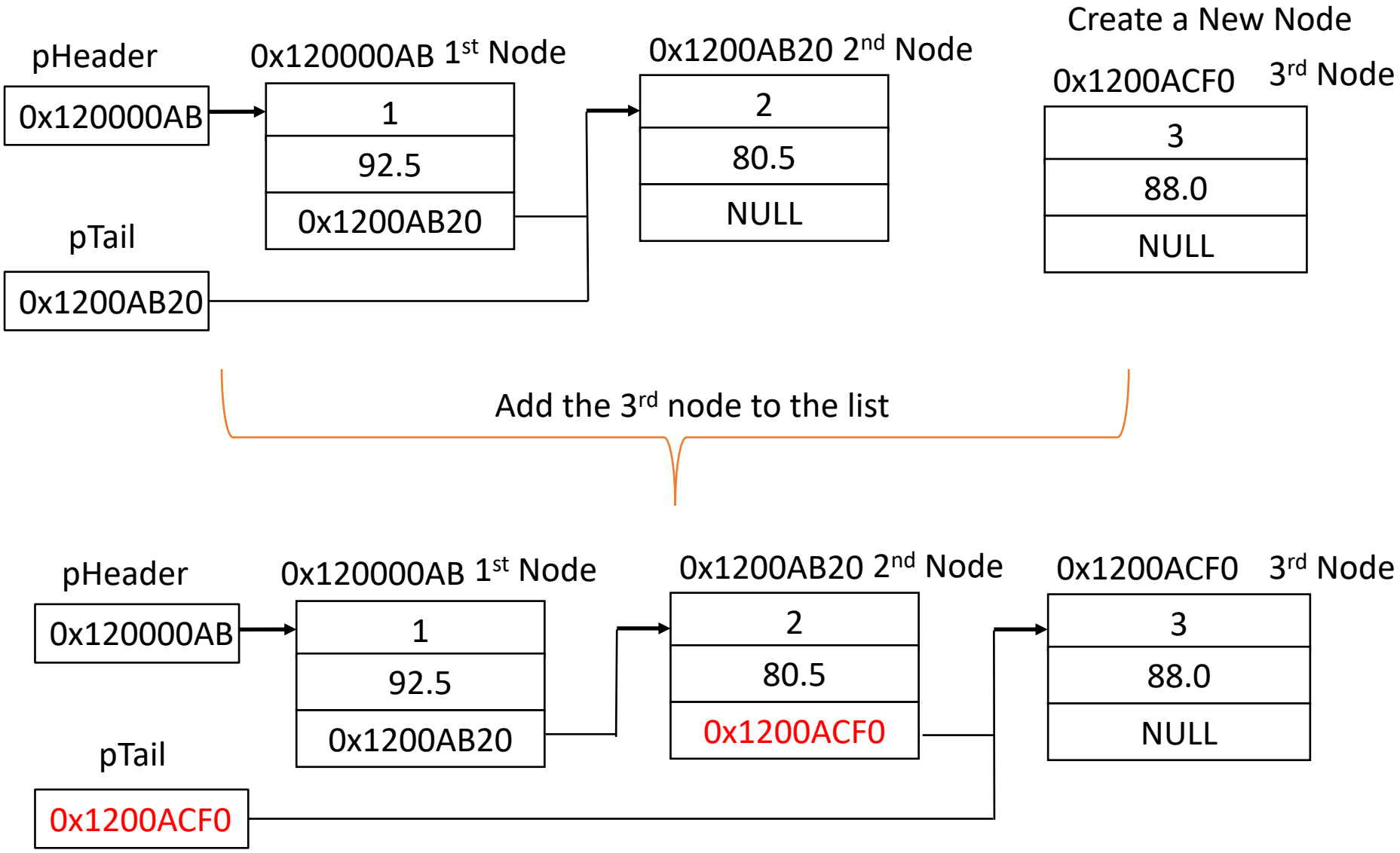
Add the 1st node to the list



Create a List



Create a List



Create a List

- The Procedure of Creating a List
  - Create a new node and insert it to the tail of the list
- Notes of Inserting a New Node
  - When the list is empty
    - Make the header pointer and the tail pointer point to the new node
  - When the list is not empty
    - Insert the new node at the tail, and change the tail.

```
struct student * createList( int num )
{
    struct student *pHeader = 0, *pNode, *pTail = 0;
    int i, id;
    float score;
    for ( i = 0 ; i < num ; i++ )
    {
       pNode = (struct student *)malloc(sizeof(struct student));
       scanf("%d %f", &id, &score);
       pNode->id = id; pNode->score = score;
       pNode->pNext = 0; //point to NULL

       if ( !pHeader )  //the list is empty
       {
            pHeader = pTail = pNode;
       }
       else  // the list is not empty
       {
            pTail->pNext = pNode; // add the new node to the list
            pTail = pNode;        // the new node becomes the tail
       }
    }
    return pHeader;
}
```

Traverse a List (遍历链表)

- Start from the header, and visit the 1st node (i.e. read values of all members), and then visit the 2nd node. This process is operated until the tail.
- We can traverse a list by using its header pointer
- Example : Print all information stored in a list

```
// the header pointer of a list is passed to this function
void printList( struct student *p )
{
    while ( p != NULL )
    {
        printf("id = %d, score = %f\n", p->id, p->score);
        p = p->pNext;
    }
}
```

