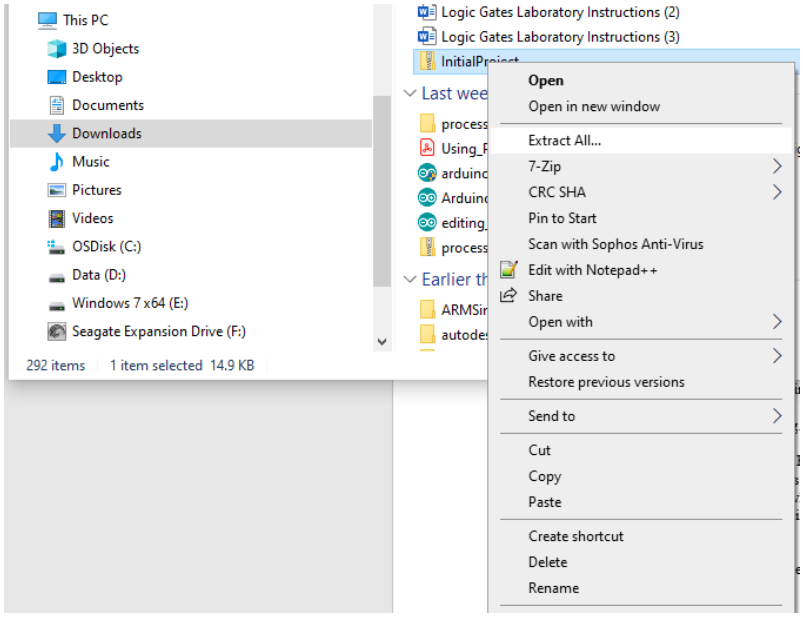
# Introduction to the C language and Variables

## Introduction:

For this class you will be using a simple hardware development environment to introduce you to concepts associated with C Programming. You will use this environment as it allows more access to the details of the hardware. The objective of this lab is to allow you to understand how to use typed variables in the C coding language.

## Importing the Example Code:

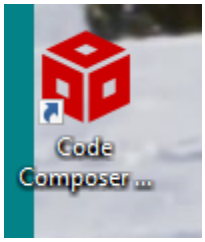
Let's import some example code to get started. To do this download the zip file InitialProject.zip from i-learn. Unzip the file by selecting the zip file and right clicking on the file name, then click Extract All… and chose where you want to extract the file. You can leave it in your Downloads directory if you like:.



