Structures
Structures

Structure

Structure Declaration

يتتم الإعلان عن الهيكل في برنامج الـ سي بالصيغة التالية:

```
struct name
{
    ......................;
    ......................;
    ......................;
}
```

اسم الهيكل

كلمة من كلمات اللغة الأساسية

يجب وضع: لانهاء الهيكل

عناصر البيانات التي يتكون منها الهيكل
إذا أردنا أن نقوم بعمل هيكل ليحمل بيانات الموظفين في الشركة، حيث نحتاج إلى تخزين اسم الموظف وعنوانه، وعمر الموظف، وراتبه، وطبقا لهذه المعلومات يكون الهيكل كما يلي:

```c
struct employee
{
    char name[40];
    char address[50];
    int age;
    float salary;
};
```

وعندما نريد استخدام الهيكل في البرنامج، يجب أن يتم الإعلان عن المتغير من نفس نوع الهيكل بالصيغة التالية:

```c
struct name variable_name;
```

مثال:

```c
struct employee emp1;
```

```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
A structured data type is one in which each value is a collection of components and whose organization is characterized by the method used to access individual components.
A record or a struct in C++, is a structured data type with a fixed number of components that are accessed by name. The components may be heterogeneous.

A field or component is a piece of the record.
struct TypeName
{
    DataType MemberName;
    DataType MemberName;
    ...
};
struct Student
{
    Int Id;
    char firstName[20];
    char lastName[30];
    float gpa;
    int programGrade;
};
ويمكن الإعلان عن عدة متغيرات من نفس الهيكل:

```c
Struct Employee
{char name[40];
 char address[50];
 int age;
 float salary;
} emp1, emp2, emp3;
```

وإذا كنا لنعلن عن هيكل آخر في البرنامج، يمكن الاستغناء عن اسم الهيكل:

```c
struct
{
 char name[40];
 char address[50];
 int age;
 float salary;
} emp1, emp2, emp3;
```
struct TypeName
{
    MemberList
};

struct StudentRec
{
    ...

} Bill, Mary, Susie, Phil;
Structures are C’s way of grouping collections of data into a single manageable unit.

- This is also the fundamental element of C upon which most of C++ is built (i.e., classes).
- Similar to Java's classes.

An example:

- Defining a structure type:

  ```
  struct coord {
    int x ;
    int y ;
  };
  ```

  - This defines a new type struct coord. No variable is actually declared or generated.
Define struct variables:

```c
struct coord {
    int x,y ;
} first, second;
```

Another Approach:

```c
struct coord {
    int x,y ;
};
```

............

```c
struct coord first, second; /* declare variables */
```

```c
struct coord third;
```
You can even use a typedef if you don't like having to use the word "struct"

typedef struct coord coordinate;
coordinate first, second;

In some compilers, and all C++ compilers, you can usually simply say just:
coord first, second;
Access structure variables by the dot (.) operator

- Generic form:
  
  \[ \text{structure} \_\text{var}.\text{member}\_\text{name} \]

- For example:
  
  \[
  \begin{align*}
  \text{first} \_\text{x} &= 50; \\
  \text{second} \_\text{y} &= 100;
  \end{align*}
  \]

- These member names are like the public data members of a class in Java (or C++).
  - No equivalent to function members/methods.

- \( \text{struct} \_\text{var}.\text{member}\_\text{name} \) can be used anywhere a variable can be used:
  - \( \text{printf} \("\%d, \%d"\, \text{second} \_\text{x}, \, \text{second} \_\text{y}\); \)
  - \( \text{scanf} \("\%d, \%d"\, &\text{first} \_\text{x}, \, &\text{first} \_\text{y}\); \)
You can assign structures as a unit with:

```plaintext
first = second;
```

instead of writing:

```plaintext
first.x = second.x;
first.y = second.y;
```

Although the saving here is not great
- It will reduce the likelihood of errors and
- Is more convenient with large structures

This is different from Java where variables are simply references to objects.

