**Name: Soham Thaker SID: 011-748-159**

1. **Bugs found & Corrected Code**

Line 31:

**Original Code:**

Scanf(“%s%\*c”, &yesNo);

**Updated code:**

scanf("%c%\*c", &yesNo);

**Solution:**

Here, %s before %\*c was removed. Error the compiler was throwing was stack around the variable yesNo was corrupted. We do not need to use %s. %s is used when a string is to be entered and stored into an array. Here we are only taking one character which is either a ‘y’ or ‘Y’ so %c does the job, takes the character, and stores that character into a variable.

Line 59:

**Original Code:**

if (yesNo == "y" || yesNo == "Y")

{

do {

printf("Please enter the contact's apartment number: ");

scanf("%d%\*c", &address->apartmentNumber);

} while (address->apartmentNumber < 1);

}

**Updated Code:**

if (yesNo == ‘y’ || yesNo == ‘Y’)

{

do {

printf("Please enter the contact's apartment number: ");

scanf("%d%\*c", &address->apartmentNumber);

} while (address->apartmentNumber < 1);

}

**Solution:**

Updated the condition from double quotes to single quote, since single quote is for a character, whereas double quote is used for a string literal. The code was skipping the if statement when double quotes were present.

Line 90:

**Original Code:**

if (yesNo == ‘y’ || yesNo == ’Y’)

{

printf("Please enter the contact's home phone number: ");

scanf("%10[^\n]%\*c", numbers->home);

}

**Updated Code:**

if (yesNo == 'y' || yesNo == 'Y')

{

printf("Please enter the contact's home phone number: ");

scanf("%10[^\n]%\*c", numbers->home);

}

**Solution:**

The variable yesNo can only either be, ‘y’ or ‘Y’. It cannot be both. However, the original condition, uses && operator which is incorrect. We need to use || operator for the condition to become true and validate if the user inputted correct character.

Line 37:

**Original Code:**

char middleInitial[5];

if (checkIfYes(yesNo))

{

printf("please enter the contact's middle initial(s): ");

scanf("%6[^\n]%\*c", name->middleInitial); //fixed error here

}

**Updated Code:**

char middleInitial[6];

if (checkIfYes(yesNo))

{

printf("please enter the contact's middle initial(s): ");

scanf("%5[^\n]%\*c", name->middleInitial); //fixed error here

}

**Solution:**

The array size was too small too hold the string values plus the null byte at the end of every string in c. Also, the argument, ‘%6’ means maximum 6 characters can be stored into the array ‘middleInitial’. However, maximum size of the array defined in the contacts.h were 5 which is why when the value of ‘middleInitial’ was printed it was either appending the following string to it or did not print the full string, rather printing partial characters. The solution to this was to resize the array to maximum 6 characters, which is size+1, 6th element being the null byte and when taking the input and storing it into the variable, I changed the argument to ‘%5’ which takes exactly 5 or less characters and stores it into the variable.

1. **Explain how the struct Contact looks when the data is laid out in memory with other structs embedded within it.**

Here, the structure Contact has three other structures embedded within it, namely Name, Address and Numbers. Also, the objects of these three structures are defined as well. The structure Contacts has access to variables of all three structures embedded within itself, making it a nested structure.

Now the structure Contact contains variables of all the three structures defined within the Contact structure. The only way to access these variables of a structure within a structure is to create objects of respective structures that hold the variables. Let us say a programmer wants to use the variable of the structure Name, called firstName within main(), all he or she would have to do is create an object of the structure Contact, like ‘struct Contact contact’. Since objects of the structures, Name, Address & Numbers are already created, they can access all the variables within this structure by writing object name of Contact structure, which is ‘contact’ followed by dot operator, followed by the object name of the structure that holds the variables he or she is trying to access, followed by the member name of that structure the programmer needs the access to. For example, ‘contacts.name.firstName’.

The reason why the programmer would have to use a second object after the dot notation is that the original structure, ‘Contact’ holds data of other structures and thus making this a nested structure. So, the structure ‘Contact’ is a structure in the memory which has access to the data of other structures defined within itself, for which the objects are already created, defined and ready to use.

1. **Function replacing the repeated code.**

**Function prototype:**

int checkIfYes(char yesNo);

**Function declaration:**

int checkIfYes(char yesNo)

{

return (yesNo == 'y' || yesNo == 'Y') ? 1 : 0;

}

**Function Utilization in main or another function:**

void getName(struct Name \*name)

{

printf("Do you want to enter a middle initial(s)? (y or n): ");

scanf("%c", &yesNo); // fixed error here

if (checkIfYes(yesNo))

{

printf("Please enter the contact's middle initial(s): ");

gets();

scanf("%6[^\n]%\*c", name->middleInitial); //fixed error here

}

}

**Explanation:**

Here the function, ‘checkIfYes’ takes the argument of char variable, ‘yesNo’ which is a user input, either ‘y’ or ‘Y’ which is then passed into the function. The parameter of this function is char ‘yesNo’ itself. The function has one return statement, which uses a ternary operator. If the value of char variable, yesNo is either ‘y’ or ‘Y’, then it will return 1, otherwise it will return 0. The return type is an int since the function either returns, 1 or 0. Now, the way if statements work is, if the value inside the brackets hold a value of 1 or more than 1, in other words, if the value returns true then if statement will be executed, otherwise if it returns 0 then, the control from if statement rolls over to the following line, skipping the sequence inside the if statement. So, if a user enters ‘y’ or ‘Y’, the if statement will run the code inside it and exit the if statement, in a normal way. If any other input is entered, the code inside if statement will be skipped.

The purpose of this function is that, if I were to change the condition, which as of now takes, ‘y’ or ‘Y’ to lets say, ‘n’ or ‘N’, I would only have to do it in place, which is inside the function and changes will be reflected at any function call in the source file.

The function, ‘checkIfYes’ is already implemented in the source code, and the program runs as expected.

1. **Explain what is %\*c does and why it needs to be in this program.**

%\*c expression is used, to consume the new line left in the input buffer, when the user hits the enter key, every time after the user inputs the values.

If %\*c is not present in the program, when the control comes over to the second scanf function which has an argument [^\n] in the scanf function, it tries to wait for the user input but instead what happens is that the newline character is already present in the input buffer since it was not cleared and thus the argument [^\n], which means read all the characters until you find the new line character becomes true, because newline was the very first character and so it doesn’t wait for the user input and skips the user input.

Thus, what %\*c does is, it clears the input buffer from the previous scanf function which was a newline character for the subsequent scanf function so that the subsequent scanf function call waits for the user input rather than interpreting the new line character as a user input and not stopping for the user input.

1. **Applications of format code “%[^\n]” in strings & why “%s” cannot be used when dealing with string literals**

“%s” format specifier ignores any leading whitespaces in a user input. For example, if a user enters “Robert Herjavec”. Here, the word, Robert will be stored into an array of string, however since there is a whitespace between the two words, ‘Herjavec’ is being ignored and the input remains in the input buffer.

So, to read string that has delimiters like whitespace, “%[^\n]” is used so that all the characters are to be stored into the string literal except the new line character. The moment the user hits the enter key, the input is converted into the delimiter for that string and the data is stored into a character variable until the newline character, excluding the newline character.