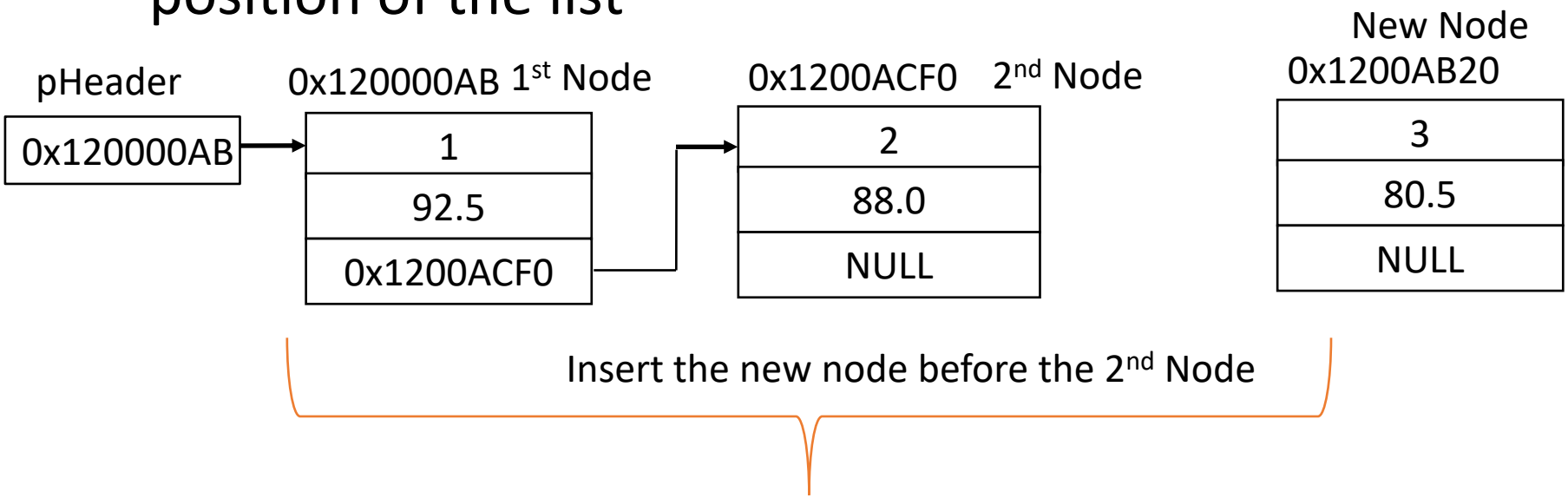
Search a Node

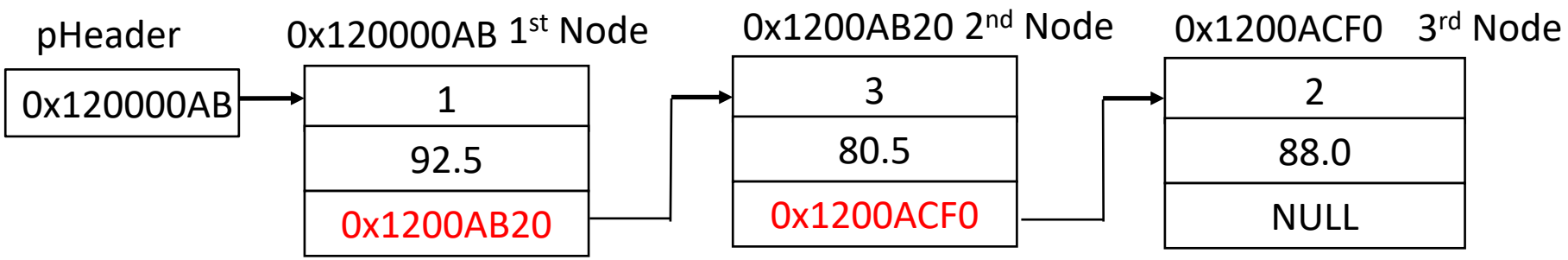
- Find a specific node and return its pointer
- Example : find a student who obtains a specific score

```
// p – header pointer of a list
// score – the specific score
// returned value – the pointer of the found node
//                  return NULL if not found
