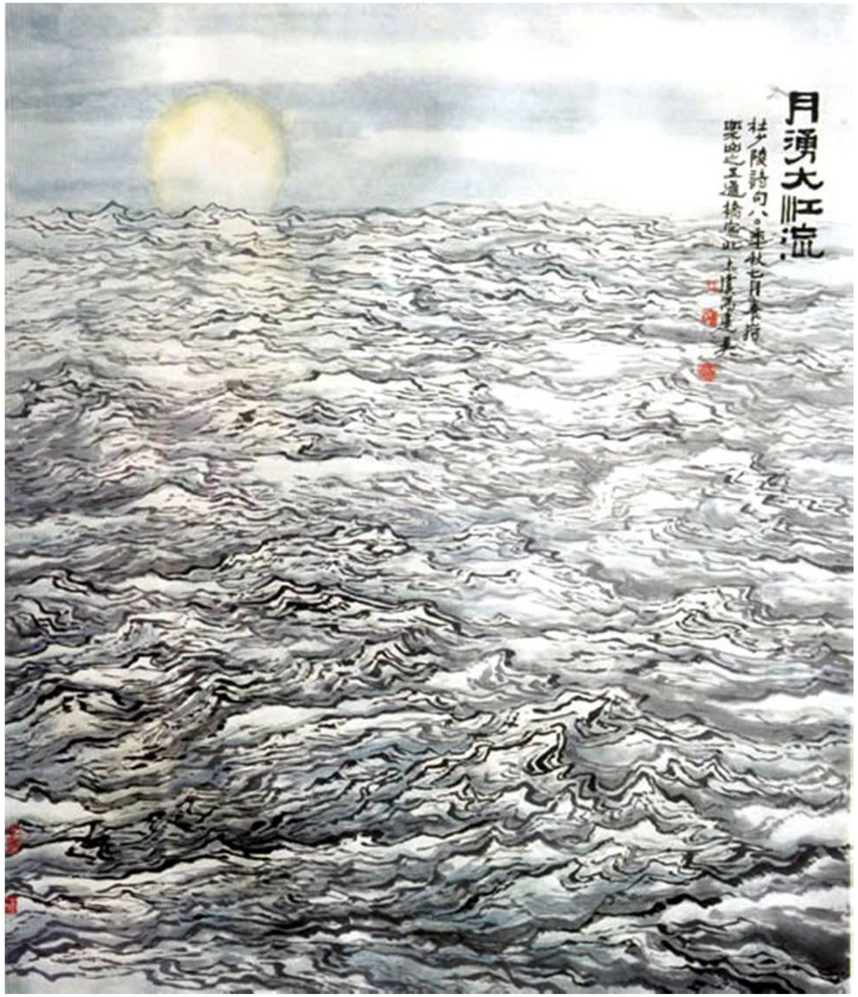
旅夜書懷

~ 杜甫

細草微風岸，
危檣獨夜舟。
星垂平野闊，
月湧大江流。
名豈文章著，
官應老病休。
飄飄何所似，
天地一沙鷗。



Chapter 7

Defining Your Own Data Types

What Is a `struct`?

- A structure is a user-defined type
  - You define it using the keyword `struct`
  - so it is often referred as a **struct**.
- Compared to the data types we have seen, some real world objects must be described by several items:
  - Time – hh:mm:ss
  - Point – (x,y)
  - Circle – (x, y, r)
  - Rational number $\frac{q}{p}$

Defining a struct

```
struct POINT
{
  float x;
  float y;
};
```

- Note:
  - This doesn’t define any variables.
    - It only creates a new type.
  - Each line defining an element in the struct is terminated by a semicolon
  - A semicolon also appears after the closing brace.

Creating Variables of Type POINT

```
POINT p1, p2;
```

- If you also want to initialize a struct:

```
POINT p1 =
{
  1.0,
  2.0
};
```

The syntax is similar to the one to initialize an array.

Accessing the Members of a struct

- Member selection operator (.)
  - `p1.x = 3.0;`
  - `p2.y += 2.0;`
    - `You may manipulate p2.y just as any variable of type float.`

Figure 7-1 on P.356

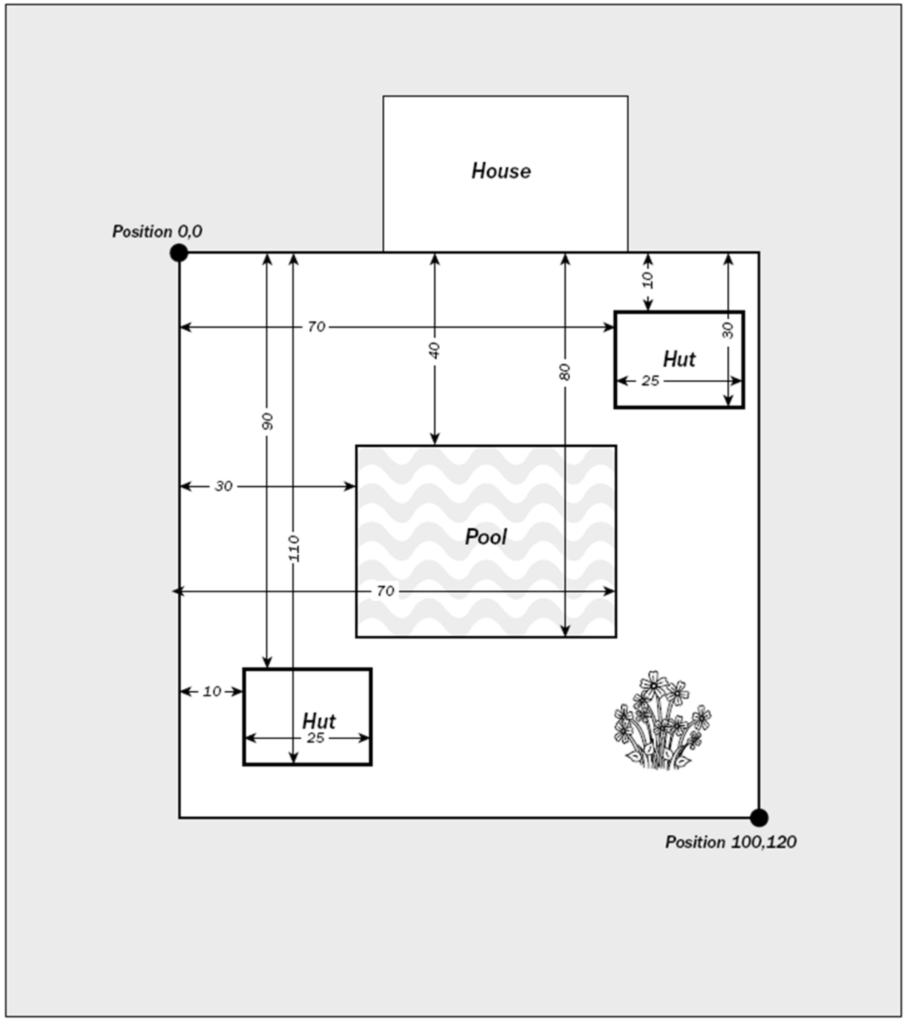


Figure 7-1

Ex7_01.cpp

- Putting the definition of the struct at global scope allows you to declare a variable of type `RECTANGLE` anywhere in the `.cpp` file.
- `Hut2 = Hut1;`
  - `Hut2.Left = Hut1.Left;`
  - `Hut2.Top = Hut1.Top;`
  - `Hut2.Right = Hut1.Right;`
  - `Hut2.Bottom = Hut1.Bottom;`

Pass by Reference

```
long Area(const RECTANGLE& aRect)
{
  return (aRect.Right - aRect.Left) *
  (aRect.Bottom - aRect.Top);
}
```

- By passing a reference , the code runs a little faster because the argument is not copied.
- The parameter is `const` so that the function cannot change the argument that is passed to it.

