Structs

Last Time

- We finished arrays
- Today we will
  - Look at a new way of storing data called structs (short for structures)

Arrays and Data Types

- Useful for storing a collection of data elements of the same data type (float, int, string).

```c
char myName[5]; //All elements chars
float salaries[NUM_EMP]; //All elements floats
char vowels[]={'A','E','I','O','U'};
```

- What about storing a collection of data elements of different data types?

Data with Different Data Types

- For example, what if we wanted to keep the following information on a particular employee:
  - employee id
  - SS#
  - number of children
  - salary
  - citizen

- The elements have different data types, so we can’t conveniently use an array. Instead we will use a struct (short for structure).

Structure Definition

To store this information:

- We would begin by defining a structure:

```c
struct employ
{
  int id
  int ssnum;
  int numchild;
  float salary;
  bool citizen;
};
```

Struct Terminology

For this struct:

- `employ` is the identifier name and a NEW data type.
- The individual components id, ssnum, etc. are called members.
Struct Declaration
- As with all data types, in order to use our new data type employ we must allocate storage space by declaring variables of this data type:

```c
employ engineer, tech;
```
- This will allocate space for two variables called engineer and tech with the previously described members id, ssn, etc.

Member Access Operator
- To access a struct member, we use the member access operator (period between struct variable name and member name).
- In the variable engineer of data type employ we can make the assignments:

```c
engineer.id = 12345;
engineer.ssn = 534534343;
engineer.numchild = 2;
engineer.salary = 4543.34;
engineer.citizen = true;
```

- How do we access the data in arrays?

Example One
- 22.1: Write a C++ struct data type realnum that will have members number, realpart, and intpart.
- 22.2: Declare a variable numinfo of that type.
- 22.3: Place the value 3.14159 in the field number.

Example Two
- 22.4: Write a C++ function split that accepts a variable of type realnum.
- 22.5: Assign the integer part of the number to the member variable intpart and the real part of the number to the member variable realpart.
- See the function prototype on the next slide.

Example Two Solution
- Function prototype:

```c
void split(realnum &);
```
- Function call:

```c
split (numinfo);
```
- Function definition: You write

```c
void split(realnum & numinfo){
    numinfo.intpart = numinfo.number / 1000;
    numinfo.realpart = numinfo.number % 1000;
}
```
Example Three

Consider the following struct data type:

```c
struct info
{
    int num;
    int divisors[10];
    int howmany;
};
```

22.6: Write a C++ function `compute` that accepts a variable of type `info` and returns all the divisors greater than 1 of the variable `num` in the array `divisors` and the number of divisors in the variable `howmany`.

Summary

- In today’s lecture we covered
  - structures