struct student * search( struct student *p, float score )
{
    struct student *pNode = NULL;
    while ( p != NULL )
    {
        if ( fabs(p->score – score) < 1e-6 ) // Why use fabs()?
        { pNode = p ; break; }
        p = p->pNext;
    }
    return pNode;
}
```

Insert a New Node to a List

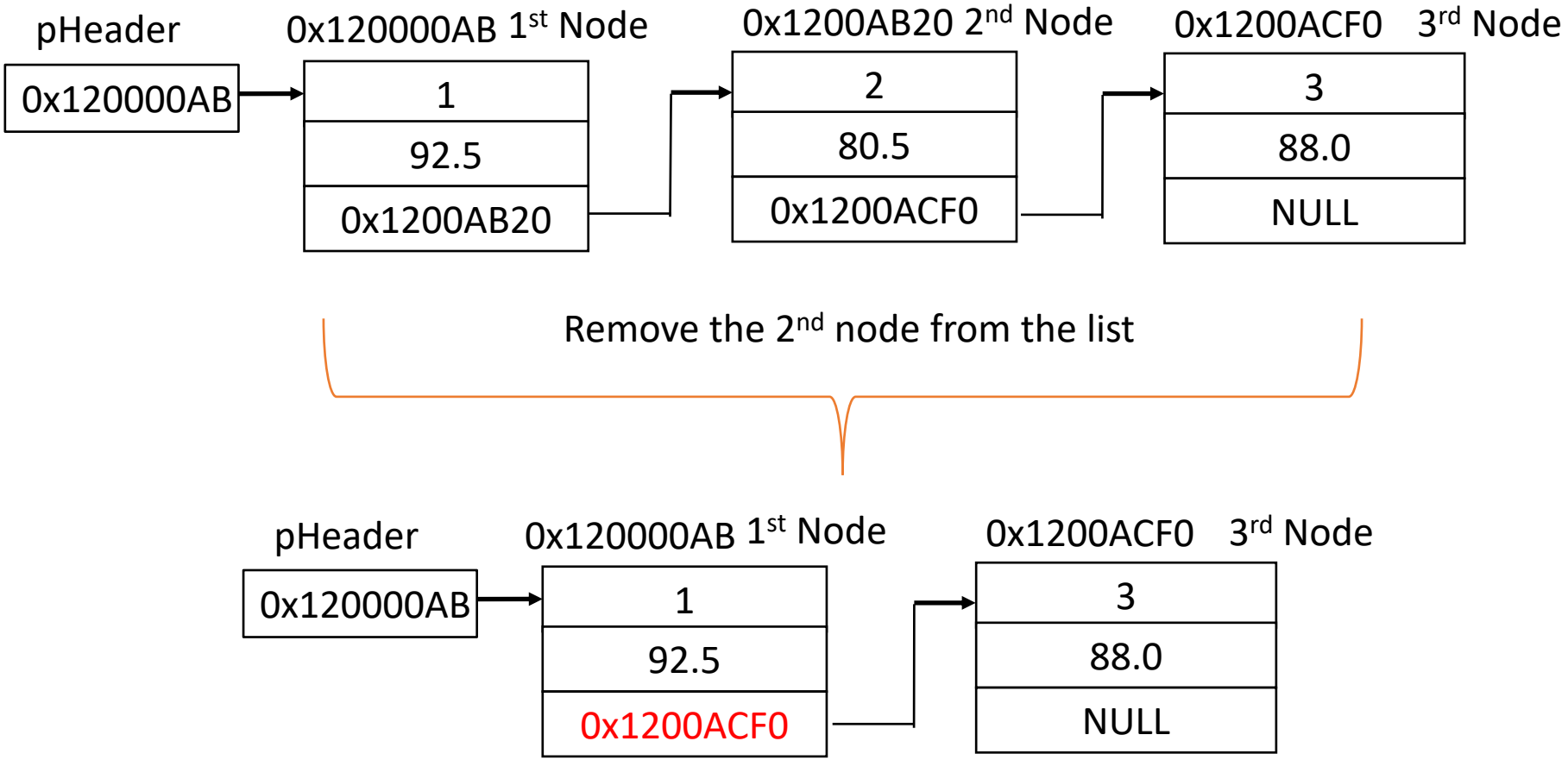
- Create a new node and insert it to the specific position of the list





