1. **Structure:**
   Compound data type that is composed of two or more related members. Unlike arrays, members can be of different types.

2. **Structure Template:**
   Blue print (plan) describing the members of the structure
   ```
   struct <structure tag> // lines 7 - 12 in structIntro
   {
       member 1;
       member 2;
       ...
   }; // End of template declaration
   ```
   **Tag:** the name of the compound type

   Members can be any C++ data type, including arrays and other structures.

   **External template:** outside of any function; structure tag can be used anywhere in file from that point on

   **Local template:** inside of a function; structure tag can be used only within that function

3. **Declaring a Structure Variable:**

   Template is the plan, and alone does not declare a structure variable or allocate memory.

   **Syntax:**
   ```
   struct tag    structure variable;
   Inventory    car_part; // line 20
   ```

   Similar to:
   ```
   int x;
   ```

   where **tag** plays same role as **int**

   Contiguous bytes are allocated to a structure variable.

   Whether or not a structure variable is external depends on where the variable is declared, not where the template is declared.
4. Initialization:

Default initializations are the same as other variables:
zero for external structures
garbage for local structures

Can include a comma-separated list of initializers enclosed in braces.

Partially-initialized structure: members that are not explicitly initialized are set to 0.

5. Accessing a Structure Member:

Syntax:
<structure variable>.<member variable>

    car_part.description  // line 23

. is member operator (dot operator)

    Left to Right association

6. Declaring a template and a structure variable can be combined.

    struct
    {    char pet_name[30];
    int pet_code;
    float pet_cost;
    } cat, dog, bird;  // Structure variables

    Structure tag is optional; cannot declare additional variables without a tag

7. On some systems, the sizeof a structure may be greater than the sum of its parts. These system align members on even addresses or addresses that are multiples of 4. Such structures may have "holes" in them.
// structIntro: Introduction to Structures

#include <iostream> // cout

using namespace std; // Structure template

struct Inventory
{
    char description[30]; // 3 structure members
    double unit_price;
    int quan_on_hand;
}; // Note semicolon

/* No memory is allocated as a result of declaring the structure template. Since the template is declared externally, variables of type Inventory can be declared throughout the source file. */

int main()
{
    Inventory car_part = { "Water Pump Kit", 42.99, 36 }; // car_part is structure variable of type Inventory

    cout << "Inventory Report for Car Part"
    << "\nPart description: " << car_part.description
    << "\nPrice for one part: " << car_part.unit_price
    << "\nQuantity on hand: " << car_part.quan_on_hand << "\n\n";

    Inventory school_supply; // school_supply is structure variable

    school_supply = car_part; // Memberwise assignment

    cout << "Inventory Report for School Supply"
    << "\nPart description: " << school_supply.description
    << "\nPrice for one part: " << school_supply.unit_price
    << "\nQuantity on hand: " << school_supply.quan_on_hand << endl;

    return 0;
}

Output:
Inventory Report for Car Part
Part description: Water Pump Kit
Price for one part: 42.99
Quantity on hand: 36

Inventory Report for School Supply
Part description: Water Pump Kit
Price for one part: 42.99
Quantity on hand: 36
49  Image of structure variables:
50  
51  car_part
52  
53  
54  school_supply
55
// structArray: Array of Structures

#include <iostream> // cin, cout, getline(), get()
#include <iomanip> // setprecision()

using namespace std;

const int MAXPARTS = 6;

struct Inventory {
    char description[30];
    double unit_price;
    int quan_on_hand;
};

int main() {
    Inventory car_part[MAXPARTS];

    cout << "Enter data for car parts.\n"
        << "Enter part description (<Enter> to stop): ";

    int ct = 0;

    while (ct < MAXPARTS &&
           cin.getline(car_part[ct].description, 30) &&
           car_part[ct].description[0])
    {
        cout << "Enter unit price: ";
        cin >> car_part[ct].unit_price;
        cout << "Enter quantity on hand: ";
        cin >> car_part[ct++].quan_on_hand; // Note ct++
        cin.get(); // Flush the newline
        if (ct < MAXPARTS)
            cout << "\nEnter description of next part (<Enter> to stop): ";
    }

