

# Structures

# For this week

---

- **HW 6**

- Check the deadline and start early
- File inputs, outputs, and stringstreams
- Office hours - MW: 10:15-11:15, at ECOT 743

- **Practicum 2**

- November 2nd (next week)

# Structures

# Motivation

---

- Last week's code on movie reviews
  - We read a bunch of lines from a file
    - What all did we use? *(just the concepts)*
  - We had to figure out the movie with the highest rating and the lowest rating
- What data types did we have to use and maintain in our code?
  - Did it feel like you wrote too many variables? *Perhaps a little bit*
  - How could we have *structured* it better?
- In HW 5, Q5 - what about the number of variables you used?

# Motivation

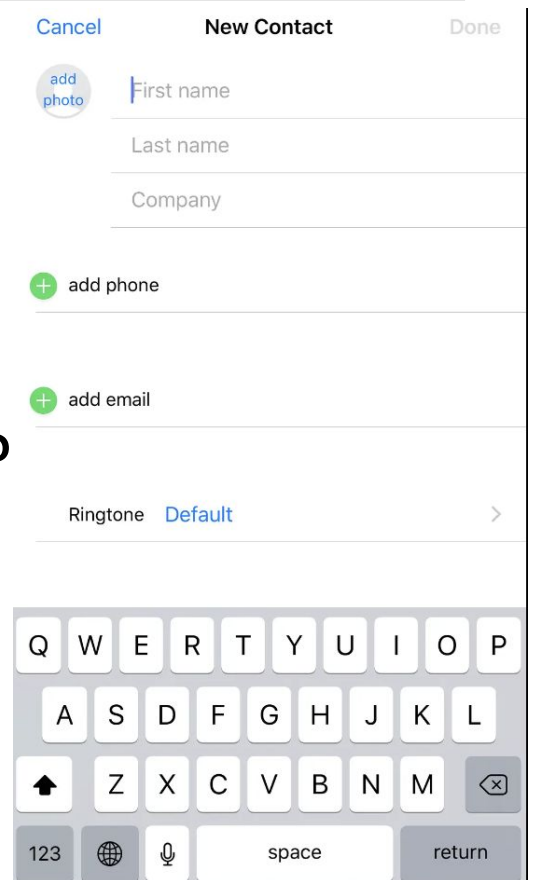
---

- For instance, let's say you need to develop an application to store contact details.
- What are some variables you like to have?



# Motivation

- You could decide on a few fields to store such as
  - Contact Name
  - Phone number
  - Email
  - A contact photo (optional)
- Are you going to store just one contact on your phone?



The screenshot shows a 'New Contact' form with a 'Cancel' button on the left and a 'Done' button on the right. The form includes an 'add photo' button with a circular icon. Below this are text input fields for 'First name', 'Last name', and 'Company'. There are two sections for adding more information: 'add phone' and 'add email', each with a green plus icon. At the bottom, there is a 'Ringtone' section with 'Default' selected and a chevron icon to the right. A virtual keyboard is visible at the bottom of the screen, showing keys for Q, W, E, R, T, Y, U, I, O, P, A, S, D, F, G, H, J, K, L, and a return key.

# Motivation

---

- Let's say your application will consist of 10 contact information
- So which is the right way to do things?
  - `string name1, name2, ..., name10;`
  - `string email1, email2, ..., email10;`
  - `string phone1, phone2, ..., phone10;`

# Motivation

---

- Let's say your application will consist of 10 contact information
- So which is the right way to do things?
  - `string names[10];`
  - `string emails[10];`
  - `string phone_numbers[10];`



# Motivation

---

- Let's say your application will consist of 10 contact information
- Option 2 looks fine
  - Only 3 arrays / vectors to keep track of
  - Names of the variables are clear, and indicates what they store
- But,
  - To display one contact, you need to look at 3 different arrays
  - To add a contact, you need to add pieces of information in 3 arrays
  - Removal - uggggh, I don't want to think of it

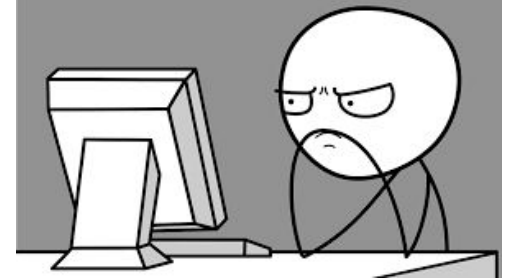
- `string names[10];`
- `string emails[10];`
- `string phone_numbers[10];`

# Motivation

---

- Using those many arrays / vectors just seems to go out of control
- What we need is somehow group the variables together, and use them as a **single entity**

- `string names[10];`
- `string emails[10];`
- `string phone_numbers[10];`



# Structures: The single entity!

---

- A **Structure** is a collection of related data items, possibly of different types that are bundled together.
- 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.

```
struct Contact {  
    string name;  
    string phone_number;  
    string email;  
};
```

# Quiz

---

- Which of the following could be potentially grouped together?
  - `string credit_card_number;`
  - `int cvv;`
  - `string bank_name;`
  - `string bank_address;`
  - `float last_month_bill;`
  - `string exp_date;`
  - `string billing_address;`
  - `double credit_limit;`

# Structures

---

- Using structures has 2 parts to it
  - Define the blueprint of the structure - which means provide the data members that needs to be grouped together
  - Define the variable of the structure

# Structures: The blueprint

---

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

```
struct Contact // has 4 members
{
    string name;
    string phone_number;
    string email;
    int age;
};
```



Structures are termed as user-defined data types, because it's the programmer that defines the blueprint (what the structure should comprise of).