Remove a Node from a List

- Remove a specific node from the list



Notes of Inserting or Removing a Node

- Searching of a node is usually required when a node is inserted or removed
- When insert or remove a node from a list, take care of the relationship of node connection.
- The details for the implementation of inserting or removing are given in the course of “data structure”

Destroy a List

- If memory of each node is allocated by function malloc(), these memory should be released.

```
void destroyList(struct student *p)
{
    struct student *pNode;
    while (p != NULL)
    {
        pNode = p;
        p = p->pNext;
        free(pNode);
    }
}
```

Answers of Exercise

- There are N students in a class. Each student takes three courses. Calculate the average scores of each student, and output the scores according to a descending order.
  - Requirement : N is inputted when the program is executed, the memory should be dynamically allocated.

Idea:

- Allocate the memory of four dynamic arrays.
  Three of them are used to store the scores of each courses.
  The other array is used to store the average scores.
- When we sort the average scores, we need another array to record the index of each student. We need to allocate memory for this array either.

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

void sort(float *pScore, int *pInd, int n)
{
    int i, j;
    float fTem, iTem;
    for ( i = 0 ; i < n-1 ; i++ )
    {
        for ( j = 0 ; j < n-1-i ; j++ )
        {
            if ( pScore[j] < pScore[j+1] )
            {
                fTem = pScore[j]; pScore[j] = pScore[j+1];
                pScore[j+1] = fTem;
                iTem = pInd[j] ; pInd[j] = pInd[j+1];
                pInd[j+1] =iTem;
            }
        }
    }
}

