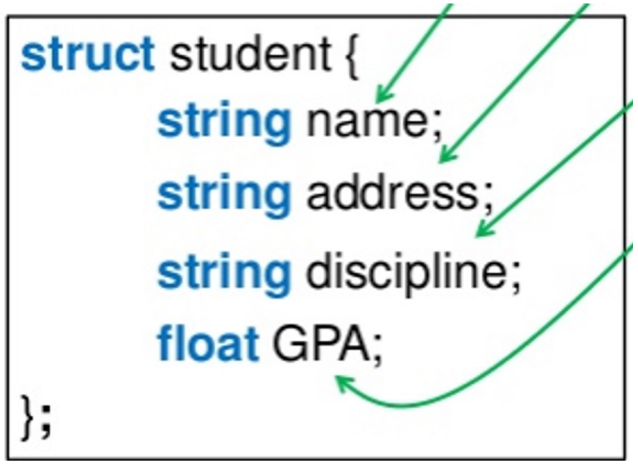
Structures

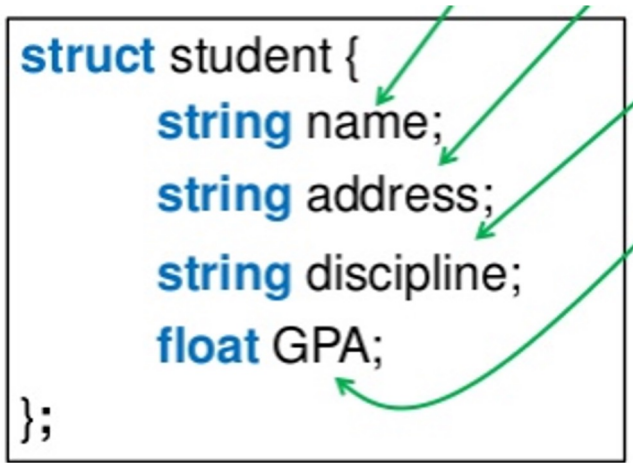
Structures: User-defined Mixed Data Types

- A Structure is a collection of related data items, possibly of different types.
- A structure type in C++ is called struct.
- A struct is heterogeneous in that it can be composed of data of different types.
- In contrast, array is homogeneous since it can contain only data of the same type.



Structures: User-defined Mixed Data Types

Define a structure type with the `struct` reserved word:




```
struct StreetAddress  //has 2 members
{
   int house_number;  //first member
   string street_name;
};

StreetAddress white_house;//defines a variable of the type
```

You use the “dot notation” to access members

```
white_house.house_number = 1600;
white_house.street_name = "Pennsylvania Avenue";
```

Structures: Assignment, but No Comparisons

Use the = operator to assign one structure value to another. All members are assigned simultaneously.

```
StreetAddress dest;
dest = white_house;
```

is equivalent to

```
dest.house_number = white_house.house_number;
dest.street_name = white_house.street_name;
```

Structures: Assignment, but No Comparisons

Use the = operator to assign one structure value to another. All members are assigned simultaneously.

```
StreetAddress dest;
dest = white_house;
```

However, you cannot compare two structures for equality.

```
if (dest == white_house) // Error
```

You must compare individual members to compare the whole `struct`:

```
if (dest.house_number == white_house.house_number
   && dest.street_name == white_house.street_name) // Ok
```