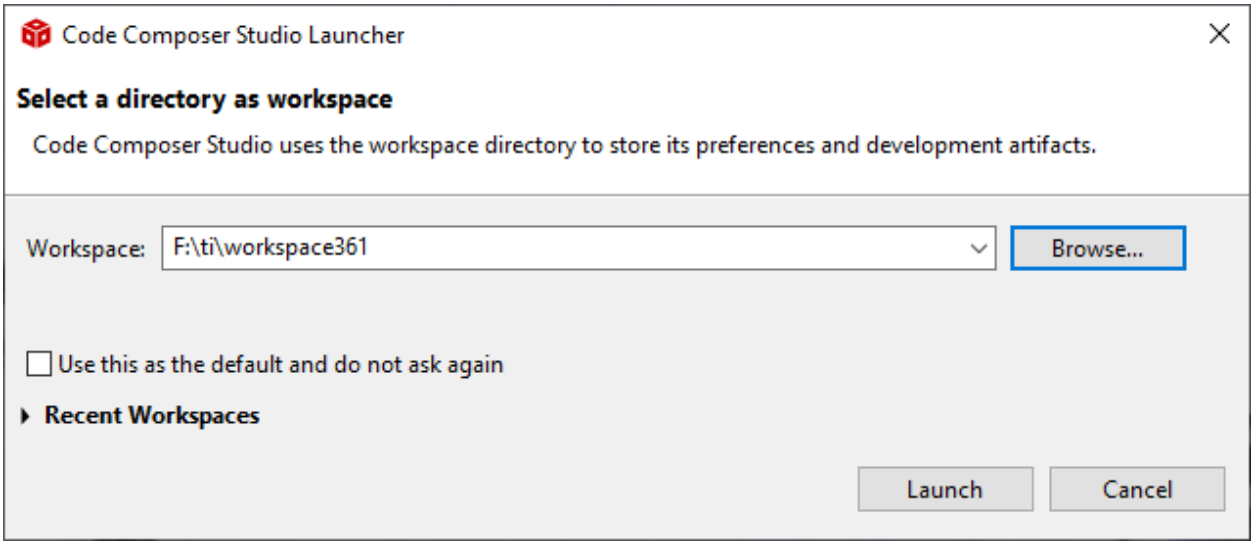

When you have the file unarchived, you can now import it into Code Composer.

# Code Composer

It is already assumed that you have installed and are at least minimally familiar with the Code Composer IDE. So start Code Composer as you normally would by double clicking on the icon:

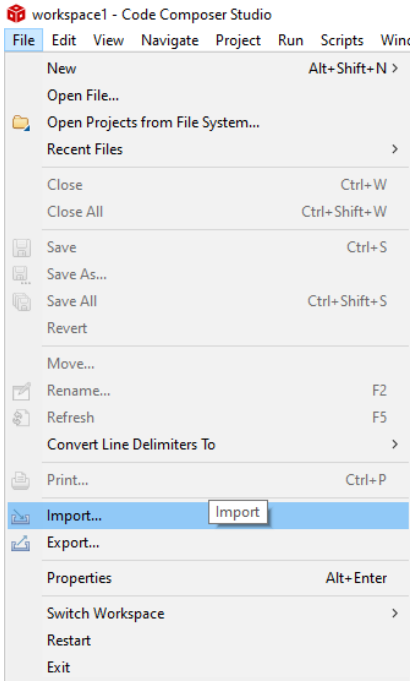



As the program starts you should be prompted for the workspace you want to use.

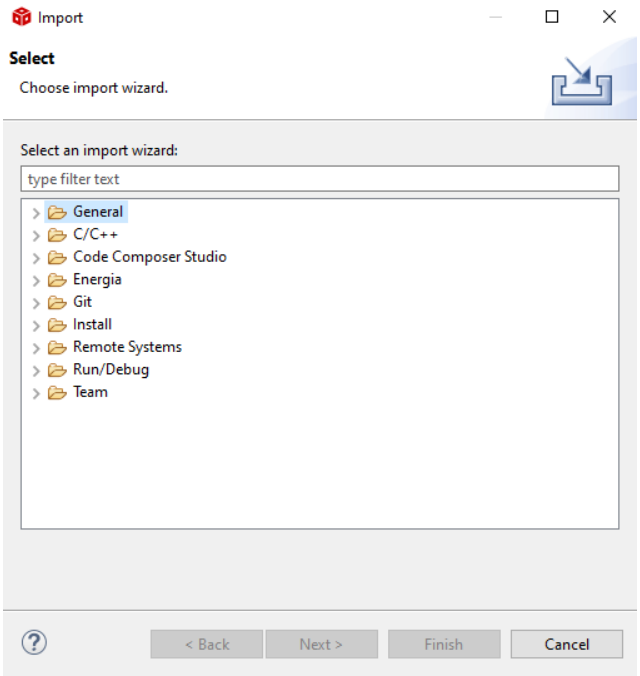



Remember this workspace is the directory where you project files will be placed. You can use several different workspaces if you want to work on very different projects. Click Launch when you have specified the work space directory (you can just use the default.)