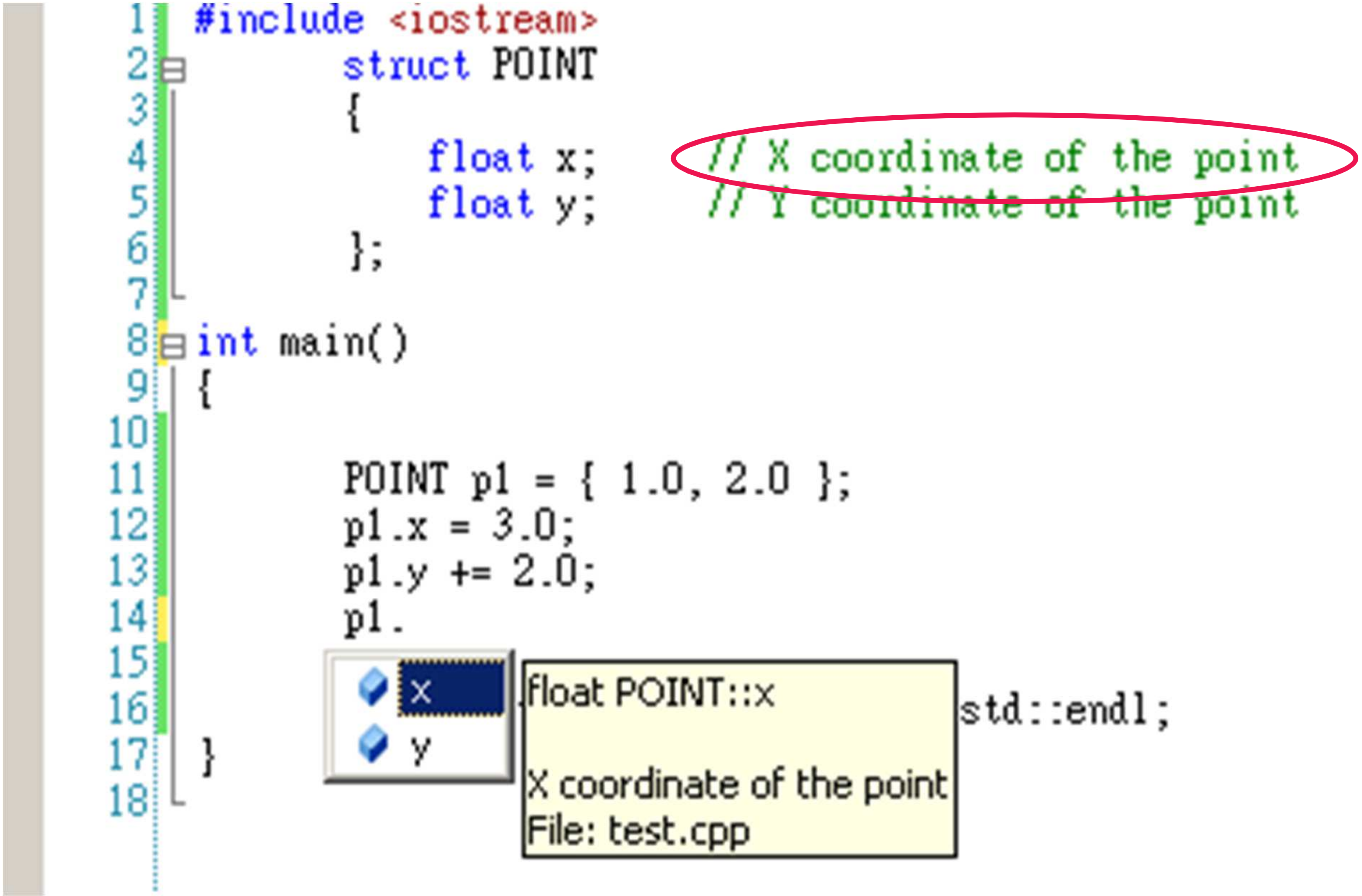
Pass by Reference (2)

```
void MoveRect(RECTANGLE& aRect, int x, int y)
{
  int length(aRect.Right - aRect.Left);     // Get length of rectangle
  int width(aRect.Bottom - aRect.Top);      // Get width of rectangle

  aRect.Left = x;                           // Set top-left point
  aRect.Top = y;                            // to new position
  aRect.Right = x + length;                 // Get bottom-right point as
  aRect.Bottom = y + width;                 // increment from new position
  return;
}
```

- Because aRect is passed as a reference, the function is able to modify the members of the RECTNAGLE directly.