```plaintext
first = second;
```

makes first and second refer to the same object.
#include<stdio.h>
#include<iostream.h>

// الاعلان عن الهيكل أو التركيب

struct employee
{
char name[40];
char address[50];
int age;
float salary;
};

main()
{
struct employee emp1; // الاعلان عن المتغير
cout<<"Enter name: ";
gets(emp1.name);
cout<<"Enter address: ";
gets(emp1.address);
cout<<"Enter age: ";
cin>> emp1.age;
cout<<"Enter salary: ";
cin>> emp1.salary;
cout<<\n"The information is: ";
cout<<emp1.name<<endl;
cout<<emp1.address<<endl;
cout<<emp1.age<<endl;
cout<<emp1.salary<<endl;
return 0;
}
Struct employee
{char name[40];
 char address[50];
 int age;
 float salary;
};
Struct employee emp1[100];

#include<stdio.h>
#include<iostream.h>

// الإعلان عن الهيكل أو التركيب
Struct employee
{char name[40];
 char address[50];
 int age;
 float salary;
}emp = { "sayed Ali ",
 "16 Hassan Raef St." 43, 1200 }

main()
{cout<<emp.name<<endl;
cout<<emp.address<<endl;
cout<<emp.age<<endl;
```c
#include<stdio.h>
#include<iostream.h>

struct employee
{
    char name[40];
    char address[50];
    int age;
    float salary;
};

struct depart
{
    int dept_no;
    char product[40];
    struct employee emp;
};

void main(void)
{
    struct depart d1;
    cout<<"Enter department No:"
    cin>>d1.dept_no;
    cout<<"Enter dept product:"
    gets(d1.product);
    cout<<"Enter employee address:"
    gets(d1.emp.address);
    cout<<"Enter employee age:"
```
```c
#include<stdio.h>
#include<iostream.h>

struct student
{
    char name[50];
    int age;
    int grade;
};

void print_struct (struct student s);

Main()
{
    struct student s;
    cout<<"Enter student name :"<<endl;
    gets(s.name);
    cout<<"\n Enter student age:";
    cin>>s.age;
    cout<<"\n Enter student grade:";
    cin>>s.grade;
    print_struct(s); // استدعاء الدالة لطبع محتويات الاري
}

void print_struct ( struct student s)
{
    cout<< s.name << endl;
    cout<< s.age <<endl;
    cout<< s.grade;
}
الاتحاد

استخدم الاتحاد في وضع عدة أنواع من البيانات المختلفة في مجموعة واحدة بحيث نتمكن من التعامل مع المجموعة ككيان واحد أو نتعامل مع كل عنصر فيها على حدة. ويعلن عن الاتحاد بالصيغة التالية:

```
union name
{
    ......................;
    ......................;
};
```

كما يتم الإعلان عن المتغيرات من نوع الاتحاد كما يلي:

```
union name variable_name;
```

والاتحاد يشبه إلى حد كبير الهيكل structure ولكن الفرق بينهما في حزج الذاكرة للبيانات، حيث يقوم المترجم في الاتحاد بحجز ذاكرة تكفي لتخزين أكبر عنصر في المجموعة فقط، وتستخدم نفس المساحة للتعامل مع بقية العناصر الموجودة في المجموعة.
Unions

- Unions allow you to use the same memory space for several different types of data... 
- not at the same time, but when you need to.

```c
union WeightType
{
    long wtInTons;
    int wtInPounds;
    float wtInOunces;
};
```
```cpp
// Example of enumeration
#include <iostream.h>

// Define days of the week as an enum
enum week_days { sun, mon, tue, wed, thu, fri, sat }

int main()
{
    week_days day1, day2;
    int diff;
    day1 = mon;
    day2 = thu;
    diff = day2 - day1;
    cout << "days between = " << diff << endl;
    if(day1 < day2)
        cout << " day1 comes before day2 \n " ;
    return 0;
}
```
1. أول عنصر من عناصر البيانات التعدادية يأخذ الرقم 0 كقيمة صحيحة، والعنصر التالي يأخذ القيمة 1، وهكذا... لبقية العناصر. كما يمكن التحكم في هذه القيم على أن تبدأ كما نريد كما يلي:

```cpp
enum suit { clubs = 1, diamonds, hearts, spades };
```

في هذا المثال حددنا أول عنصر `clubs` سوف يأخذ القيمة 1، وبالتالي العنصر التالي سوف يأخذ القيمة 2 وهكذا.

2. لا يمكن إدخال أو طباعة المتغيرات من النوع التعدادي، بينما يطبق عليها عمليات المعالجة الأخرى مثل التكليف أو المقارنة، كما يلي:

```cpp
enum direction { north, south, east, west };
direction dir1 = south;
cout<< dir1 ;
```

في هذا المثال الإخراج لن يكون كلمة `south` التعدادية لهذه الكلمة، بعكس أن تتم تمثيل القيمة `south` بالعدد 1 لأنها تمثل القيمة.

أمثلة على أنواع البيانات التعدادية:

```cpp
enum months { Jan, Feb, Mar, Apr, May, Jun,
```
enum colors { Red, Green, Blue, Yellow, Cyan, Magenta, White, Black };

enum meridian { am, pm };

to the time zone:

enum switch { off, on };

status of the device:

enum coins { penny, nickel, dime, quarter, half_dollar, dolar };

several jewels:

enum chess { pawn, knight, bishop, rook, Queen, king };

7. Models of cars on display:

enum cars { Fiat, Toyota, Ford, Mazda };

It can be used for the enumeration type to define data of the


do as follows:

enum Boolean { false, true };

1. عرف نوع البيانات الهيكل: S_temp

لكلا من السجلات التالية:

| أ | سجل الطالب ويكون من رقم تسجيل الطالب، عدد ساعات الدراسة، ومتوسط درجات الطالب. |

```
struct S_temp
{
    int id_num;
    int credits;
    float avg;
};
```

| ب | سجل الطالب ويكون من اسم الطالب، تاريخ ميلاد الطالب، عدد الساعات الدراسية، ومتوسط درجات الطالب. |

```
struct S_temp
{
    char name [50];
    int month;
    int day;
    int year;
    int credits;
    float avg;
};
```
#include <iostream.h>

// define the structure
struct date
{
    int month;
    int day;
    int year;
}

int main()
{
    // define the structure variable.....
    struct date current;
    cout << "\n Enter the current month " ;
    cin >> current.month ;
    cout << " \n Enter the current day : " ;
    cin >> current.day ;
    cout << " \n Enter the current year : " ;
    cin >> current.year ;
    cout << " \n the date entered is : " << current.month << '/' << current.day
          << '/' << current.year << endl ;

    return 0 ;
}
#include <cstdlib>
#include <iostream>
using namespace std;

void odd (int a);
void even (int a);

int main ()
{
  int i;
  do {
    cout << "Type a number (0 to exit): ";
    cin >> i;
    odd (i);
  } while (i!=0);
  system("PAUSE");
  return EXIT_SUCCESS;
}

void odd (int a)
{
  if ((a%2)!=0) cout << "Number is odd.\n";
  else even (a); }

void even (int a)
{
  if ((a%2)==0) cout << "Number is even.\n"; else odd (a); }
Structured data type: a collection of components whose organization is characterized by method used to access individual components.

Examples of structured types in C++

array ________________

struct ________________

union ________________

class ________________
TypeDefs

◆ Typedef statements allow you to introduce a new name for an existing type.
◆ In my opinion, it is most useful for software engineering type reasons.
◆ It follows this pattern:
  – `typedef ExistingTypeName NewTypeName`
  – `typedef int Boolean;`
  – `Boolean dataOK;`
** Enums **

- An enumerated type is a user-defined data type whose domain is an ordered set of literal values expressed as identifiers.

```c
enum Days { SUN, MON, TUE, WED, THU, FRI, SAT };
enum Animals { RODENT, CAT, DOG, BIRD, REPTILE, HORSE, BOVINE };
Animals inPatient;
Animals outPatient;
```
Allowed Operations

- `inPatient = Dog;`
- `inPatient = Animals( inPatient + 1 );`
- `switch ( inPatient )`
- You can also declare variables of this new type at the time of declaration.
- `enum Months { JAN, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, DEC } birthMonth, firstMonth, lastMonth;`
Accessing a field

To assign a value to one of the fields, you use the member selector operator ‘.’

StudentRec Bill;
Bill.firstName = “Fred”;
Bill.lastName = “Williams”;
Bill.gpa=2.73;
Bill. programGrade=2;
## Aggregate Operations

<table>
<thead>
<tr>
<th>Aggregate Operation</th>
<th>Allowed on Structs?</th>
</tr>
</thead>
<tbody>
<tr>
<td>I/O</td>
<td>No</td>
</tr>
<tr>
<td>Assignment</td>
<td>Yes</td>
</tr>
<tr>
<td>Arithmetic</td>
<td>No</td>
</tr>
<tr>
<td>Comparison</td>
<td>No</td>
</tr>
<tr>
<td>Argument Passage</td>
<td>Yes, value and reference</td>
</tr>
<tr>
<td>Return from a function</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Assignment

- Just a quick word about assignment.
- The way it is copied is through member-wise assignment.
- Each field in the struct is simply matched up and copied.
- This becomes extremely important when you start dealing with “dynamic data”
Data abstraction is the separation of a data’s logical properties from its implementation.

An Abstract Data Type is a data type whose properties are specified independently of any particular implementation.

This leads us to Classes.
Structures Containing Structures

- Any “type” of thing can be a member of a structure.
- We can use the coord struct to define a rectangle
  
  ```
  struct rectangle {
    struct coord topleft;
    struct coord bottomrt;
  }
  ```

- This describes a rectangle by using the two points necessary:
  
  ```
  struct rectangle mybox ;
  ```

- Initializing the points:
  
  ```
  mybox.topleft.x = 0  ;
  mybox.topleft.y = 10  ;
  mybox.bottomrt.x = 100 ;
  mybox.bottomrt.y = 200  ;
  ```
```c
#include <stdio.h>

struct coord {
    int x;
    int y;
};

struct rectangle {
    struct coord topleft;
    struct coord bottomrt;
};

int main () {
    int length, width;
    long area;
    struct rectangle mybox;
    mybox.topleft.x = 0;
    mybox.topleft.y = 0;
    mybox.bottomrt.x = 100;
    mybox.bottomrt.y = 50;
    width = mybox.bottomrt.x – mybox.topleft.x;
    length = mybox.bottomrt.y – mybox.topleft.y;
    area = width * length;
    printf ("The area is %ld units.\n", area);
}
```
Arrays within structures are the same as any other member element.

For example:
```c
struct record {
    float  x;
    char  y [5] ;
} ;
```

Logical organization:
#include <iostream.h>

struct data {
    float amount;
    char fname[30];
    char lname[30];
} rec;

int main () {
    struct data rec;
    cout<<"Enter the donor's first and last names, \n";
    cout<<"separated by a space: ";
    cin>>rec.fname>> rec.lname;
    cout<<"\nEnter the donation amount: ";
    cin>>rec.amount;
    cout<<  rec.fname,rec.lname,rec.amount);
}
Arrays of Structures

◆ The converse of a structure with arrays:
◆ Example:

```c
struct entry {
    char fname [10] ;
    char lname [12] ;
    char phone [8] ;
}

struct entry list [1000];
```

◆ This creates a list of 1000 identical entry(s).
◆ Assignments:

```c
list [1] = list [6];
strcpy (list[1].phone, list[6].phone);
```
#include <stdio.h>

struct entry {
  char fname [20];
  char lname [20];
  char phone [10];
};

int main() {
  struct entry list[4];
  int i;
  for (i=0; i < 4; i++) {
    printf ("Enter first name: ");
    scanf ("%s", list[i].fname);
    printf ("Enter last name: ");
    scanf ("%s", list[i].lname);
    printf ("Enter phone in 123-4567 format: ");
    scanf ("%s", list[i].phone);
  }
  printf ("\n\n");
  for (i=0; i < 4; i++) {
    printf ("Name: %s %s", list[i].fname, list[i].lname);
    printf ("Phone: %s\n", list[i].phone);
  }
}"
Simple example:

```c
struct sale {
    char customer[20] ;
    char item[20] ;
    int amount ;
};

struct sale mysale = { "yehia elmashad", "CD Rom", 1000 } ;
```
Initializing Structures

Structures within structures:

```
struct customer {
    char firm [20] ;
    char contact [25] ;
};
struct sale {
    struct customer buyer ;
    char item [20] ;
    int amount ;
} mysale =
{ { “Delta Academy", " yehia elmashad "} ,
   “CDRom", 1000
};
```
Arrays of structures

```c
struct customer {
    char firm [20] ;
    char contact [25] ;
} ;

struct sale {
    struct customer buyer ;
    char item [20] ;
    int amount ;
} ;

struct sale y1990 [100] = {
    {  { "Delta Academy", "Yehia ElMashad"} ,
        "CD Rom" , 1000 },
    {  { "Wilson & Co.", "Ed Wilson"} ,
        "Thingamabob" , 290
    }
} ;
```
```c
void strcat(char dest[], char source[]); // see footnote

concatenates the character string source onto the end of the character string dest, taking proper care of the NULLs

#include <iostream.h>
#include <string.h> // for standard C-string functions

void main() {
    const int MAXLINE = 80;
    char a[MAXLINE] = "The quick brown fox";
    char b[MAXLINE] = " jumps over the lazy dog";
    strcat(a, b);
    cout << "a contains: " << a << endl;
}

produces the output:

a contains: The quick brown fox jumps over the lazy dog
```
int strlen(char source[]);

returns the length of the character string source, excluding the terminating NULL character

#include <iostream.h>
#include <string.h>

void main() {
    const int MAXLINE = 80;
    char a[MAXLINE] = "The quick brown fox";
    int length;
    length = strlen(a);
    cout << "The following string has " << length << " characters:" << endl;
    cout << a << endl;
}

produces the output:

The following string has 19 characters:  
The quick brown fox
The `strcpy()` function copies the character string `source` into the character string `dest`.

```c
#include <iostream.h>
#include <string.h>

void main() {
    const int MAXLINE = 80;
    char a[MAXLINE] = "The quick brown fox";
    char b[MAXLINE] = " jumps over the lazy dog";
    strcpy(a, b);
    cout << a << endl;
}
```

produces the output:

```
jumps over the lazy dog
```
int strcmp(char first[], char second[]);

compares the character strings first and second lexicographically, returning:

< 0      if first precedes second
  0      if first and second are the same
> 0      if first follows second

The comparison is based on the ASCII code sequence, so upper-case letters precede lower-case letters.
#include <iostream.h>
#include <string.h>

void main() {
    char a[] = "Frederick";
    char b[] = "Freddy";
    int k;
    k = strcmp(a, b);
    if (k < 0)
        cout << a << " before " << b << endl;
    else if (k == 0)
        cout << a << " equals " << b << endl;
    else // k > 0
        cout << b << " before " << a << endl;
}

produces the output: Freddy before Frederick
1. A structure brings together a group of
   a. items of the same data type.
   b. related data items.
   c. integers with user-defined names.
   d. variables.

2. True or false: A structure and a class use similar syntax.

3. The closing brace of a structure is followed by a _________.

4. Write a structure specification that includes three variables—all of type int—called hrs, mins, and secs. Call this structure time.

5. True or false: A structure definition creates space in memory for a variable.

6. When accessing a structure member, the identifier to the left of the dot operator is the name of
   a. a structure member.
   b. a structure tag.
   c. a structure variable.
   d. the keyword struct.

7. Write a statement that sets the hrs member of the time2 structure variable equal to 11.

8. If you have three variables defined to be of type struct time, and this structure contains three int members, how many bytes of memory do the variables use together?

9. Write a definition that initializes the members of time1—which is a variable of type struct time, as defined in Question 4—to hrs = 11, mins = 10, secs = 59.

10. True or false: You can assign one structure variable to another, provided they are of the same type.
1. b, d
2. true
3. semicolon
4.
    struct time
    {
        int hrs;
        int mins;
        int secs;
    };
5. false; only a variable definition creates space in memory
6. c
7. time2.hrs = 11;
8. 18 in 16-bit systems (3 structures times 3 integers times 2 bytes), or 36 in 32-bit systems
9. time time1 = { 11, 10, 59 };
10. true
11. temp = fido.dogs.paw;
12. c
13. enum players { B1, B2, SS, B3, RF, CF, LF, P, C };
14.
    players joe, tom;
    joe = LF;
    tom = P;
15. a. no
    b. yes
    c. no
    d. yes
16. 0, 1, 2
17. enum speeds { obsolete=78, single=45, album=33 };}
11. Write a statement that sets the variable temp equal to the paw member of the dogs member of the fido variable.

12. An enumeration brings together a group of
   a. items of different data types.
   b. related data variables.
   c. integers with user-defined names.
   d. constant values.

