Practice: File I/O, Structures, Union, and Abstract Data Types

FILE I/O

- fopen, fclose
- fgetc, fscanf, fgets, fread…
- fputc, fprintf, fputs, fwrite…
- Google is your friend!

```c
#include <stdio.h>
void swap1(char s1, char s2);
void swap2(char * s1, char * s2);
void swap3(char ** s1, char ** s2);
int main(void)
{
        FILE * f1;
        FILE * f2;
        char tmp;
        char buffer[1024];

        f1 = fopen("input.txt", "r");
        f2 = fopen("output.txt", "w");

        printf("%c\n", fgetc(f1));
        fprintf(stdout, "%c\n", fgetc(f1));
        fprintf(f2, "%c\n", fgetc(f1));

        scanf("%c", & tmp);
        printf("%c\n", tmp);
        fscanf(stdin, "%s", buffer);
        printf("%s\n", buffer);

        fscanf(f1, "%c", &tmp);
        printf("%c\n", tmp);

        fgets(buffer, 1024, f1);
        printf("%s", buffer); // be aware of \n character!

        fscanf(f1, "%s", buffer);
        printf("%s", buffer);
        printf("End! \n");
```

FILE I/O

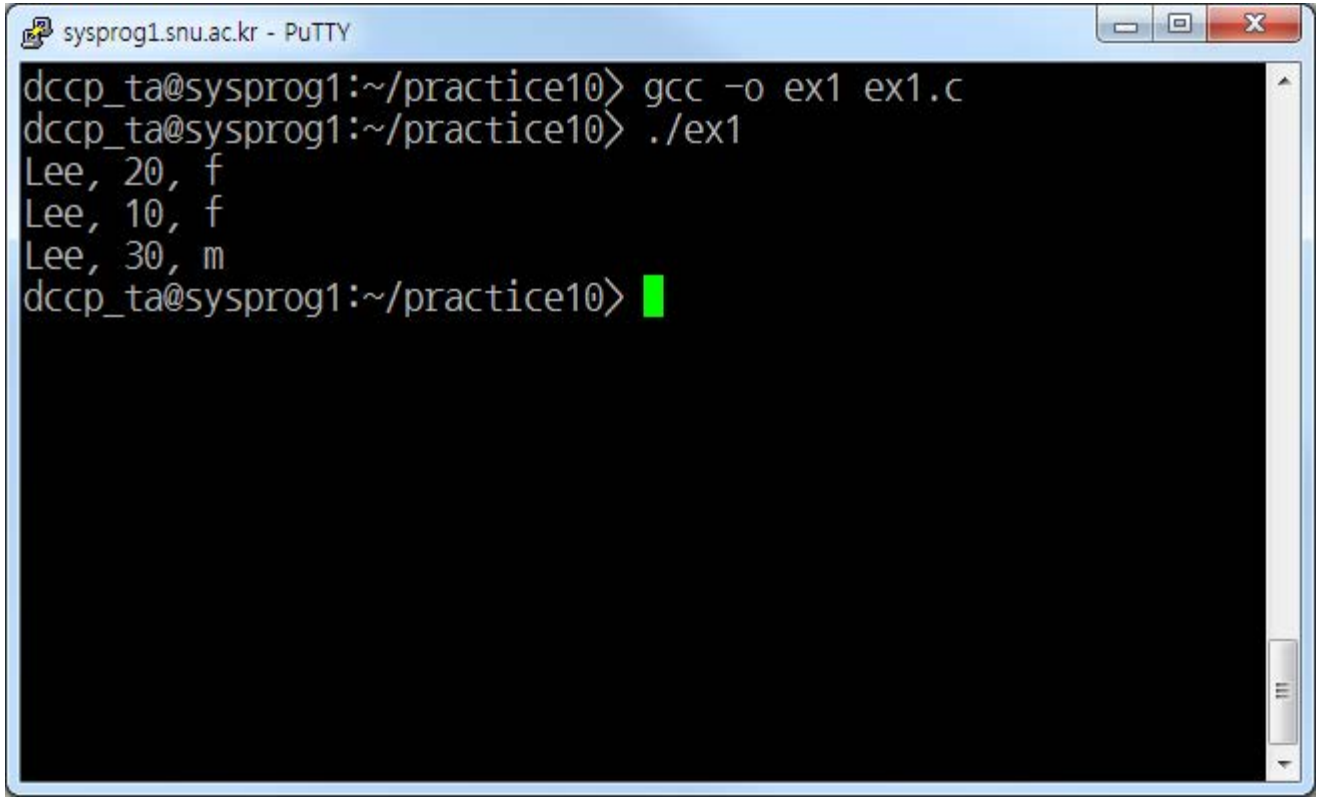
```
dccp_ta@sysprog1:~> cat input.txt
abcdefg
hijklmnop
qrstuvwxyz
dccp_ta@sysprog1:~> !./
./a.out
a
b
12345
1
2345
d
efg
hijklmnopEnd!
dccp_ta@sysprog1:~> cat output.txt
c
dccp_ta@sysprog1:~>
```

