Week 2

Struct

Initialize a struct

```
#include <cstdio>
int main() {
    struct Student {
        char id;
        unsigned char chinese;
        unsigned char english;
        unsigned char math;
    };
    Student s[] = {
        {'A', 41, 13, 93},
        {'B', 43, 21, 38},
        {'C', 11, 69, 78},
        {'D', 81, 86, 80},
        {'E', 19, 47, 59} };
    for (int i=0; i<sizeof(s)/sizeof(s[0]); ++i)
        printf("%c %d %d %d\n", s[i].id,
                s[i].chinese, s[i].english, s[i].math );
    return 0;
}
```

C++ equivalent to stdio.h

By convention, we capitalize the type name.

warning: comparison between signed and unsigned integer

Read From a Text File

```
#include <cstdio>
int main() {
    struct Student {
        char id;
        unsigned char chinese;
        unsigned char english;
        unsigned char math;
    };
    Student s;
    FILE* fp;
    fp = fopen("b.txt", "r");

    for (size_t i=0; i<5; ++i) {
        fscanf(fp, "%c %hhu %hhu %hhu\n", &s.id,
                &s.chinese, &s.english, &s.math);
        printf("%c %d %d %d\n", s.id,
                s.chinese, s.english, s.math );
    }
    return 0;
}
```

u - unsigned int
h - short (half)
hh - char (half-half)

Precedence of '.' is higher than '&'

Q: What is the size of this text file?

```
A 41 13 93
B 43 21 38
C 11 69 78
D 81 86 80
E 19 47 59
```

Write Into a Binary File

```
#include <cstdio>

int main() {
    struct Student {
        char id;
        unsigned char chinese;
        unsigned char english;
        unsigned char math;
    };
    Student s;
    FILE* fp = fopen("b.txt", "r");
    FILE* outfile = fopen("b.dat", "wb");

    for (size_t i=0; i<5; ++i) {
        fscanf(fp, "%c %hhu %hhu %hhu\n", &s.id,
                &s.chinese, &s.english, &s.math);
        fwrite(&s, sizeof(s), 1, outfile);
    }
    fclose(fp);
    fclose(outfile);
    return 0;
}
```

1. This is a binary file which you cannot inspect its contents with a text editor.
2. Try to dump its contents with "od -t d1 b.dat" or "xxd b.dat"

Q: What is the size of the file "b.dat"?

fwrite(const void *ptr, size_t size, size_t nmemb, FILE *stream);

size vs. nmemb

```
#include <cstdio>

int main() {
    struct Student {
        char id;
        unsigned char chinese;
        unsigned char english;
        unsigned char math;
    };
    Student s[] = {
        {'A', 41, 13, 93},
        {'B', 43, 21, 38},
        {'C', 11, 69, 78},
        {'D', 81, 86, 80},
        {'E', 19, 47, 59} };

    FILE* outfile = fopen("b1.dat", "wb");
    fwrite(s, 4, 5, outfile);
    fclose(outfile);
    return 0;
}
```

Same output,
but you don't need a for-loop.

time(2)

man 2 time

- NAME
  - time - get time in seconds
- SYNOPSIS
  - `#include <ctime>`
  - `time_t time(time_t *tloc);`
- DESCRIPTION
  - time() returns the time as the number of seconds since the Epoch, 1970-01-01 00:00:00 +0000 (UTC).
  - If tloc is non-NULL, the return value is also stored in the memory pointed to by tloc.
- RETURN VALUE
  - On success, the value of time in seconds since the Epoch is returned.

time()

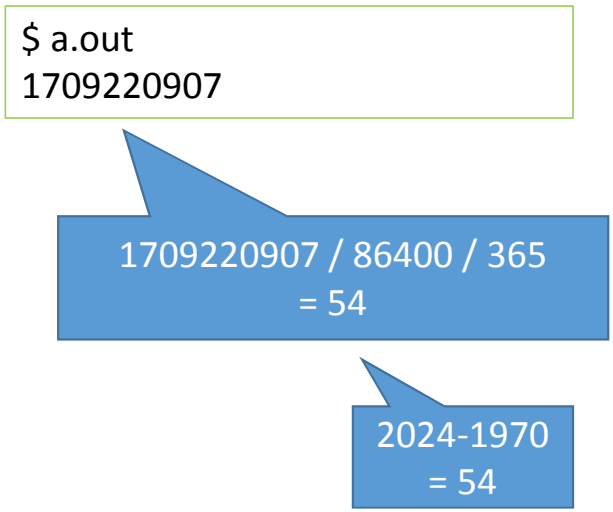
```
#include <ctime>
time_t time ( time_t * timer );
```

```
/* time example */
#include <iostream>
#include <ctime>

int main ()
{
  time_t seconds;

  seconds = time(NULL);
  std::cout << seconds/3600
    << " hours since"
    << " January 1, 1970\n";

  return 0;
}
```

```
/* time example */
#include <iostream>
#include <ctime>

int main ()
{
  time_t seconds;

  time(&seconds);
  std::cout << seconds/3600
    << " hours since"
    << " January 1, 1970\n";

  return 0;
}
```

time()

```
#include <iostream>
#include <ctime>
using std::cout;
using std::endl;

int main() {
    time_t t;
    time(&t);
    cout << t << endl;
    return 0;
}
```



srand()

```
#include <cstdio>
#include <ctime>
#include <cstdlib>

int main() {
    time_t t = time(NULL);     // current time
    // srand(t);               // random seed
    for (int i=0; i<5; ++i)
        printf("%d\t", rand() % 100);
    printf("\n");
    return 0;
}
```