Intellisense Assistance with Structures



```
#include <iostream>
        struct POINT
        {
            float x;      // X coordinate of the point
            float y;      // Y coordinate of the point
        };

int main()
{

        POINT p1 = { 1.0, 2.0 };
        p1.x = 3.0;
        p1.y += 2.0;
        p1.

                                          std::endl;
}

```

float POINT::x

X coordinate of the point
File: test.cpp

The struct RECT

- There is a pre-defined structure RECT in the header file `windows.h`, because rectangles are heavily used in Windows programs.

```
struct RECT
{
  int left;           // Top left point
  int top;            // coordinate pair

  int right;          // Bottom right point
  int bottom;         // coordinate pair
};
```

Using Pointers with a struct

- `RECT* pRect = NULL;`
  - Define a pointer to RECT

- `RECT aRect;`
- `pRect = &aRect;`
  - Set pointer to the address of aRect

Accessing Structure Members through a Pointer

- `RECT aRect = { 0, 0, 100, 100};`
- `RECT* pRect = &aRect;`

- `(*pRect).Top += 10;`
  - The parenthesis to de-reference the pointer are necessary (P.77)

- `pRect->Top += 10;`
  - Indirect member selection operator

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

struct tm

- struct tm

- gmtime() - Convert time_t to tm as UTC time
- localtime() - Convert time_t to tm as local time
- mktime() - Convert tm structure to time_t
- asctime() - Convert tm structure to string
- strftime() - Format time as string

struct tm

- The structure contains 9 members of type int:

```
struct tm {
    int tm_sec;     /* seconds after the minute - [0,59] */
    int tm_min;     /* minutes after the hour - [0,59] */
    int tm_hour;    /* hours since midnight - [0,23] */
    int tm_mday;    /* day of the month - [1,31] */
    int tm_mon;     /* months since January - [0,11] */
    int tm_year;    /* years since 1900 */
    int tm_wday;    /* days since Sunday - [0,6] */
    int tm_yday;    /* days since January 1 - [0,365] */
    int tm_isdst;   /* daylight savings time flag */
    };
```

localtime() and asctime() example

#include <ctime>
struct tm * localtime (const time_t * timer);

#include <ctime>
char * asctime (const struct tm * timeptr);

```
#include <iostream>
#include <ctime>

int main ()
{
  time_t rawtime;
  struct tm * timeinfo;

  time ( &rawtime );
  timeinfo = localtime ( &rawtime );
  std::cout << "It is Year "
           << timeinfo->tm_year + 1900 << std::endl;
  // years since 1900
  std::cout <<"Current local time and date: "
  << asctime (timeinfo) << std::endl;

  return 0;
}
```

strftime()

#include <ctime>
size_t strftime (char * buffer, size_t maxsize, const char * format, const struct tm * timeptr);

```
#include <iostream>
#include <ctime>

int main ()
{
  time_t rawtime;
  struct tm * timeinfo;
  char buffer [80];

  time ( &rawtime );
  timeinfo = localtime ( &rawtime );

  strftime (buffer,80,"Now it's %I:%M%p.",timeinfo);
  std::cout << buffer << std::endl;

  return 0;
}
```

In addition to the default format like
Thu Dec 27 21:31:04 2012
you may define your own format to display the time.