Structure

- Struct person introduces a structure which contains three members, i.e., name, age, and gender

```
#include <stdio.h>

struct person {
    char name[10];
    int age;
    char gender;
};

int main()
{
  struct person per1 = {"Lee", 20, 'f'};
  struct person per2;
  struct person *pp;

  printf("%s, %d, %c\n", per1.name,
      per1.age, per1.gender);
  per1.age = 10;
  per2 = per1;
  printf("%s, %d, %c\n", per2.name,
      per2.age, per2.gender);

  pp = &per1;
  (*pp).age = 30;
  pp->gender = 'm';
  printf("%s, %d, %c\n", pp->name,
      pp->age, pp->gender);

  return 0;
}
```

Structure (contd.)



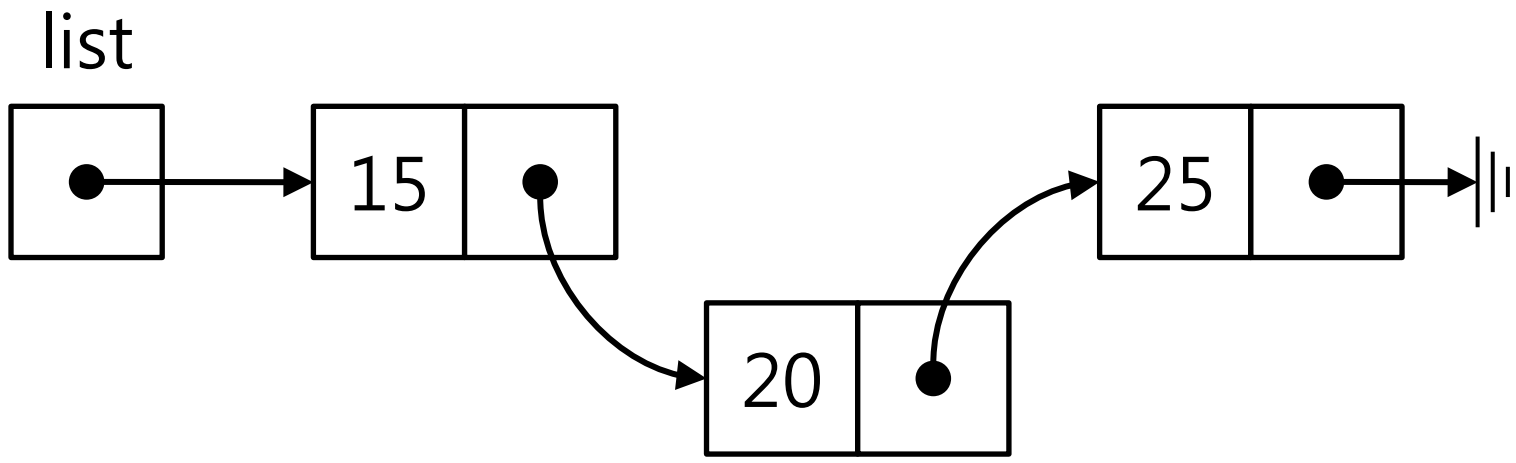
Union

- members share storage

```
typedef union foo {
    int n;
    float r;
} number;

int main(void)
{
    number m;
    m.n = 2345;
    printf("n: %10d  r: %16.10e\n", m.n, m.r);
    m.r = 2.345;
    printf("n: %10d  r: %16.10e\n", m.n, m.r);
    return 0;
}
```

ADT: Linked Lists

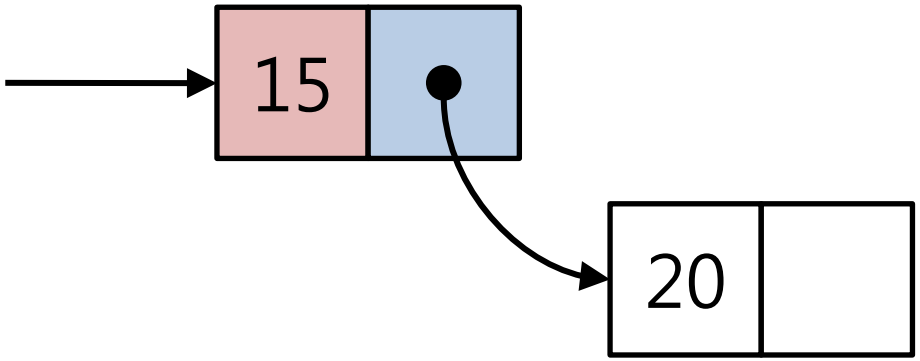
- A list of elements a_1, a_2, ..., a_k
- Each node contains a reference to the next node in the sequence
- What is the benefit of a linked list over an array?



ADT: Linked Lists (contd.)

- A node of a linked list is implemented with a structure
- It has two members: element and next
  - Next is a pointer to struct node

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

struct node {
  int element;
  struct node *next;
};
```

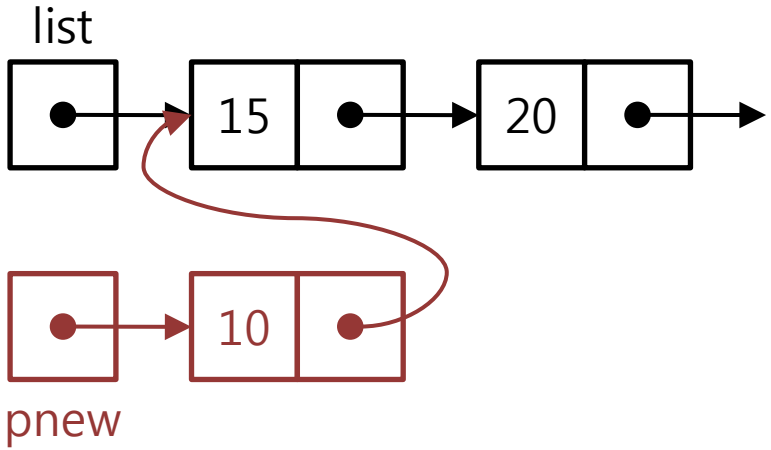


ADT: Linked Lists (contd.)

- Linsert(list, x) gets list, which is a pointer to a linked list, and x as its input
  - It inserts x at the front of the list

```
struct node *Linsert(struct node *list,
                     int x)
{
  struct node *pnew;
  pnew = malloc(sizeof(struct node));
  if (pnew == NULL) {
    printf("malloc error\n");
    return NULL;
  }
  pnew->element = x;
  pnew->next = list;
  return pnew;
}
```

e.g. `list = Linsert(list, 10)`

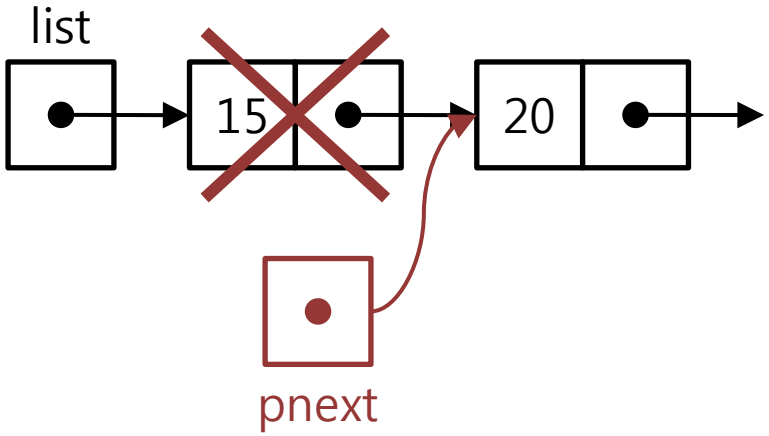


ADT: Linked Lists (contd.)

- Lremove(list) also gets list, which is a pointer to a linked list as its input
  - It removes the first element from the list

```
struct node *Lremove(struct node *list)
{
  struct node *pnext;
  if (list == NULL) {
    printf("list is empty");
    return list;
  }
  pnext = list->next;
  free(list);
  return pnext;
}
```

e.g. `list = Lremove(list)`



ADT: Linked Lists (contd.)

- Lsearch(list, x) gets list and x as its input
  - If x is in the list, it returns the pointer of the node that contains x; NULL otherwise
- PrintList prints all elements in the list

```
struct node *Lsearch(
    struct node *list, int x)
{
  struct node *pn;
  for (pn = list;
       pn != NULL;
       pn = pn->next)
    if (pn->element == x) return pn;
  return NULL;
}
```

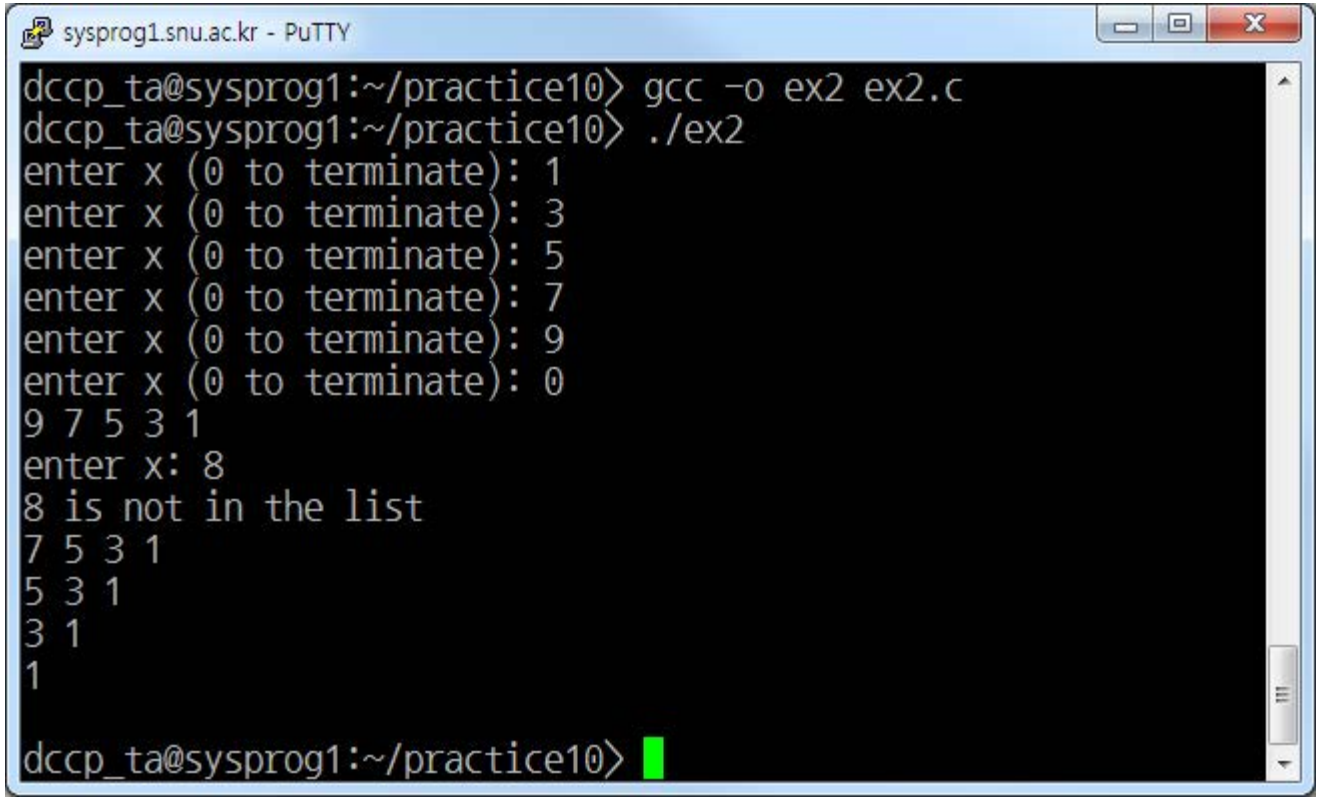
```
void PrintList(struct node *list)
{
  if (list == NULL) {
    printf("\n");
    return;
  }
  printf("%d ", list->element);
  PrintList(list->next);
}
```

ADT: Linked Lists (contd.)

- A program that creates a linked list by inserting elements, and searches it for an element x

```
int main()
{
  struct node *list = NULL;
  int x;

  while (1) {
    printf("enter x (0 to terminate): ");
    scanf("%d", &x);
    if (x == 0) break;
    list = Linsert(list, x);
  }
  PrintList(list);
  printf("enter x: ");
  scanf("%d", &x);

  if (Lsearch(list, x) != NULL)
    printf("%d is in the list\n", x);
  else
    printf("%d is not in the list\n", x);

  while (list != NULL) {
    list = Lremove(list);
    PrintList(list);
  }

  return 0;
}
```

ADT: Linked Lists (contd.)



ADT: Stack

- A list of elements a_1, a_2, ..., a_k
- Elements are inserted and deleted only at one place called the top

```
#include <stdio.h>
#define MAX 100

struct stack {
   int starray[MAX];
   int top;
};
```

ADT: Stack (contd.)

```
struct stack *create(void)
{
   struct stack *st;

   st = malloc(sizeof(struct stack));
   if (st == NULL) {
      printf("malloc error\n");
      return NULL;
   }
   st->top = 0;
   return st;
}
```

ADT: Stack (contd.)

```
int is_empty(struct stack *st)
{
   return st->top == 0;
}

void push(struct stack *st, int x)
{
   if (st->top == MAX) {
      printf("stack is full\n");
      return;
   }
   st->starray[st->top++] = x;
}
```

ADT: Stack (contd.)

```
int pop(struct stack *st)
{
   if (is_empty(st)) {
      printf("stack is empty\n");
      return;
   }

   return st->starray[--st->top];
}
```

ADT: Stack (contd.)

```
int main(void)
{
   struct stack *st;
   int x;

   st = create();
   while (1) {
      printf("enter x (0 to terminate): ");
      scanf("%d", &x);
      if (x == 0) break;
      push(st, x);
   }
   while (!is_empty(st))
      printf("%d ", pop(st));
   printf("\n");

   return 0;
}
```

Exercise

Exercise 1

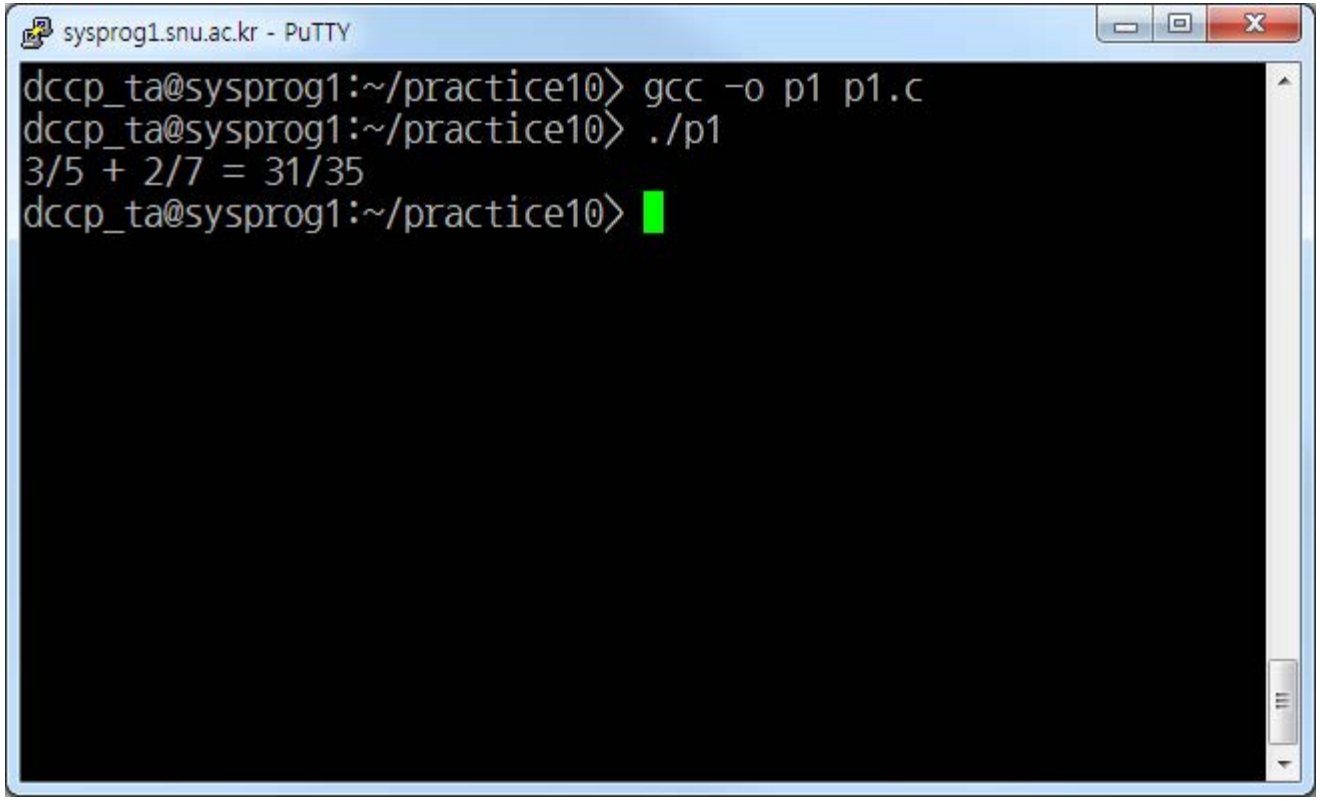
- Write a program that reads a file and removes all the whitespace characters
  - input.txt, output.txt

Exercise 2

- 1. Declare a structure for fraction numbers
- 2. Write a function fracsum(a, b) for fraction number addition

```
struct fraction a = {3, 5}, b = {2, 7};
struct fraction c;

c = fracsum(a, b);

printf("%d/%d + %d/%d = %d/%d\n", a.num,
  a.denom, b.num, b.denom, c.num, c.denom);
```

Exercise 2 (contd.)

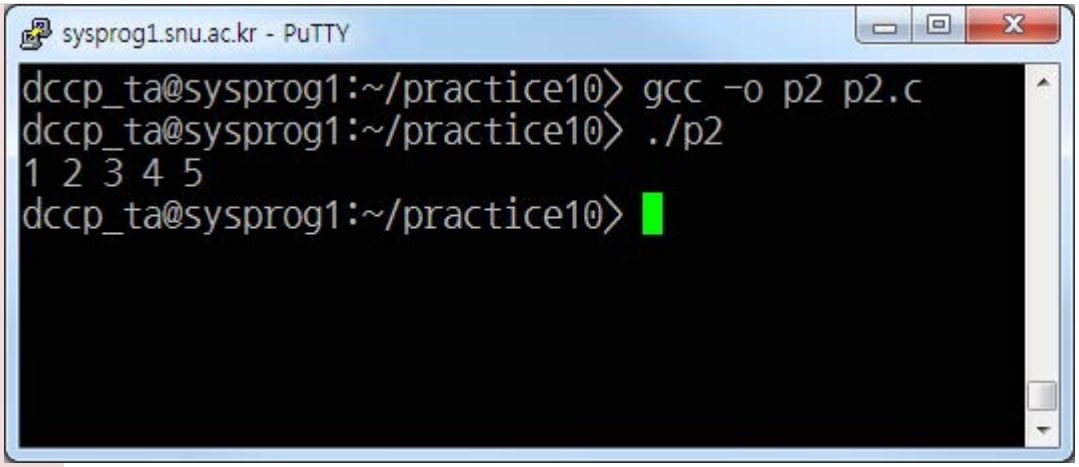


Exercise 3

- Write the following linked list operations
  - struct node *Lappend(struct node *list1, struct node *list2)
    - Appends list2 at the end of list1
    - Be careful that list1 may be NULL

Exercise 3 (contd.)

```
struct node *list1 = NULL;
struct node *list2 = NULL;
struct node *list3;

list1 = Linsert(list1, 3);
list1 = Linsert(list1, 2);
list1 = Linsert(list1, 1);
list2 = Linsert(list2, 5);
list2 = Linsert(list2, 4);
list3 = Lappend(list1, list2);
PrintList(list3);
```




```
struct node *list = NULL;
list = Linsert(list, 8);
list = Linsert(list, 4);
list = Lappend(NULL, list);
PrintList(list);
```