13. Write a statement that declares an enumeration called players with the values B1, B2, SS, B3, RF, CF, LF, P, and C.

14. Assuming the enum type players as declared in Question 13, define two variables joe and tom, and assign them the values LF and P, respectively.

15. Assuming the statements of Questions 13 and 14, state whether each of the following statements is legal.
   a. joe = QB;
   b. tom = SS;
   c. LF = tom;
   d. difference = joe - tom;

16. The first three enumerators of an enum type are normally represented by the values __________, __________, and __________.

17. Write a statement that declares an enumeration called speeds with the enumerators obsolete, single, and album. Give these three names the integer values 78, 45, and 33.
1. A phone number, such as (212) 767-8900, can be thought of as having three parts: the area code (212), the exchange (767), and the number (8900). Write a program that uses a structure to store these three parts of a phone number separately. Call the structure phone. Create two structure variables of type phone. Initialize one, and have the user input a number for the other one. Then display both numbers. The interchange might look like this:

Enter your area code, exchange, and number: 415 555 1212
My number is (212) 767-8900
Your number is (415) 555-1212

2. A point on the two-dimensional plane can be represented by two numbers: an x coordinate and a y coordinate. For example, (4,5) represents a point 4 units to the right of the vertical axis, and 5 units up from the horizontal axis. The sum of two points can be defined as a new point whose x coordinate is the sum of the x coordinates of the two points, and whose y coordinate is the sum of the y coordinates.

Write a program that uses a structure called point to model a point. Define three points, and have the user input values to two of them. Then set the third point equal to the sum of the other two, and display the value of the new point. Interaction with the program might look like this:

Enter coordinates for p1: 3 4
Enter coordinates for p2: 5 7
Coordinates of p1+p2 are: 8, 11
// ex4_1.cpp
// uses structure to store phone number
#include <iostream>
using namespace std;

struct phone
{
    int area; // area code (3 digits)
    int exchange; // exchange (3 digits)
    int number; // number (4 digits)
};

int main()
{
    phone ph1 = { 212, 767, 8900 }; // initialize phone number
    phone ph2; // define phone number
    // get phone no from user
    cout << "\nEnter your area code, exchange, and number";
    cout << "\n(Don't use leading zeros): ";
    cin >> ph2.area >> ph2.exchange >> ph2.number;

    cout << "\nMy number is " // display numbers
        << '(' << ph1.area << ')'
        << ph1.exchange << '-' << ph1.number;

    cout << "\nYour number is 
        << '(' << ph2.area << ')'
        << ph2.exchange << '-' << ph2.number << endl;
    return 0;
}
// ex4_2.cpp
// structure models point on the plane
#include <iostream>
using namespace std;

struct point
{
    int xCo;       //X coordinate
    int yCo;       //Y coordinate
};

int main()
{
    point p1, p2, p3;            //define 3 points

    cout << "Enter coordinates for p1: ";  //get 2 points
    cin >> p1.xCo >> p1.yCo;          //from user
    cout << "Enter coordinates for p2: ";
    cin >> p2.xCo >> p2.yCo;

    p3.xCo = p1.xCo + p2.xCo;        //find sum of
    p3.yCo = p1.yCo + p2.yCo;        //p1 and p2

    cout << "Coordinates of p1+p2 are: "  //display the sum
         << p3.xCo << "", " << p3.yCo << endl;
    return 0;
}
3. Create a structure called Volume that uses three variables of type Distance (from the ENGLSTRC example) to model the volume of a room. Initialize a variable of type Volume to specific dimensions, then calculate the volume it represents, and print out the result. To calculate the volume, convert each dimension from a Distance variable to a variable of type float representing feet and fractions of a foot, and then multiply the resulting three numbers.
// ex4_3.cpp
// uses structure to model volume of room
#include <iostream>
using namespace std;

struct Distance
{
    int feet;
    float inches;
};

struct Volume
{
    Distance length;
    Distance width;
    Distance height;
};

int main()
{
    float l, w, h;
    Volume room1 = { {16, 3.5}, {12, 6.25}, {8, 1.75} };

    l = room1.length.feet + room1.length.inches/12.0;
    w = room1.width.feet + room1.width.inches / 12.0;
    h = room1.height.feet + room1.height.inches / 12.0;

    cout << "Volume = " << l*w*h << " cubic feet\n";
    return 0;
}