A struct can contain a pointer

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

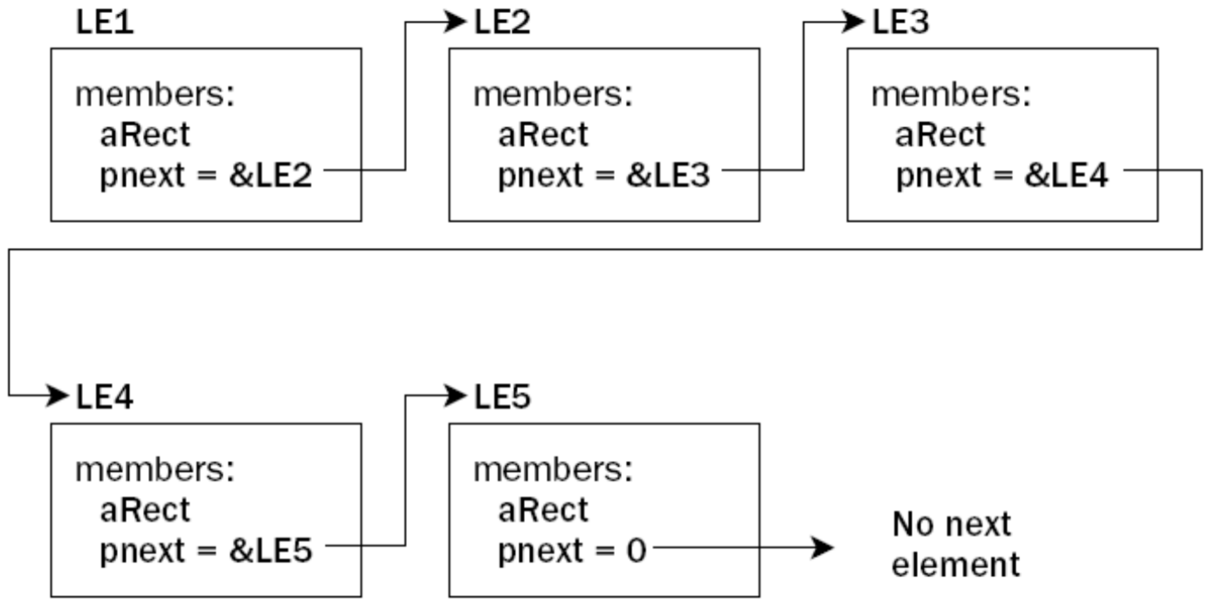


Figure 7-3 Linked List

Dynamic Memory Allocation (P.201)

- 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];
```

error C2057: expected constant expression

Why Use Pointers? (P.183)

- Use pointer notation to operate on data stored in an array
- Enable access within a function to arrays, that are defined outside the function
- 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

Exercise to Upload

1. Based on Ex7_01.cpp, write a function EqualAreaRect() which compares the area of two rectangles. In your main(), use at least two test cases to demonstrate that your function is working fine.
2. 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.

Sort an array of rational numbers

- Modify your own bubble sort function to sort an array of rational numbers.
- Suppose you defined a structure

```
struct Q {
    int q;
    int p;
    };
```

- a function to display the array

```
void print_array(Q a[], int n)
{
    for (int i=0; i<n; i++)
        cout << a[i].q << '/' << a[i].p << ' ';
    cout << endl;
}
```

- and a main program to test it.

```
int main()
{
    Q a[] = { {7,3}, {1,5}, {6, 5}, {4, 3} };
    int size = sizeof(a) / sizeof(a[0]);
    print_array(a, size);
    bsort(a, size, cmp);
    print_array(a, size);
    return 0;
}
```

The output should be
7/3 1/5 6/5 4/3
1/5 6/5 4/3 7/3

- Now all you need to supply is a cmp() function and a revised bsort() function.

Homework

- Write a program so that when the user input a number n, it will generate an array with n*n rational numbers, and sort the array.
- Use your own bubble sort function to sort the array, and measure the elapsed time by time().

100	2s
200	32s
300	167s

- Compare the result with the qsort() function.
- The default stack size is 1MB.
  - Properties - Configure Properties - Linker - System - Stack Reserve Size