.........
```

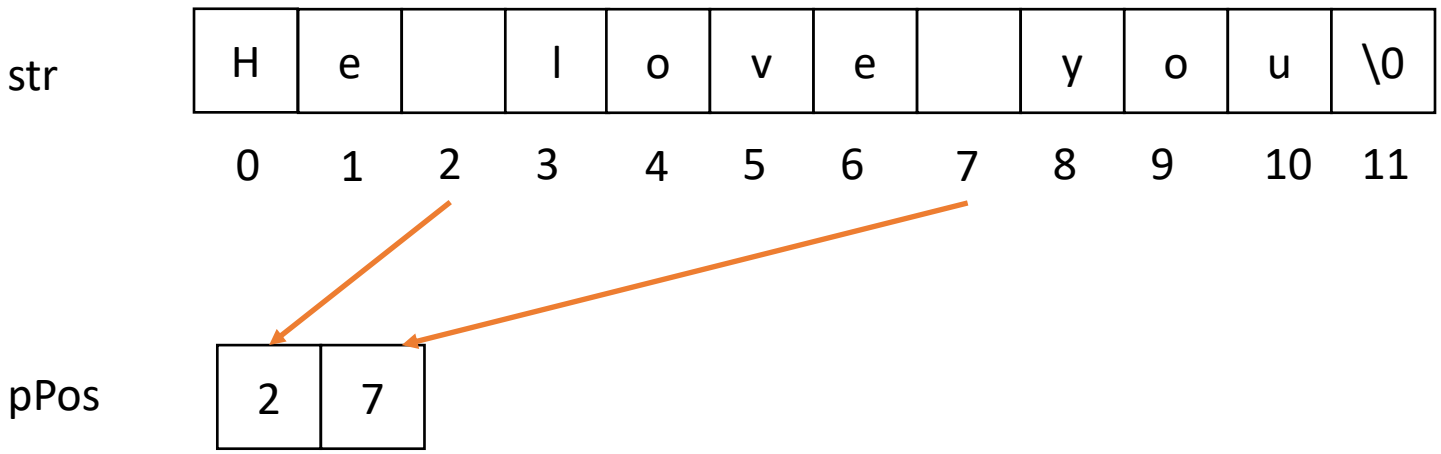
```
.........
int main()
{
    int num, i;
    float *pCourse1, *pCourse2, *pCourse3, *pAverage;
    int *pInd;

    scanf("%d", &num);
    pCourse1 = (float *)malloc(sizeof(float)*num);
    pCourse2 = (float *)malloc(sizeof(float)*num);
    pCourse3 = (float *)malloc(sizeof(float)*num);
    pAverage = (float *)malloc(sizeof(float)*num);
    pInd = (int *)malloc(sizeof(int)*num);

    for ( i = 0 ; i < num ; i++ ){
        scanf("%f %f %f", pCourse1+i, pCourse2+i, pCourse3+i);
        pAverage[i] = (pCourse1[i] + pCourse2[i] + pCourse3[i]) / 3.0f;
        pInd[i] = i;
    }

    sort(pAverage, pInd, num);
    for ( i = 0 ; i < num ; i++ ) printf("%d %f\n", pInd[i], pAverage[i]);

    free(pCourse1); free(pCourse2); free(pCourse3); free(pAverage); free(pInd);
    return 0;
}
```

Answers of Exercise

- Reverse the order of words in a string
  - Example: the input is “He love you”, and the output is “you love He”

Idea:

- Split the string according to the space character
- We need an array to record the positions of all space characters, and then we can output the words in a reversed order according to them



According to the array pPos, output each words in the reversed order

Output characters from str[8] (7+1) to str[10] (last)
Output characters from str[3] (2+1) to str[6] (7-1)
Output characters from str[0] (first) to str[1] (2-1)

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

int main()
{
    char str[256];
    //num – character number, spaceNum – space character number
    int i, j,  num = 0, spaceNum = 0;
    int *pPos = NULL; //record positions of space characters
    gets(str);

    i = 0;
    //calculate number of characters and space characters
    while(str[i] != '\0')
    {
        if (str[i] == ' ') spaceNum++;
        num++; i++;
    }
    if ( spaceNum == 0 ) // if no space characters
    {
        printf("%s", str); return 0;
    }

    pPos = (int *)malloc(sizeof(int)*spaceNum);
    ......
```

```
    .......
    i = 0; j = 0;
    while(str[i] != '\0') // record the positions of space characters
    {
        if (str[i] == ' ')
        {
            pPos[j] = i; j++;
        }
        i++;
    }

    for ( i = pPos[spaceNum-1]+1 ; i <= num -1 ; i++ )
        printf("%c", str[i]);
    printf(" ");
    for ( j = spaceNum - 2 ; j >= 0 ; j-- )
    {
        for ( i = pPos[j]+1 ; i <= pPos[j+1]-1 ; i++ )
            printf("%c", str[i]);
        printf(" ");
    }
    for ( i = 0 ; i <= pPos[0]-1 ; i++ )
        printf("%c", str[i]);
    printf("\n");
    free(pPos);
    return 0;
}
```