A convenient random seed

struct tm *localtime(const time_t *timep);

```
struct tm {
    int tm_sec;     /* Seconds (0-60) */
    int tm_min;     /* Minutes (0-59) */
    int tm_hour;    /* Hours (0-23) */
    int tm_mday;    /* Day of the month (1-31) */
    int tm_mon;     /* Month (0-11) */
    int tm_year;    /* Year - 1900 */
    int tm_wday;    /* Day of the week (0-6, Sunday = 0) */
    int tm_yday;    /* Day in the year (0-365, 1 Jan = 0) */
    int tm_isdst;   /* Daylight saving time */
};
```

Convert a Timestamp to `struct tm`

```
int main() {
    time_t t;
    struct tm *now;
    time(&t);
    now = localtime(&t);
    printf("%d-%d-%d\n",
          now->tm_year , now->tm_mon , now->tm_mday);
    return 0;
}
```

122-1-24

```
int tm_mday;   /* Day of the month (1-31) */
int tm_mon;    /* Month (0-11) */
int tm_year;   /* Year - 1900 */
```

You may also get tm_wday (0~6), which corresponds to Sunday~Saturday.

Exercise: What Day Is Today?

- Design a program which automatically gets the system clock and shows today with the following format:
  - Thursday 2022-2-24

C Time Library <ctime>

- Types
  - **clock_t** - Clock type
  - **size_t** - Unsigned integral type
  - **time_t** - Time type
  - **struct tm** - Time structure (See Chapter 7)
- Time manipulation
  - **clock** - Ticks since the program was launched
  - **time** - Get current time
  - **mktime** - Convert tm structure to time_t
- Macro
  - **CLOCKS_PER_SEC** - Clock ticks per second
- Conversion
  - **asctime** - Convert tm structure to string
  - **ctime** - Convert time_t value to string
  - **gmtime** - Convert time_t to tm as UTC time
  - **localtime** - Convert time_t to tm as local time
  - **strftime** - Format time as string

ctime()

```
#include <ctime>
char * ctime ( const time_t * timer );
```

```
/* ctime example */
#include <iostream>
#include <ctime>

int main ()
{
  time_t t;
  time ( &t );
  std::cout << "The current local time is: "
    << ctime (&t)  << std::endl;
  return 0;
}
```

Converts the time_t object (seconds since 1970.1.1) to a C string containing a human-readable version of the corresponding local time and date.

output

The current local time is:
Sat Nov 10 11:57:33 2012

Almost the same as the command “date” in Linux.

strftime()

```
#include <cstdio>
#include <ctime>

int main() {
    time_t t = time(NULL);
    struct tm* now = localtime(&t);
    char buf1[40];
    char buf2[40];
    char buf3[40];
    strftime(buf1, sizeof(buf1), "%a, %d %b %Y %T %z", now);
    strftime(buf2, sizeof(buf2), "%Y-%m-%d %H:%M:%S", now);
    strftime(buf3, sizeof(buf3), "%F %T", now);
    printf("%s\n%s\n%s\n", buf1, buf2, buf3);
    return 0;
}
```

Thu, 24 Feb 2022 00:28:59 +0800

2022-02-24 00:33:44

struct timeval

- Q: The unit of time_t is “second”, which is not accurate enough for me. Is there any function in Linux that can measure time in millisecond (ms)?
- A: Actually you can measure in microsecond (μs), with another function.
  - #include <sys/time.h>
  - int gettimeofday(struct timeval *tv, struct timezone *tz);
    - struct timeval {
    - time_t tv_sec; /* seconds */
    - suseconds_t tv_usec; /* microseconds */
    - };

gettimeofday()

```
#include <stdio.h>
#include <sys/time.h>

int main() {
    struct timeval tv;
    gettimeofday(&tv, NULL);
    printf("%ld.%06ld\n", tv.tv_sec, tv.tv_usec);
    return 0;
}
```

Exercise: Measure Elapsed Time

- Try to get the timestamp before you perform a task (e.g., a calculation test of 10 multiplications).
- Get the timestamp after you finish the task.
- Calculate and display how many seconds (and microseconds) has elapsed.

A struct can contain a pointer (P.274)

```
struct ListElement
{
  RECT aRect;              // RECT member of structure
  ListElement* pNext;      // Pointer to a list element
};
```

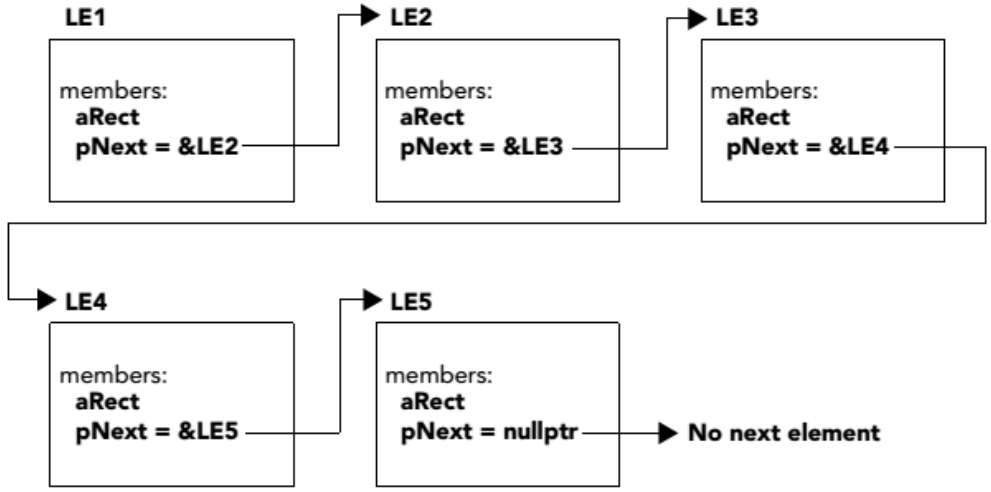
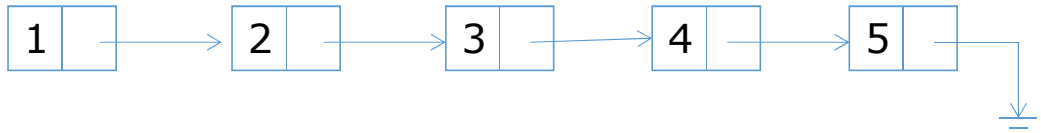


FIGURE 7-3 Linked List

Create a Linked List

```
struct ListElement
{
  int value;          // value of an element
  ListElement* pNext;     // Pointer to a list element
};
int main()
  {
      ListElement LE5 = { 5, NULL };
      ListElement LE4 = { 4, &LE5 };
      ListElement LE3 = { 3, &LE4 };
      ListElement LE2 = { 2, &LE3 };
      ListElement LE1 = { 1, &LE2 };
      PrintList(&LE1);
      return 0;
  }
```



Dynamic Memory Allocation (P.163)

- Sometimes depending on the input data, you may allocate different amount of space for storing different types of variables at execution time

```
int n = 0;
cout << "Input the size of the vector - ";
cin >> n;
int vector[n];
```

What if we cannot know in advance how many data we shall handle?
(e.g. the number of people waiting for a school bus on Saturday morning)

Why Use Pointers? (P.148)

- Use pointer notation to operate on data stored in an array
  - *(a + i) == a[i]
- Enable access within a function to arrays, that are defined outside the function
  - void swap(int* a, int* b)
- Allocate space for variables dynamically.

Free Store (Heap)

- To hold a string entered by the user, there is no way you can know in advance how large this string could be.

- Free Store - When your program is executed, there is unused memory in your computer.
- You can dynamically allocate space within the free store for a new variable.

The new Operator

- Request memory for a double variable, and return the address of the space
  - `double* pvalue = NULL;`
  - `pvalue = new double;`
- Initialize a variable created by `new`
  - `pvalue = new double(9999.0);`
- Use this pointer to reference the variable (indirection operator)
  - `*pvalue = 1234.0;`

The delete Operator

- When you no longer need the (dynamically allocated) variable, you can free up the memory space.
  - `delete pvalue;`
    - Release memory pointed to by pvalue
  - `pvalue = NULL;`
    - Reset the pointer to NULL

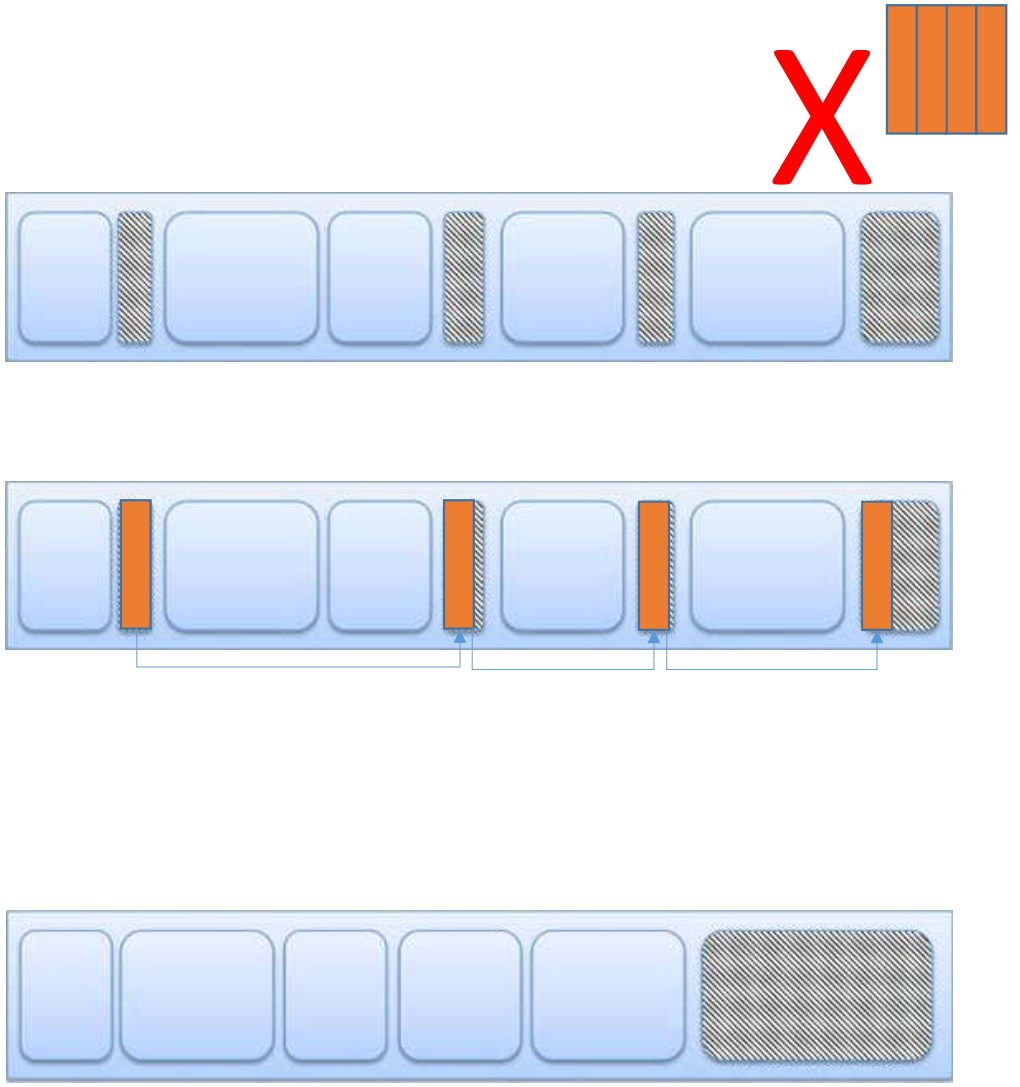
- After you release the space, the memory can be used to store a different variable later.

Allocating Memory Dynamically for Arrays

- Allocate a string of twenty characters
  - `char* pstr;`
  - `pstr = new char[20];`
  - `delete [] pstr;`
    - Note the use of square brackets to indicate that you are deleting an array.
  - `pstr = 0;`
    - Set pointer to null

Linked-List

- Sometimes you cannot find a continuous space to store a huge array.
- However, if you break it into pieces, your memory still have many free segments to store those small pieces.
- All you need is linking them together.
- Alternatively, you can do “memory compaction”, but it’s time-consuming.

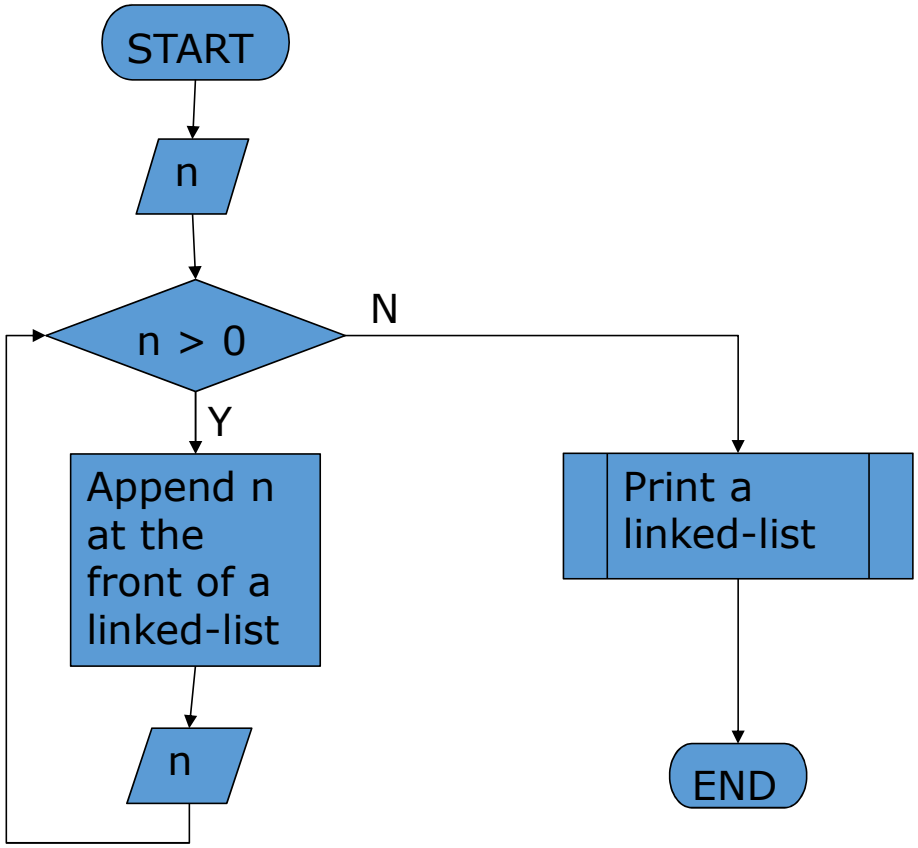


Exercise:
Storing Unknown Number of Integers

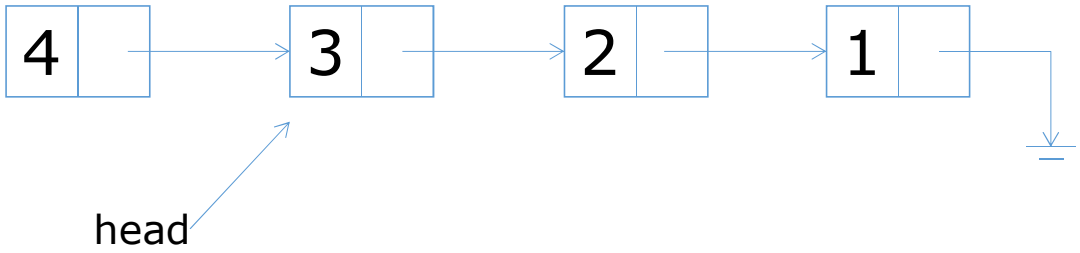
1. Write a program to read a series of positive integers from the user. The total number of input is unknown. Stop when the user supplies 0 or a negative number. Then output the series of numbers in reserve order.
   - For example, the input is 1 3 5 7 2 4 6 0,
     the output will be 6 4 2 7 5 3 1.
   - Hint: Store the input numbers in a linked list.

- See the following 6 slides.

Flowchart

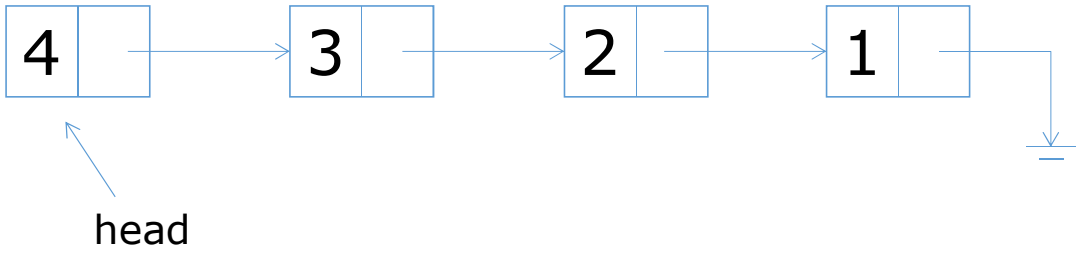

START
n
n > 0
N
Y
Append n at the front of a linked-list
n
Print a linked-list
END

Adding a New Element at the front (1)



