

Re: Passing Structure to a function

Source: <http://coding.derkeiler.com/Archive/C/ CPP/comp.lang.cpp/2003-11/1417.html>

From: Niklas Borson (niklasb_at_microsoft.com)

Date: 11/13/03

Date: 12 Nov 2003 15:17:02 -0800

"kazack" <kazack@talon.net> wrote in message news:<GAMsb.6713\$Bv6.2039282@news1.epix.net>...

>

[snip]

>

> #include <string>

> #include <iostream>

> #include <fstream>

> using namespace std;

>

> int Menu();

> void Menu_Selection(int, struct phonerec phone);

> void Input_New_Record(struct phonerec phone);

> void Write_To_File(struct phonerec phone);

You don't need the 'struct' keyword to refer to a structure type, only to declare/define it. I suspect the only reason you used the 'struct' keyword above is because otherwise you got compiler errors. The reason for the errors is that phonerec hasn't been declared or defined yet. You should probably move the structure definition (below) so that it precedes the function declarations, and then remove the 'struct' keyword from the function declarations.

What you have actually done is essentially equivalent to this:

```
struct phonerec;
void Menu_Selection(int, phonerec phone);
void Input_New_Record(phonerec phone);
void Write_To_File(phonerec phone);
```

The first line above declares (but does not define) the phonerec structure. A structure declaration merely says that a structure exists but not what its members are. Some uses of a type require its definition (e.g., creating an instance of the type), but others require only its declaration (e.g., creating a pointer or reference to the type).

A declaration like the one above is called a "forward declaration"

comp.lang.c++. Re: Passing Structure to a function

because it declares a struct which will be defined later. This is sometimes necessary. For example, if struct A contains a member of type B* and struct B contains a member of type A*, then either A or B must be forward declared.

It is also possible for a struct to be declared but not defined at all, or at least not publicly. This is called an "opaque struct". The FILE structure in the C runtime library is an example of this. Windows handles are another example. Yet another is the "Pimpl idiom" (Google is your friend).

```
> struct phonerec
> {
> string fname;
> string lname;
> string number;
> };
>
>
> int main()
> {
> phonerec phone;
> int Menu_Choice;
> Menu_Choice = Menu();
```

I recommend using an enum rather than an int for Menu_Choice. Then you can use meaningful names instead of raw numbers like 1, 2, 3.

I also like function names to be verbs and type names to be nouns. Menu would be a good name for a class that represented a menu. A better name for your function in my opinion would be Show_Menu.

Note that you can also declare and initialize a variable in one step, and this is usually preferred in C++, e.g.:

```
MenuChoice choice = Show_Menu();

>
> if(Menu_Choice != 3)
> {
> Menu_Selection(Menu_Choice,phone);
> main();
> }
```

Note that you're passing phone by value to Menu_Selection. This means a copy of phone is passed to the function. Anything the function does with phone will only modify its copy. The phone object in main will be untouched.

Also, you probably just want to have a loop here rather than calling main recursively:

Re: Passing Structure to a function

comp.lang.c++. Re: Passing Structure to a function

```
for (MenuChoice choice = Show_Menu();
    choice != QuitMenuItem;
    choice = Show_Menu())
{
    Process_Command(choice, &phone);
}

> return 0;
> }
>
>
> void Menu_Selection(int Menu_Choice, struct phonerec phone)
> {
> switch(Menu_Choice)
> {
> case 1:
> {
> Input_Phone_Record(phone);
> break;
> }
> case 2:
> {
> //Reserved For Menu Option 2 to search for record
> // When this feature is set up it will input one record at
> // a time into the struct and check to see if search is equal
> // and will do this for all records. I know this is not the
> //most efficient way but it lets me know if I am using
> //structs properly.
> break;
> }
> }
> }
```

Again, I would choose a more verb-like name for the above function (e.g., ProcessCommand) and use an enum instead of raw integers to represent the menu selection.

Also, phone is passed by value throughout your program, which means if any function modifies phone it changes only its own copy. I would probably use something more like the following function declarations:

```
void Process_Command(MenuChoice choice, phonerec* phone);
phonerec Input_Phone_Record();
void Write_Phone_Record(const phonerec& phone);
```

In the first function, phone is an "in/out" parameter so it is declared as a pointer (alternatively it could be declared as a non-const reference). The second function does not use the value of the existing phone record but simply returns a new one. In the

third function, phone is an "in" parameter; it could be passed by value but it is more efficient to declare it as a const reference.

```
> void Input_New_Record(struct phonerec phone)
> {
> cout << "Enter The First Name: ";
> cin >> phone.fname;
> cout << endl << "Enter The Last Name: ";
> cin >> phone.lname;
> cout << endl << "Enter The Phone Number: ";
> cin >> phone.number;
> Write_To_File(phone);
> }
>
> void Write_To_File(struct phonerec phone)
> {
> ofstream outdata;
> outdata.open("phonebook.dat",ios::app);
> outdata << phone.fname;
> outdata << ",";
> outdata << phone.lname;
> outdata << ",";
> outdata << phone.number;
> outdata << endl;
> outdata.close();
> }
```

The following function is essentially equivalent to the one above but is, I think, a little more concise and readable. Also the parameter is declared as a const reference which reduces unnecessary copying of the phonerec structure.

```
void Write_Phone_Record(const phonerec& phone)
{
    ofstream outdata("phonebook.dat", ios::app);
    outdata << phone.fname << ',' << phone.lname << ',' << phone.number << endl;
}

> int Menu()
> {
> int choice;
> while(choice != 1 && choice !=2 && choice !=3)
> {
> cout << endl << endl << endl;
> cout << "1. Input New Record" << endl;
> cout << "2. Search For A Record" << endl;
> cout << "3. Quit Program" << endl;
> cout << "Please Enter A Choice: ";
> cin >> choice;
> }
> return choice;
```

> }
>
> *Thank you once again. And also how valuable are unions?*

They have their place, but for the most part they're dangerous and evil.

> *From what I have*
> *glanced at all it is is a renamed structure or something along that line.*

There's a crucial difference. All the members of a union share the same storage. So if you have a union of, say, an int, a long, and a char*, only one of those members is actually valid at any given time — and it's up to you the programmer to know which.

It looks like you've picked a good learning exercise. Good luck!