Structures

Sometimes we want to group different kinds of data together. For example, let's think about a bank account. What kind of information does your bank account store about you? Most importantly, it stores your bank account number (an integer) and your account balance (a float). It also stores your name and address (both strings). Often in programming, we want to group related data like this together in a single construct, and we can do that with structs.

The first thing to think about when making a structure, shortened to *struct* in C++, is what its kind, or tag, will be. For the bank account example above, a good tag might be *bankAccount*. We can define a *bankAccount* struct like this:

```
typedef struct _bankAccount{
      int accountNumber;
      float accountBalance;
      string name;
      string address;
} bankAccount;
```

We tell the computer that we are declaring (creating) a struct by saying:

```
typedef struct _tag {
} tag;
```

tag is the kind of struct you are making. (The tag is a placeholder for the variable name, which is made unnecessary by the typedef. We won't worry about that here.) Inside the curly braces, we define the fields of the struct by saying what kind they are and what their names are. This part of the struct is kind of like a database. In our *bankAccount* struct, we have four fields: accountNumber, accountBalance, name, and address. We also tell the computer the type of each field by saying something like "int accountNumber;" so that the computer knows that the accountNumber field is an integer. We close the curly braces and put a semicolon at the end. The struct declaration goes before the int main() part of the program, as the next example shows.

Each time we go to use a struct, we need to give it a unique name. Think about bank accounts again. Each bank account is different, and my bank account and your bank account hold different information and have different, unique names. We'll do the same thing here by giving each instance of our struct a unique variable name. So let's make some accounts like this:

```
#include <iostream>
using namespace std;

typedef struct _bankAccount{
      int accountNumber;
      float accountBalance;
      string name;
      string address;
} bankAccount;


int main(){
```

```
    bankAccount alisonBankAccount; //make a bankAccount for Alison
    alisonBankAccount.accountNumber = 14538;  //assign the account a number
    //give Alison some money
    alisonBankAccount.accountBalance = 1000000.01;
    alisonBankAccount.name = "Alison Norman";  //assign name and address
    alisonBankAccount.address =
          "1600 Pennsylvania Avenue NW Washington, DC 20500";
}
```

We use dot notation to access the fields of a struct. To get the field of a struct, we write the variable name of an instance of the struct (alisonBankAccount), a period, and then the name of a field in that struct (accountNumber). We have to assign values to the fields in the struct because when we initially define the struct, the computer only knows the type of value to expect, but not the actual values. That way, we can have multiple different bankAccounts, all with different balances and information. For example, let's pretend that in addition to Alison's bank account, we made an account for Bob Bobson. Here's the same code as above, but now we've added a second instance of the bankAccount struct:

```
//make two different bank accounts
bankAccount alisonBankAccount;
bankAccount bobBankAccount;

//assign two different account numbers
alisonBankAccount.accountNumber = 14538;  //assign the account a number
bobBankAccount.accountNumber = 54324;

//assign different account balances
alisonBankAccount.accountBalance = 1000000.01;
bobBankAccount.accountBalance = -122.52;

//give Bob some money to get out of debt
//i.e. add 300.95 to the current value of accountBalance
bobBankAccount.accountBalance += 300.95;

cout << "Bob has $" << bobBankAccount.accountBalance << ".  Yay Bob!\n";
```

We can give Alison and Bob's *bankAccount* structs completely different information, and we can also modify the values that the fields hold. Notice how we gave Bob some timely money to get him out of debt with the line "bobBankAccount.accountBalance += 300.95;". The "+=" operator means that you increase the value of the variable to the left of the "+=" by the value on the right side of the "+=" and save it as the same variable. In other words, the following two lines are equivalent:

```
bobBankAccount.accountBalance += 300.95;
bobBankAccount.accountBalance = bobBankAccount.accountBalance + 300.95;
```

Challenge 1:

Add another bank account to the example above. Deposit $500 dollars into Alison's account, then withdraw $400 from Bob's account and place it in the account you created. Then print out the balances from all three accounts.

Challenge 2:

Create a struct for super heroes. The fields in the struct will include the superhero's name, his arch-nemesis, his weakness, and his sidekick (these are all strings). Add some extra fields of your own. Then make some instances of the superhero struct using your favorite super hero's for inspiration.

Source