    cout << fixed << setprecision(2)
        << "\nINVENTORY OF CAR PARTS\n";
    for (int i = 0; i < ct; ++i)
    {
        cout << "Part description: " << car_part[i].description
            << "\nPrice for one part: " << car_part[i].unit_price
            << "\nQuantity on hand: " << car_part[i].quan_on_hand
            << "\n\n";
    }
    return 0;
}
Output:
50 Enter data for car parts.
51 Enter part description (<Enter> to stop): Water Pump Kit
52 Enter unit price: 42.99
53 Enter quantity on hand: 36
54
55 Enter description of next part (<Enter> to stop): Rubber Thermo Mount
56 Enter unit price: 2.58
57 Enter quantity on hand: 30
58
59 Enter description of next part (<Enter> to stop): Hose Water Upper
60 Enter unit price: 24.07
61 Enter quantity on hand: 20
62
63 Enter description of next part (<Enter> to stop):
64
65 INVENTORY OF CAR PARTS
66 Part description: Water Pump Kit
67 Price for one part: 42.99
68 Quantity on hand: 36
69
70 Part description: Rubber Thermo Mount
71 Price for one part: 2.58
72 Quantity on hand: 30
73
74 Part description: Hose Water Upper
75 Price for one part: 24.07
76 Quantity on hand: 20
77
78
79

Array Image:
80 car_part[0] . . . car_part[5]
81
82 Each element is type Inventory.
Details of the first three elements:

<table>
<thead>
<tr>
<th>car_part[0]</th>
<th>description</th>
<th>Water Pump Kit</th>
<th>unit_price</th>
<th>42.99</th>
<th>quan_on_hand</th>
<th>36</th>
</tr>
</thead>
</table>

| car_part[1] | description | Rubber Thermo Mount | unit_price | 2.58 | quan_on_hand | 30 |

// structByValue: Passing a structure by value

#include <iostream>  // cout
#include <iomanip>  // setprecision()

using namespace std;

const int MAXPARTS = 2;

struct Inventory
{
    char description[30];
    double unit_price;
    int quan_on_hand;
};

double total_value(Inventory auto_part);  // By value

int main()
{
    Inventory car_part[MAXPARTS] =
    {
        { "Water Pump Kit", 42.99, 36 },  // Inner braces are not necessary
        { "Rubber Thermo Mount", 2.58, 30 }
    };

cout << fixed << setprecision(2)
<< "INVENTORY OF CAR PARTS\n"
for (int i = 0; i < MAXPARTS; ++i)
{
    cout << "Part description: " << car_part[i].description
    << "\nPrice for one part: " << car_part[i].unit_price
    << "\nQuantity on hand: " << car_part[i].quan_on_hand
    << "\nTotal value: " << total_value(car_part[i])
    << "\n\n";
}
return 0;
}

double total_value(Inventory auto_part)
{
    return auto_part.unit_price * auto_part.quan_on_hand;
}

Output:
INVENTORY OF CAR PARTS
Part description: Water Pump Kit
Price for one part: 42.99
Quantity on hand: 36
Total value: 1547.64

Part description: Rubber Thermo Mount
Price for one part: 2.58
Quantity on hand: 30
Total value: 77.40
Stack Image:

<table>
<thead>
<tr>
<th>copy of car_part[i]</th>
<th>auto_part</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td></td>
</tr>
<tr>
<td>car_part[0]</td>
<td></td>
</tr>
<tr>
<td>car_part[1]</td>
<td></td>
</tr>
</tbody>
</table>

main()

Details of the elements are shown below.

car_part[0]

<table>
<thead>
<tr>
<th>description</th>
<th>Water Pump Kit</th>
</tr>
</thead>
<tbody>
<tr>
<td>unit_price</td>
<td>42.99</td>
</tr>
<tr>
<td>quan_on_hand</td>
<td>36</td>
</tr>
</tbody>
</table>

car_part[1]