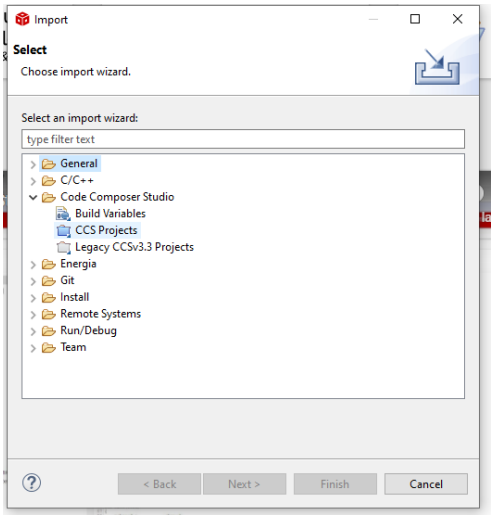
If you haven't already connect your MSP432P401R to the computer via a USB cable. Now let's import the project into Code Composer. To do this select File->Import



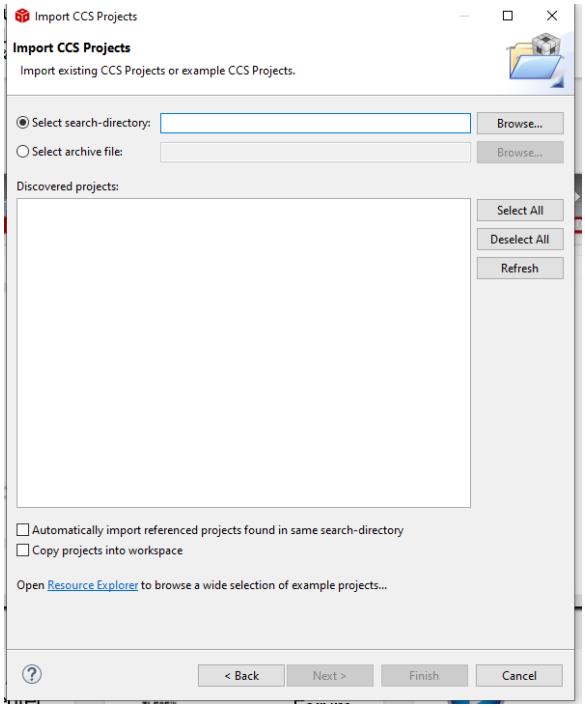

Then you should see this dialogue box:



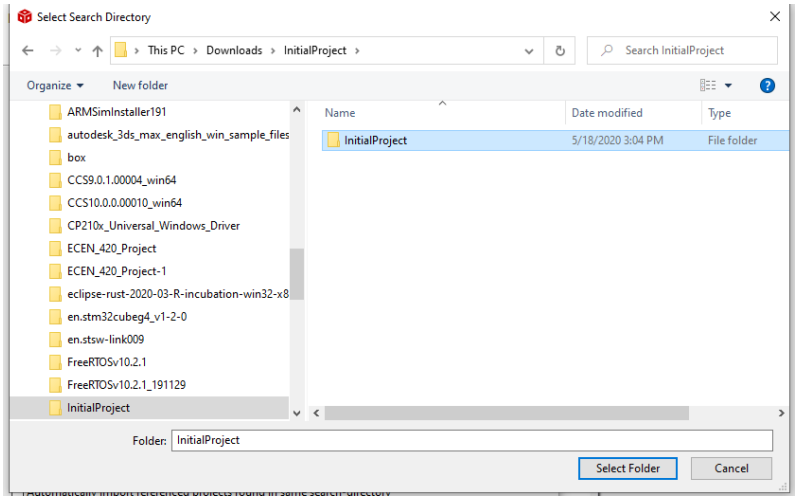
Now select Code Composer Studio, then CCS Projects:



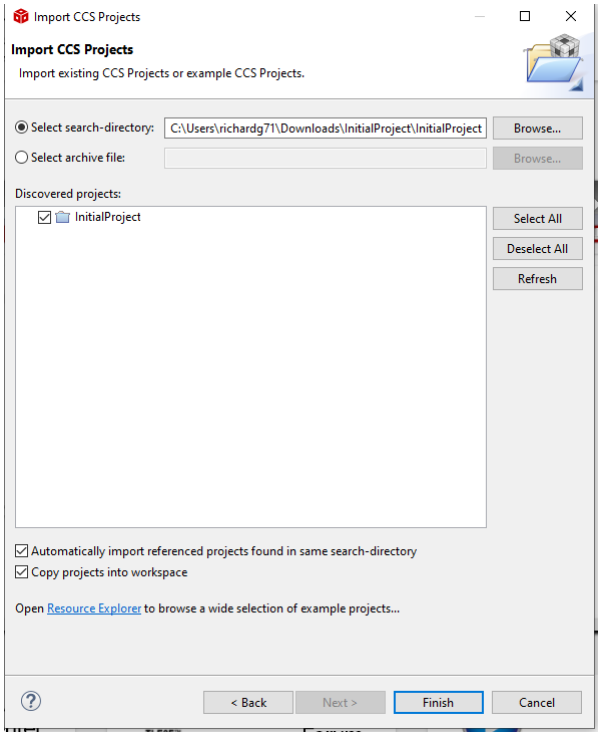

And you should see this dialogue box:



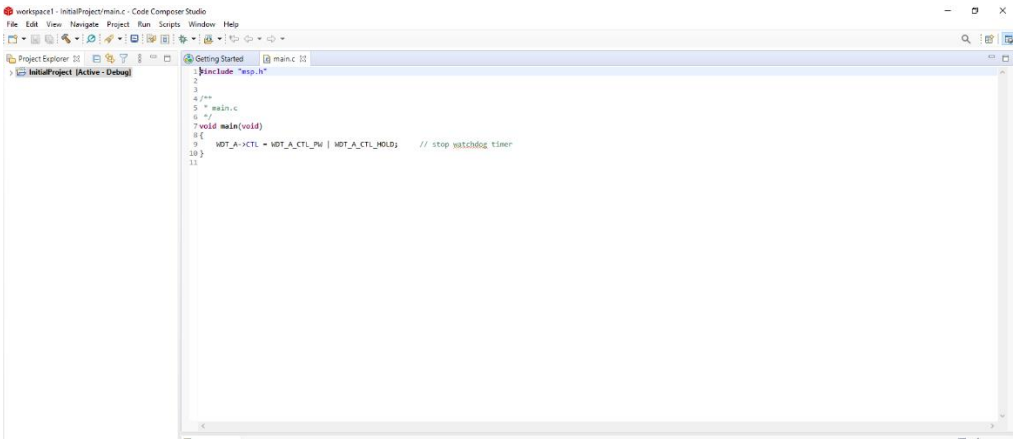
Browse to the directory where you unzip the archive. In my case it was my Downloads directory:




Click the Select Folder button. Then you should see this in the dialogue box:



Make sure the bottom two check boxes are checked, then click Finish. You should now see this project in the Project Explorer window:

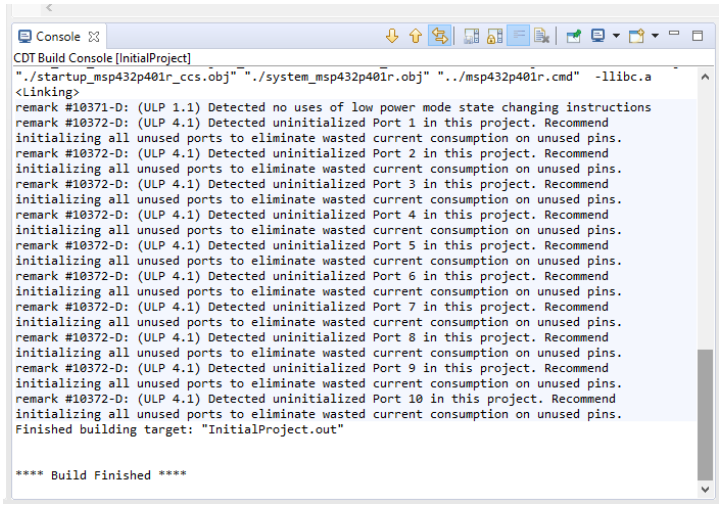



You will also see the main.c file in the Editor Window. Now this will look somewhat different from a Python code set. So let's look at some details. First, you will notice there is a main function. In C this is where execution of the code starts, not at the top of the file.