Remember to define the structure on top of the program, so it's visible to the entire program

# Structures: The Variable

---

```
struct Contact // has 4 members
{
    string name;
    string phone_number;
    string email;
    int age;
};
```

Use the structure to create a variable of it's type

```
Contact contact;
```



# Test your knowledge

---

```
Contact contact;
```

In the above code, what is the data type?

1. Contact
2. contact

What about the variable name?

1. Contact
2. contact



# Structures: Declaration vs Initialization

---

- We've seen `Contact contact;`
- Is this declaration or initialization?

# Structures Initialization

---

- We see default / garbage values when we declare a structure.
- `Contact contact` declares a variable `contact` of type `Contact`

To initialize a structure variable,

```
Contact contact = {"John", "9297889388", "john@abc.com", 24};
```

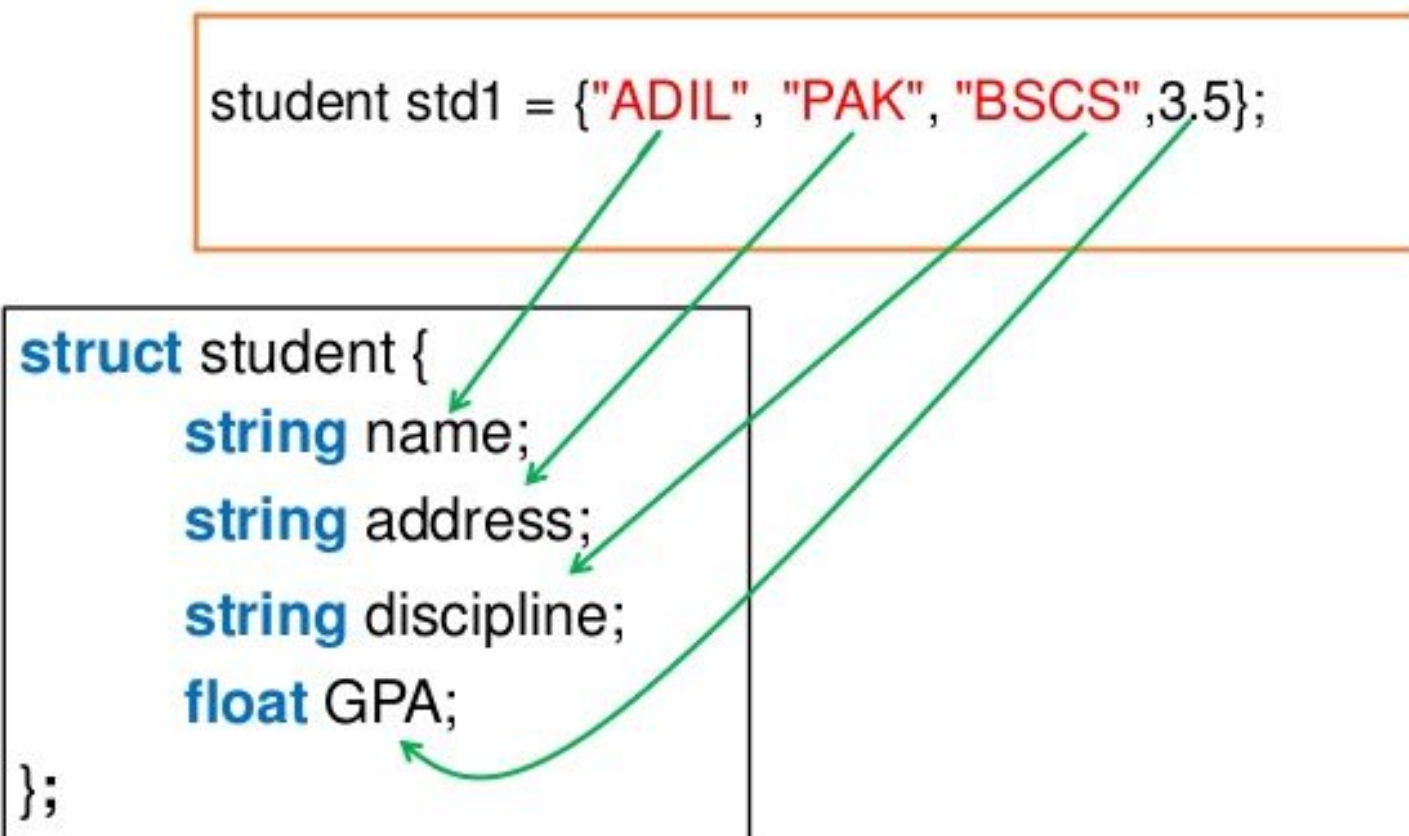
**Note: The initializer list must be in the same order as the structure type definition.**