<table>
<thead>
<tr>
<th>description</th>
<th>Rubber Thermo Mount</th>
</tr>
</thead>
<tbody>
<tr>
<td>unit_price</td>
<td>2.58</td>
</tr>
<tr>
<td>quan_on_hand</td>
<td>30</td>
</tr>
</tbody>
</table>
// structByRef: Passing a structure by reference

#include <iostream> // cout
#include <iomanip> // setprecision()

using namespace std;

const int MAXPARTS = 2;

struct Inventory
{
    char description[30];
    double unit_price;
    int quan_on_hand;
};

double total_value(const Inventory& auto_part); // By reference

int main()
{
    // car_part is array of Inventory structures
    Inventory car_part[MAXPARTS] =
    {
    { "Water Pump Kit", 42.99, 36 },
    { "Rubber Thermo Mount", 2.58, 30 }
    }

    cout << fixed << setprecision(2)
    << "INVENTORY OF CAR PARTS"
    for (int i = 0; i < MAXPARTS; ++i)
    {
        cout << "Part description: " << car_part[i].description
        << "\nPrice for one part: " << car_part[i].unit_price
        << "\nQuantity on hand: " << car_part[i].quan_on_hand
        << "\nTotal value: " << total_value(car_part[i])
        << "\n";
    }
    return 0;
}

double total_value(const Inventory& auto_part)
{
    return auto_part.unit_price * auto_part.quan_on_hand;
}

Output:
INVENTORY OF CAR PARTS
Part description: Water Pump Kit
Price for one part: 42.99
Quantity on hand: 36
Total value: 1547.64

Part description: Rubber Thermo Mount
Price for one part: 2.58
Quantity on hand: 30
Total value: 77.40
Details of the elements are shown below.

**car_part[0]**
- **description**: Water Pump Kit
- **unit_price**: 42.99
- **quan_on_hand**: 36

**car_part[1]**
- **description**: Rubber Thermo Mount
- **unit_price**: 2.58
- **quan_on_hand**: 30
// structByPtr: Passing a structure using a pointer

#include <iostream>   // cout
#include <iomanip>    // setprecision()

using namespace std;

const int MAXPARTS = 2;

struct Inventory
{
    char description[30];
    double unit_price;
    int quan_on_hand;
};

double total_value(const Inventory* pcInv); // Pointer to const Inventory

int main()
{
    // car_part is array of Inventory structures
    Inventory car_part[MAXPARTS] =
    {
        { "Water Pump Kit", 42.99, 36 },
        { "Rubber Thermo Mount", 2.58, 30 }
    };

    cout << fixed << setprecision(2) << "INVENTORY OF CAR PARTS
";
    for (int i = 0; i < MAXPARTS; ++i)
    {
        cout << "Part description: " << car_part[i].description << 
            "\nPrice for one part: " << car_part[i].unit_price << 
            "\nQuantity on hand: " << car_part[i].quan_on_hand << 
            "\nTotal value: " << total_value(&car_part[i]) << 
            "\n\n"; // or total_value(car_part + i)
    }
    return 0;
}

double total_value(const Inventory* pcInv)
{
    // Indirect member operator
    return pcInv->unit_price * pcInv->quan_on_hand;
}

Output:
INVENTORY OF CAR PARTS
Part description: Water Pump Kit
Price for one part: 42.99
Quantity on hand: 36
Total value: 1547.64