Structure Initialization

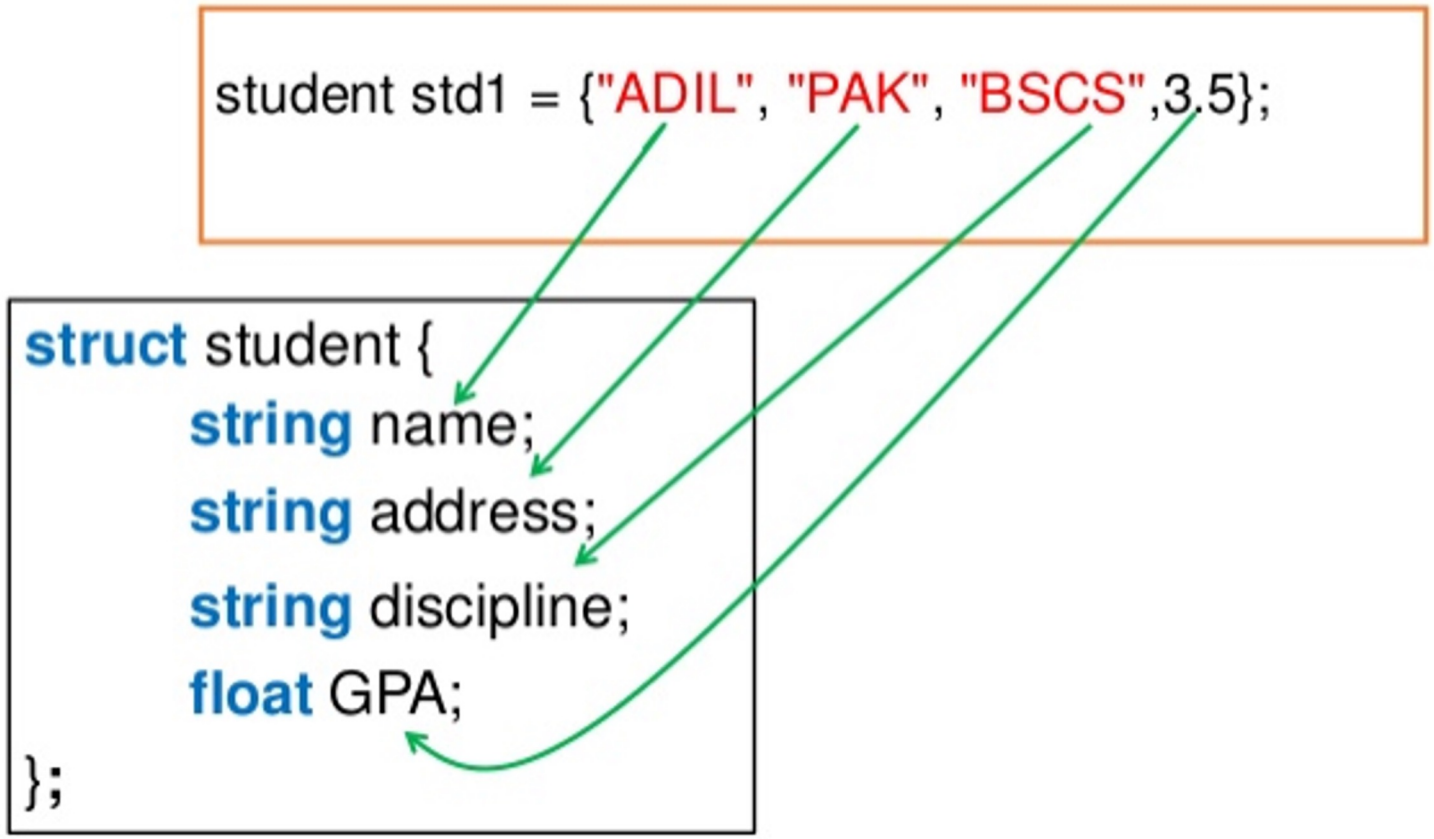
Structure variables can be initialized when defined, similar to array initialization:

```
struct StreetAddress
{
   int house_number;
   string street_name;
};

StreetAddress white_house = {1600, "Pennsylvania Ave.“};
```

The initializer list must be in the same order as the structure type definition.

Structure Initialization



Functions and `struct`

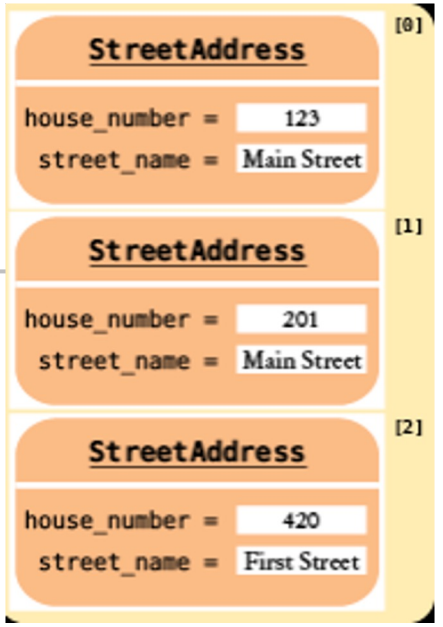
Structures can be function arguments and return values. For example:

```
void printSddress(StreetAddress address)
{
   cout << address.house_number << " " << address.street_name;
}
```

A function can return a structure. For example:

```
StreetAddress makeRandomAddress()
{
   StreetAddress result;
   result.house_number = 100 + rand() % 100;
   result.street_name = "Main Street";
   return result;
}
```

Arrays of Structures



You can put structures into arrays. For example:

```
StreetAddress delivery_route[ROUTE_LENGTH];
delivery_route[0].house_number = 123;
delivery_route[0].street_name = "Main Street";
```

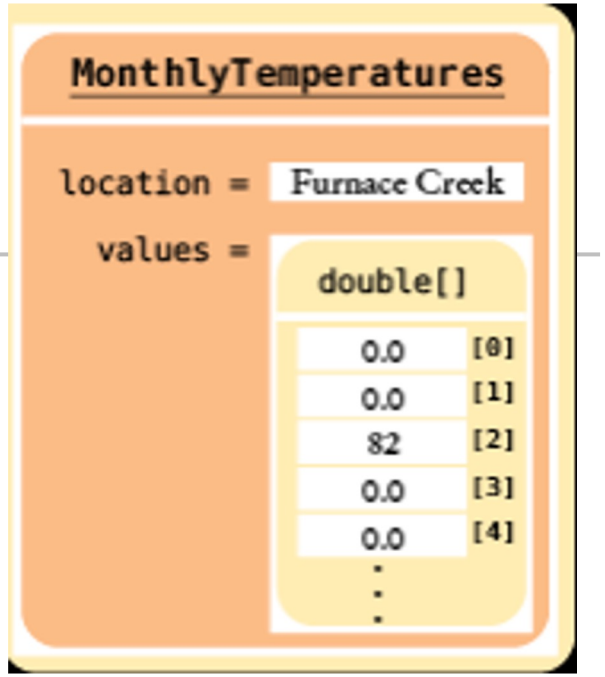
You can also access a structure value in its entirety, like this:

```
StreetAddress start = delivery_route[0];
```

Of course, you can also form vectors of structures:

```
vector<StreetAddress> tour_destinations;
tour_destinations.push_back(white_house);
```

Structures with Array Members



Structure members can contain arrays.

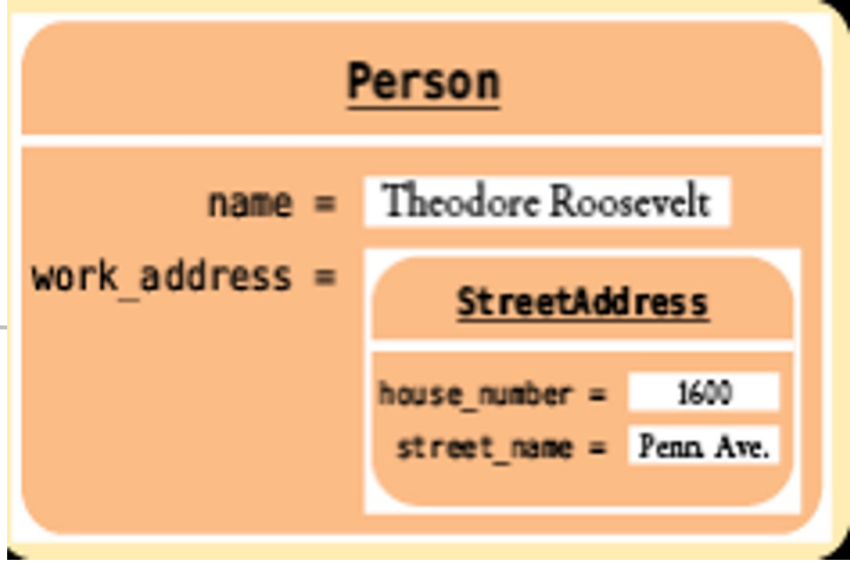
For example:

```
struct MonthlyTemperatures
{
   string location;
   double values[12];
};
```

To access an array element, first select the array member with the dot notation, then use brackets:

```
MonthlyTemperatures death_valley_noon;
death_valley_noon.values[2] = 82;
```

Nested Structures



A struct can have a member that is another structure. For example:

```
struct Person
{
    string name;
    StreetAddress work_address;
};
```

You can access the nested member in its entirety, like this:

```
Person theo;
theo.work_address = white_house;
```

To select a member of a member, use the dot operator twice:

```
theo.work_address.street_name = "Pennsylvania Ave.";
```