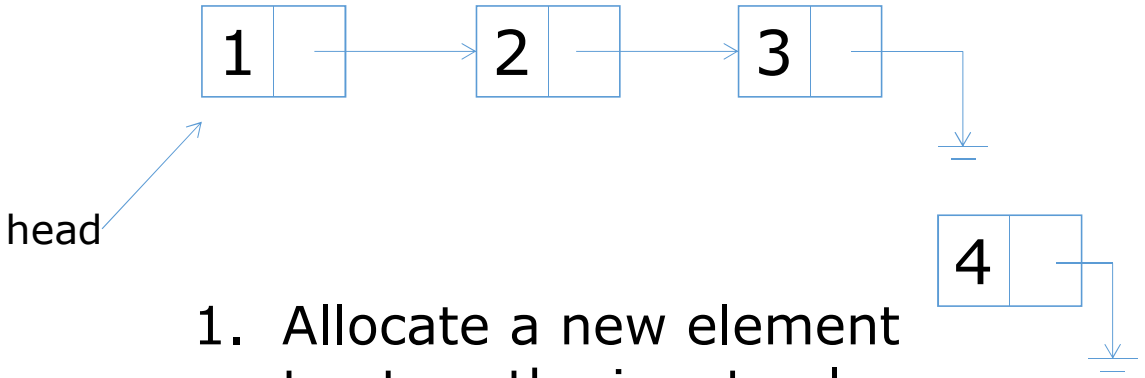
1. Allocate a new element to store the input value.
2. Update LE4.pNext to point to LE3.

Adding a New Element at the front (2)



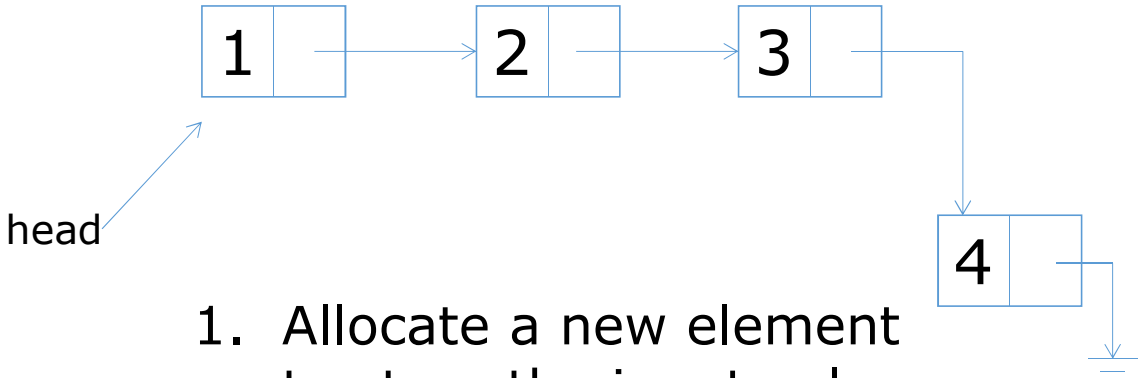
1. Allocate a new element to store the input value.
2. Update LE4.pNext to point to LE3.
3. Update head pointing to LE4.

Appending a New Element (1)



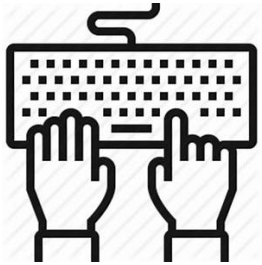
1. Allocate a new element to store the input value.

Appending a New Element (2)

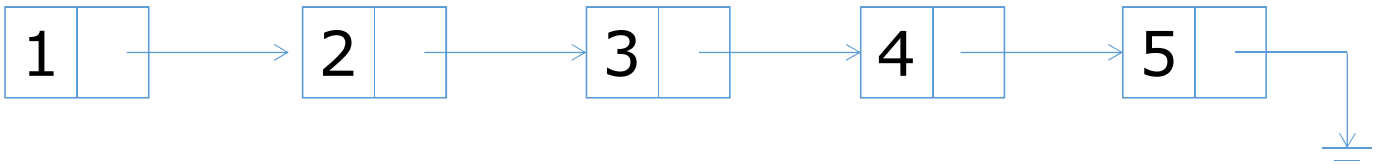


1. Allocate a new element to store the input value.
2. Update LE3.pNext to point to LE4.

Print a Linked List



```
void PrintList(ListElement* p)
{
    while (p != NULL)
    {
        std::cout << p->value;
        p = p->pNext;
    }
}
```



HW: Linked-List Sorting

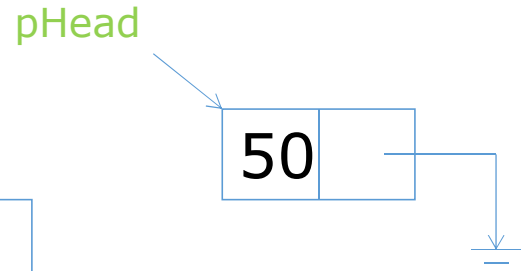
- Input: 1 3 5 7 2 4 6
- Output: 1 2 3 4 5 6 7

- Hint: When you insert a value into a linked-list, you need not always add in the front or at the back of the list. You may insert in a proper position.

- See the following 10 slides.

insertToList(pHead, v)

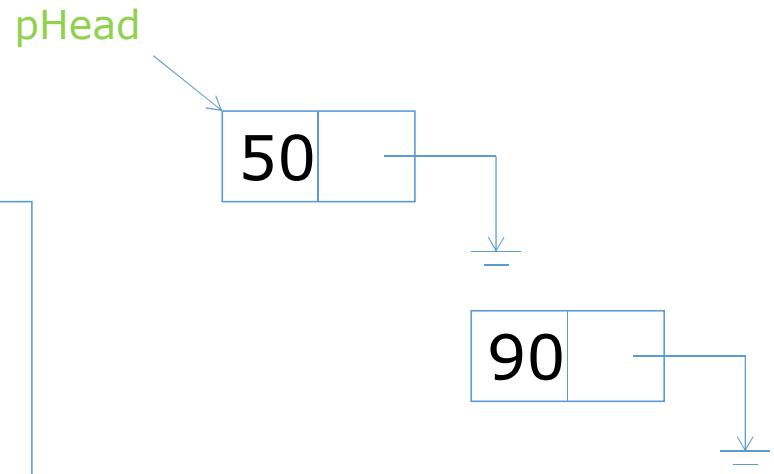
- There will be 4 cases to handle.
- The first case is the simplest - when the list is empty (pHead == NULL).




```
p = new ListElement;
p->value = v;
p->pNext = NULL;

pHead = p;
return pHead;
```

insertToList(pHead, v)

- When the list has only an element (`pHead->pNext == NULL`), and its value is smaller than v.

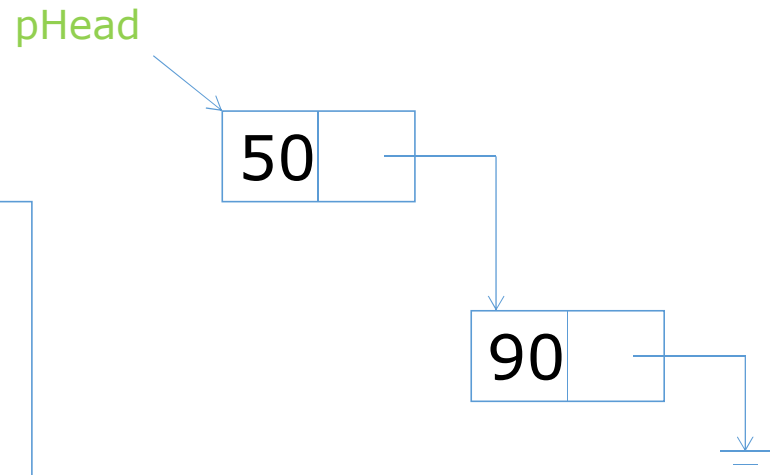



```
p = new ListElement;
p->value = v;
p->pNext = NULL;

pHead->pNext = p;
return pHead;
```

insertToList(pHead, v)

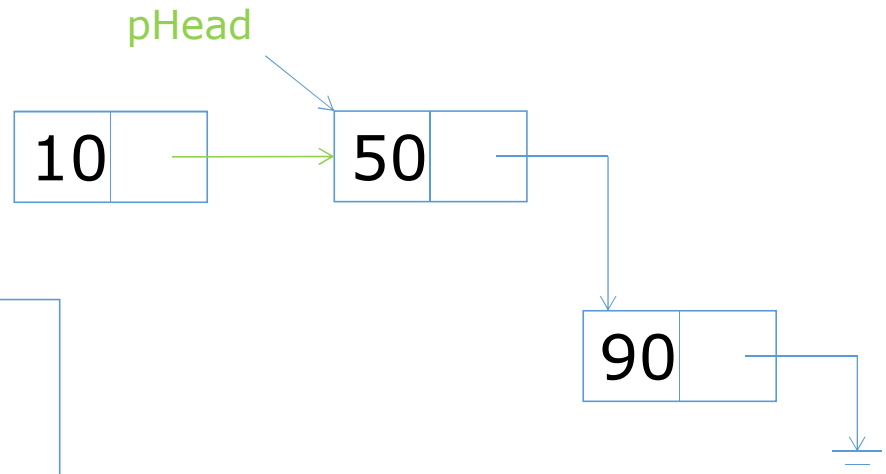
- When the list has only an element, and its value is smaller than v.

```
p = new ListElement;
p->value = v;
p->pNext = NULL;

pHead->pNext = p;
return pHead;
```



insertToList(pHead, v)

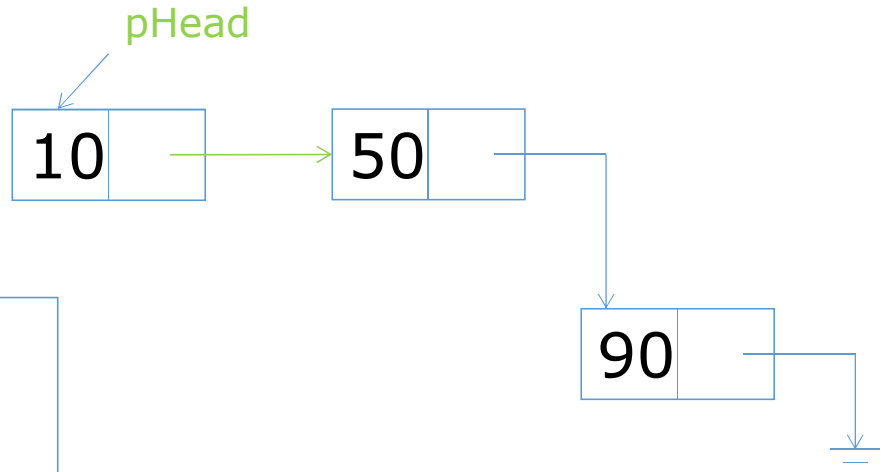
- If v is smaller than the first element, we shall have a new head.




```
p = new ListElement;
p->value = v;
p->pNext = pHead;

pHead = p;
return pHead;
```

insertToList(pHead, v)