Part description: Rubber Thermo Mount
Price for one part: 2.58
Quantity on hand: 30
Total value: 77.40
Stack Image:

```
<table>
<thead>
<tr>
<th>i</th>
<th>pcInv</th>
<th>total_value()</th>
</tr>
</thead>
<tbody>
<tr>
<td>car_part[0]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>car_part[1]</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

main()

Details of the elements are shown below.

**car_part[0]**

- **description**: Water Pump Kit
- **unit_price**: 42.99
- **quan_on_hand**: 36

**car_part[1]**

- **description**: Rubber Thermo Mount
- **unit_price**: 2.58
- **quan_on_hand**: 30

**Note:**

Direct member operator: structure variable . member variable

Indirect member operator: pointer to structure -> member variable
// structMoney: Returning a structure

#include <iostream> // cout
#include <iomanip> // setw(), setprecision()

using namespace std;

const int CENTS_PER_DOLLAR = 100;

struct Money
{
    int dollars;
    int cents;
};

Money sum(Money m1, Money m2); // Returns type Money
double total_value(Money m);

int main()
{
    Money pay = {1400, 85}, overtime = {351, 63};

    cout << fixed << setprecision(2)
        << "Regular pay: " << setw(8) << total_value(pay)
        << "\nOvertime pay: " << setw(8) << total_value(overtime)
        << "\nTotal pay: " << setw(8) << total_value(sum(pay, overtime))
        << endl;

    return 0;
}

Money sum(Money m1, Money m2)
{
    Money temp;
    temp.cents = (m1.cents + m2.cents) % CENTS_PER_DOLLAR;
    temp.dollars = m1.dollars + m2.dollars +
        (m1.cents + m2.cents) / CENTS_PER_DOLLAR;
    return temp;
}

double total_value(Money m)
{
    return m.dollars + 0.01 * m.cents;
}

Output:
Regular pay: 1400.85
Overtime pay: 351.63
Total pay: 1752.48
Images of structure variables:

<table>
<thead>
<tr>
<th>pay</th>
<th>overtime</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Stack during total_value() at line 24:

```
Copy of pay  m
            total_value()
    overtime
    pay
main()
```

Stack during sum():

```
            temp
Copy of pay  m1
Copy of overtime  m2
            sum()
    overtime
    pay
main()
```

To pass structures by reference:

```
32 Money sum(Money& m1, Money& m2)    // m1 and m2 become aliases
42 double total_value(Money& m)       // m becomes an alias
```

Lines 16 and 17: same changes

References were introduced primarily for use with structures and objects.

Would `const Money&` work?
// SellParts: Reference return type

#include <iostream>       // cout
#include <iomanip>        // setprecision()

using namespace std;

struct Inventory
{
    char description[30];
    double unit_price;
    int quan_on_hand;
};

Inventory& sell_part(Inventory& auto_part); // Returns a Inventory reference

int main()
{
    Inventory pump = { "Water Pump Kit", 42.99, 2 },
                    mount = { "Rubber Thermo Mount", 2.58, 30 };

    cout << fixed << setprecision(2);
    sell_part(pump); // Nothing is done with return value
    sell_part(sell_part(pump));
    sell_part(pump) = mount; // Structure memberwise assignment
    sell_part(pump);
    sell_part(mount);

    return 0;
}

Inventory& sell_part(Inventory& auto_part)
{
    if (auto_part.quan_on_hand)
    {
        --auto_part.quan_on_hand;
        cout << "Selling 1 " << auto_part.description << " for $" << auto_part.unit_price << " leaving " << auto_part.quan_on_hand << " part(s)\n";
    }
    else
    {
        cout << "Out of parts. Reorder: " << auto_part.description << "\n";
        return auto_part;
    }

Output:
Selling 1 Water Pump Kit for $42.99 leaving 1 part(s)  (23)
Selling 1 Water Pump Kit for $42.99 leaving 0 part(s)  (25)
Out of parts. Reorder: Water Pump Kit                (25)
Out of parts. Reorder: Water Pump Kit                (27)
Selling 1 Rubber Thermo Mount for $2.58 leaving 29 part(s) (29)
Selling 1 Rubber Thermo Mount for $2.58 leaving 29 part(s) (31)
Stack during `sell_part()` at line 23:

```
main()
  mount
  pump
    sell_part()
  auto_part
```

Returning a reference:
Normally, C++ returns a copy of the return value, called a temporary, to the caller. Returning a reference turns off this copying mechanism and returns the actual object. Typically, the reference refers to a reference argument. Thus, the caller accesses one of its local variables.

You cannot return a local variable by reference. In `structMoney`, `sum()` could not return `Money&`.

Details of pump and mount are shown below.

**pump**

- **description**: Water Pump Kit
- **unit_price**: 42.99
- **quan_on_hand**: 2

**mount**

- **description**: Rubber Thermo Mount
- **unit_price**: 2.58
- **quan_on_hand**: 30
1. **struct Pal**
   
   ```
   char name[30];
   char phone[13];
   ```

   ... 

   ```
   Pal chum[30];    // static array of 30 Pal types
   ```

   Image of `chum` array:

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>...</th>
<th>29</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name</td>
<td>name</td>
<td>...</td>
<td>phone</td>
</tr>
<tr>
<td>James</td>
<td>Pat</td>
<td>Yuri</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>phone</td>
<td>phone</td>
<td>phone</td>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

   Disadvantages of the array:
   a. Don't always knows the size of the array or size may need to vary
      
      Could use a dynamic array, but
   
   b. Inserting new chum (Robin) is not easy
   
   c. Deleting unwanted chum (Pat) is not easy

2. **Linked List**:
   Data organization consisting of a list of nodes (structures);
   
   Each node contains a pointer (a link) to the next node in the list;
   new member:  **Pal* next**;
   
   Last node in the list, the **Tail** of the list, has a link of 0 (null pointer);
   
   **Head** points to the First node in the list.
Image of the Linked List of Pal types:

Note the new structure template:

```c
struct Pal
{
    char name[30];
    char phone[13];
    Pal* next;
};
```
// LinkedList: Linked list of structures

#include <iostream>    // getline(), cout
#include <cstdlib>     // exit()

using namespace std;

const int NAME_SIZE = 30;
const int PHONE_SIZE = 13;

struct Pal
{
    char name[NAME_SIZE];
    char phone[PHONE_SIZE];
    Pal* next;          // link to next Pal
};

void create_list(Pal** phead);
void print_list(const Pal* pPal);
void delete_list(Pal** phead);

int main()
{
    Pal* head = 0;

    create_list(&head);
    print_list(head);
    delete_list(&head);
    return 0;
}

void create_list(Pal** phead)
{
    Pal* prev;
    Pal* curr;

    for (int i = 1; i <= 3; i++)  // Three pals
    {
        curr = new Pal;        // Allocate memory for a Pal node
        if (curr)
        {
            cout << "Enter pal's name";  // If new was successful
            cin.getline(curr->name, NAME_SIZE);
            cout << "Enter pal's phone number";
            cin.getline(curr->phone, PHONE_SIZE);
            curr->next = 0;

            if (!*phead) // or if (*phead == 0)
                *phead = curr;  // First pal
            else
                prev->next = curr;

            prev = curr;
        }
    }
}

// If new failed
{
cerr << "Not able to allocate memory for a pal.\n";
exit(EXIT_FAILURE);
} // End of if
// End of for loop
void print_list(const Pal* pPal) // Traverse Linked List
{
    cout << "Here are your pals:\n";
    while (pPal) // or while (pPal != 0)
    {
        cout << pPal->name << " can be reached at " << pPal->phone << "\n";
pPal = pPal->next;
    }
}
void delete_list(Pal** phead)
{
    Pal* prev;
Pali curr;
curr = *phead; // Set curr to point to first node in list
*phead = 0; // Set the head to 0
while(curr) // While curr node exists
{
    prev = curr;
curr = curr->next;
delete prev;
}

Output:
Enter pal's name=========> James
Enter pal's phone number==> 415-333-1234
Enter pal's name=========> Pat
Enter pal's phone number==> 510-231-8844
Enter pal's name=========> Yuri
Enter pal's phone number==> 510-222-9876
Here are your pals:
James can be reached at 415-333-1234
Pat can be reached at 510-231-8844
Yuri can be reached at 510-222-9876
Press any key to continue
Stack during creat_list():

Free Store: