**Why code quality is important**

it is important to keep your code tidy because

* It will be easy to navigate through your code
* To find missing brackets or other symbols and errors
* For other users to understand and input code into the project (group projects)

Here is an example of “messy” code

{ if (tbFirstNameBox.Text == "Please enter First name" || tbFirstNameBox.Text == ""){ tabControl1.SelectedTab = tabPage1;return false;}if(tbLastNameBox.Text=="Please enter Last name”||tbLastNameBox.Text == ""){tabControl1.SelectedTab = tabPage1;return false;}if (tbAddressBox.Text == "Please enter full address" || tbAddressBox.Text == ""){tabControl1.SelectedTab=tabPage1;return false;}return true;}

This code would be considered messy because all elements of the code are combined without spacing to differentiate separate elements. It is important to ensure the accuracy of the data to be able to read it back quickly and make changes accordingly. If the code is hard to read and differentiate, it may be inefficient when changes are due.

Here is an example of “tidy” code

//Name input

private bool validation\_name()

{

if (tbFirstNameBox.Text == "Please enter First name" || tbFirstNameBox.Text =="")

{

tabControl1.SelectedTab = tabPage1;

return false;

}

if (tbLastNameBox.Text == "Please enter Last name" || tbLastNameBox.Text == "")

{

tabControl1.SelectedTab = tabPage1;

return false;

}

if (tbAddressBox.Text == "Please enter full address" || tbAddressBox.Text == "")

{

tabControl1.SelectedTab = tabPage1;

return false;

}

return true;

}

This code is ‘tidy’ because it is spaced out correctly the brackets are aligned and easy to navigate each line of code is on its line to prevent confusion and human error it also contains notes to state what the list of code does so when referring back to this code after a period of time you will still understand what the code means and can do, making it a lot easier to use in a new program.

It is important to define integers correctly

* To prevent using the incorrect integer
* To prevent duplication of the same integer

Here is an example of incorrectly defined integers

//calculations

float a = pond1 \* 2 + pond2 \* 2;

float b = pond1 \* pond2 \* pond3;

float c = pond1 \* pond2;

This is defined incorrectly because it does not explain what each “pond1 to pond 3” actually is making the calculation a lot harder to understand and would cause issues with calculation a way to resolve this is to state what the int is in its name or at least leave a comment above saying what each one is eg.

//a is the perimeter

//b is the volume

//c is the area

Here is an example of correctly defined integers

//calculations

float perimeter = pondLength \* 2 + pondWidth \* 2;

float volume = pondLength \* pondWidth \* pondDepth;

float area = pondLength \* pondWidth;

This is defined correctly as it states what each int means and makes it easy for the user to understand how the calculation is made they are also not the same or similar which prevents human error such as: float a = Aa \* 2 + Aaa \* 2;