# Structure Initialization

---

```
student std1 = {"ADIL", "PAK", "BSCS", 3.5};
```

```
struct student {  
    string name;  
    string address;  
    string discipline;  
    float GPA;  
};
```



# Accessing Structure data members

---

Let contact be initialized to,

```
Contact contact = {"John", "9297889388", "john@abc.com", 24};
```

To access each data member, use the **dot notation on the variable**. For instance,

```
cout << contact.name; // prints name of contact
contact.email = "john@def.com"; // updates the email of contact

if (contact.age >= 21) { // conditional on a structure element
    cout << "At least 21 years old" << endl;
}
```

# Test your understanding

---

- Which is a valid structure definition?

A. 

```
struct Student {  
    string name,  
    float gpa,  
    string address;  
}
```

B. 

```
Struct Student {  
    string name;  
    float gpa;  
    string address;  
}
```

C. 

```
struct Student {  
    string name;  
    float gpa;  
    string address;  
};
```

D. 

```
Struct Student {  
    string name;  
    float gpa;  
    string address;  
};
```

# Let's try to build a structure - Netflix User profile

---



# Structures: Assignment, but No Comparisons

---

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

```
UserProfile user_profile1;  
user_profile1 = user_profile2;
```

is equivalent to

```
user_profile1.name = user_profile2.name;  
user_profile1.age = user_profile2.age;  
user_profile1.language = user_profile2.language;
```

# Structures: Assignment, but No Comparisons

---

However, you cannot compare two structures for equality.

```
if (contact2 == contact1) // Error
```

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

```
if (contact2.name == contact1.name  
    && contact2.age == contact1.age  
    && contact2.email == contact1.email  
    && contact2.phone_number == contact1.phone_number) {  
  
}
```



# Structures in functions

---

You can pass structures as function arguments, and have functions return them.

Let's think about writing a function to print a contact... (4 parts)

- Name of the function
- Parameters of the function
- Output of the function
- Body of the function

# Structures in functions

---

You can pass structures as function arguments, and have functions return them too

```
void print_contact(Contact contact)
{
    cout << contact.name << " " << contact.phone_number;
}
```

A function can return a structure. For example:

```
Contact createNewContact()
{
    Contact contact; // create a contact
    contact.name = "John";
    contact.age = 30;
    return contact;
}
```

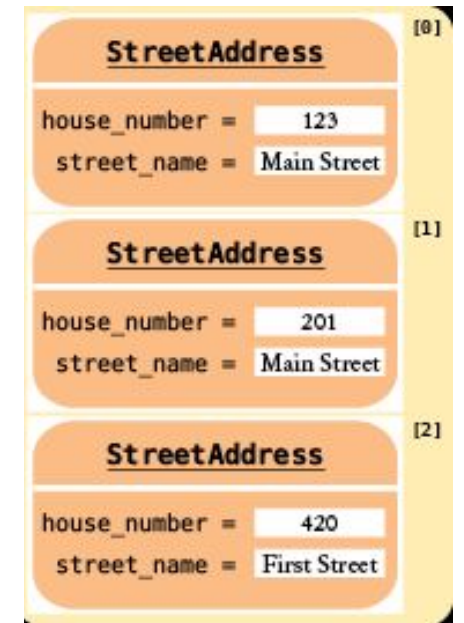
# Arrays of Structures

---

When you need more than 1 address, you need an array / vector of addresses in your program.

```
StreetAddress address[3]; //list of address
```

Q. How do you access the second address' street name?



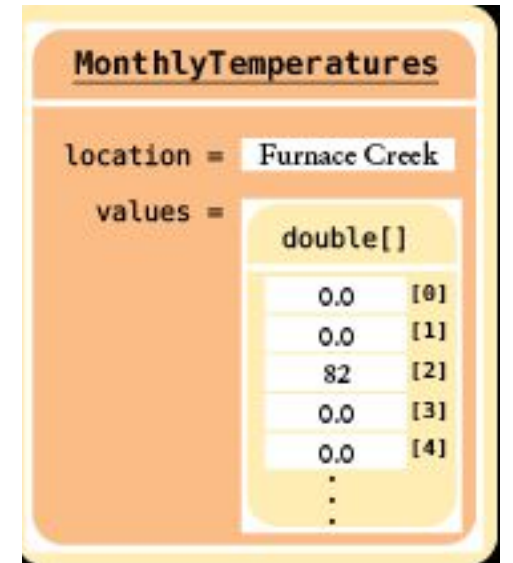
# Structures with Array Members

Structure members can contain arrays.

For example:

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

How do you access the third temperature value?



# 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.";
```