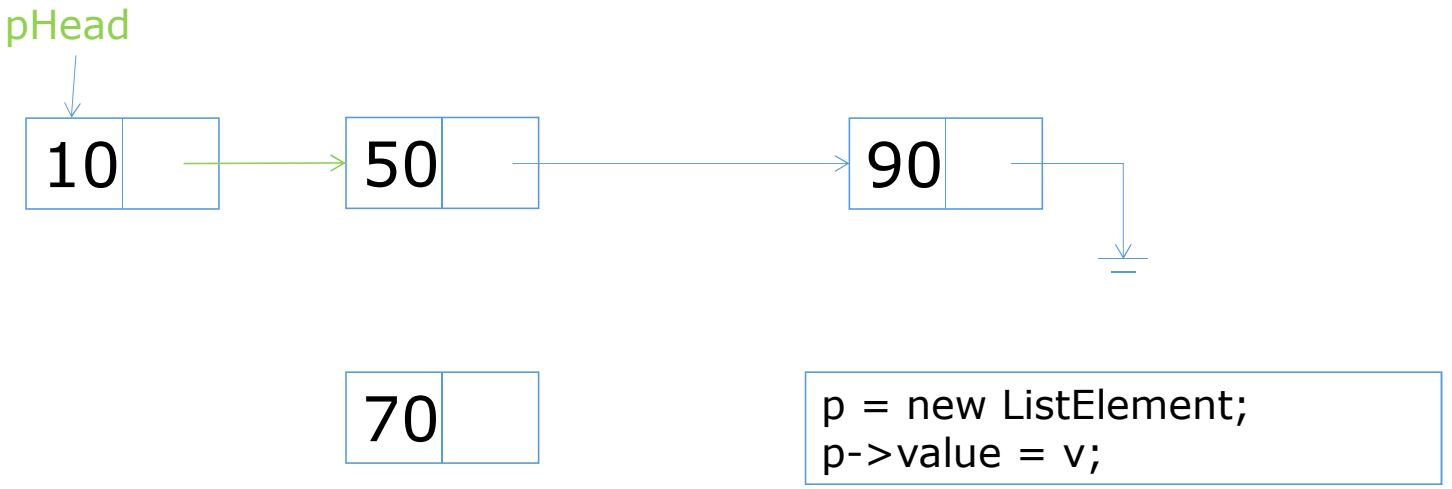
- If v is smaller than the first element, we shall have a new head.




```
p = new ListElement;
p->value = v;
p->pNext = pHead;

pHead = p;
return pHead;
```

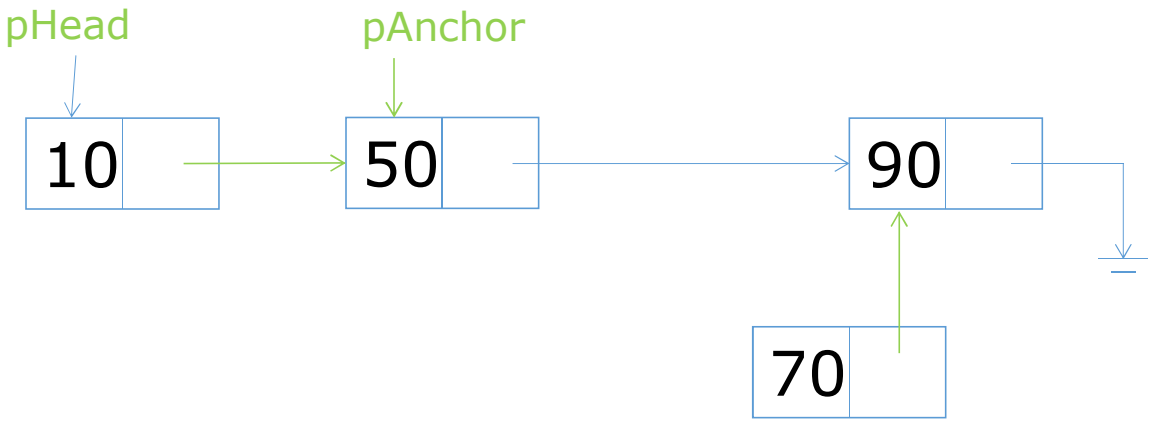
insertToList(pHead, v)

- If v is larger than the first element, let's compare it with the next element.



insertToList(pHead, v)

- If v is still larger, let's continue with the next. When there is no next, or v is smaller, this is the proper position.

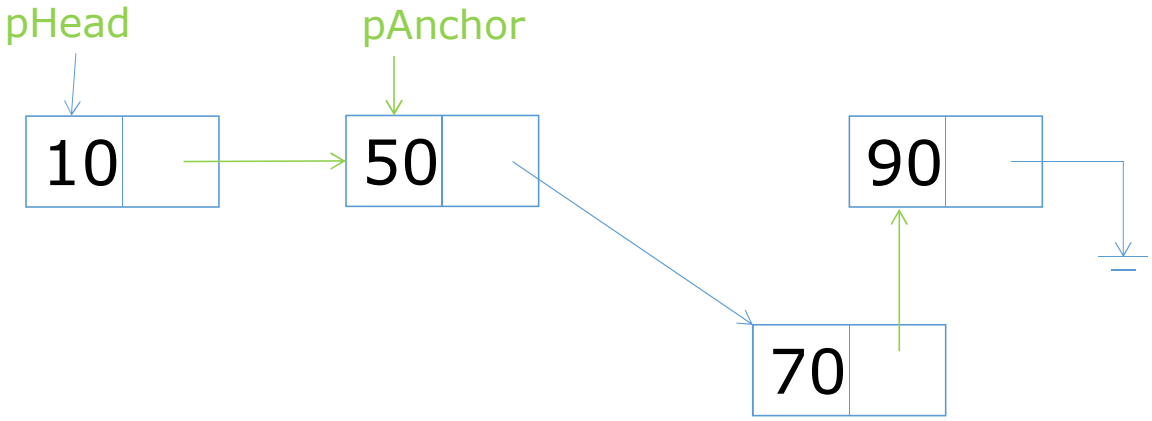


pAnchor is the largest node which is smaller than v.

```
p->pNext = pAnchor->pNext;
pAnchor->pNext = p;
```

insertToList(pHead, v)

- If v is still larger, let's continue with the next. When there is no next, or v is smaller, this is the proper position.




```
p->pNext = pAnchor->pNext;
pAnchor->pNext = p;
```

main()

- The problem can be easily solved by

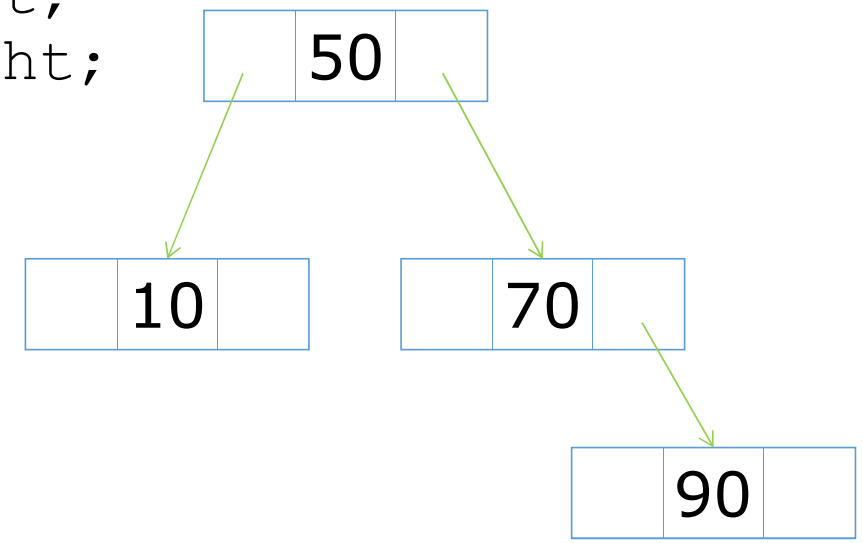
```
int main()
{
    ListElement* pList = NULL;
    int n;
    while (cin >> n)
    {
        pList = insertToList(pList, n);
    }
    printList(pList);
    return 0;
}
```

Binary Tree

Binary Tree

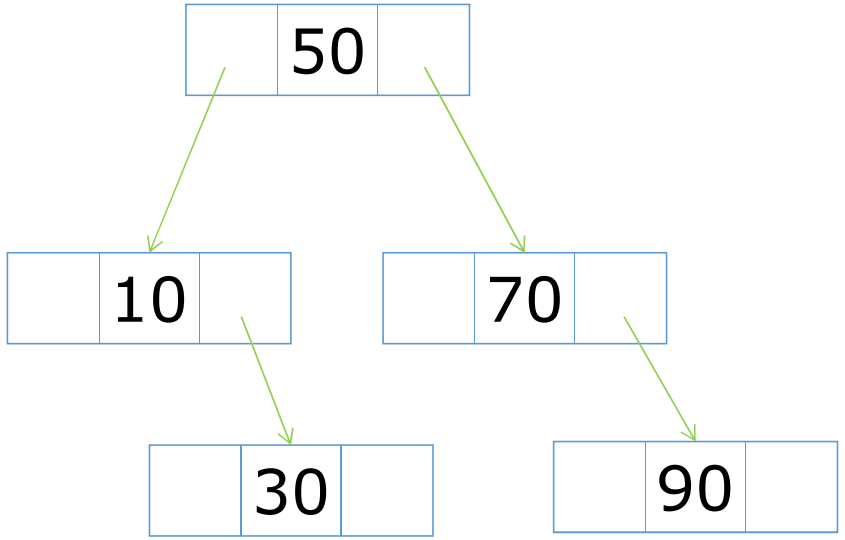
- Let's design a `struct Node`

```
{
    int value;
    Node* pLeft;
    Node* pRight;
};
```



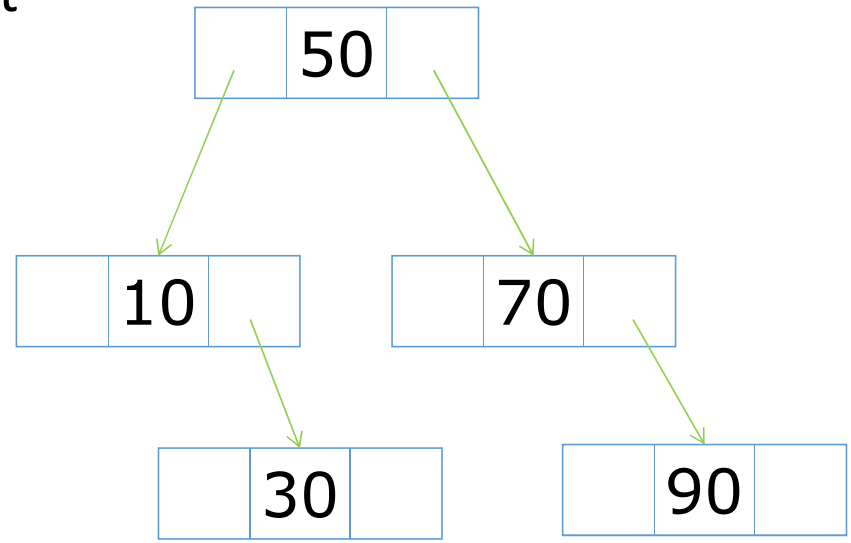
Binary Tree

- Let's design a function `insertToTree()`, which will recursively insert smaller value in its left sub-tree, and larger (or equal) values in its right sub-tree.
- pHead need not be changed.



Binary Tree

- Let's recursively design a function `printTree(p)`, which recursively print
  1. its left subtree,
  2. its value, and
  3. its right subtree.



請這幾位同學下課後來找我

- 112321019 藍俊翔
- 112321023 秦翔浩
- 112321029 蘇崢幃