Also notice that there is a void before main function declaration. This tells the system that this particular function is set up to return nothing (void means nothing in C). C is a typed programming language, which means you'll always need to tell it the type of variables. Inside of the main function () you'll see a void, this simply means that in this case we are not going to pass the function anything.

Also notice the curly braces around the statements in the function. In Python you use indentation to denote the statements in a function, in C you will use {}.

Before you start making changes, let's make sure the code compiles. Right click on the project, then click on Build Project. In the Console window you should see the following:



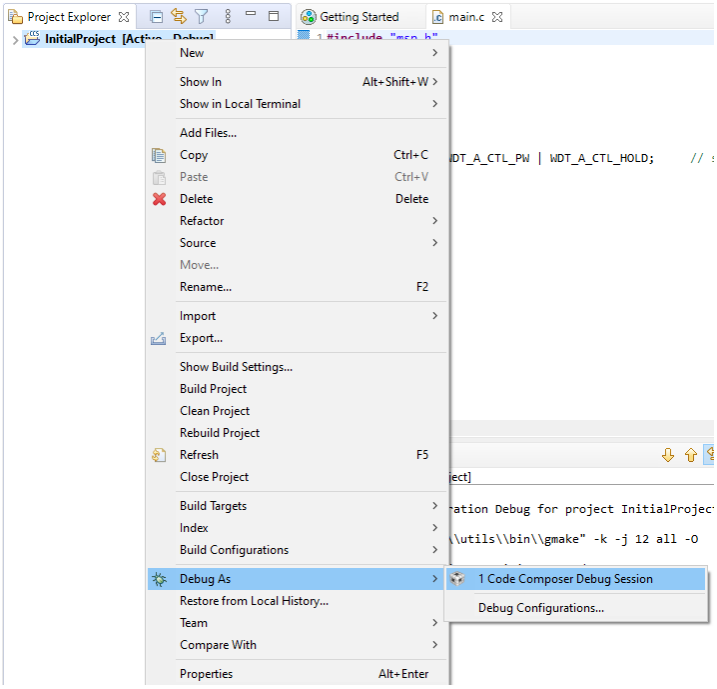

The first time you build a project you will often get suggested actions, like those shown above. They normally only occur the first time you build a new project.

The Build process is also unique to the C programming language, many languages, like Python or Java, are interpreted languages, means that there is a program running on the device that takes each command and interprets it

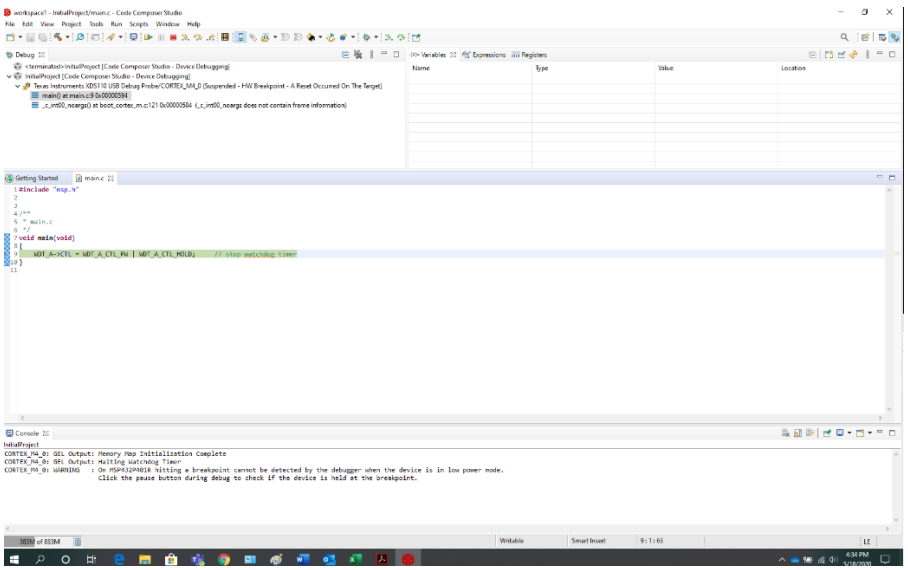
for the computer. Thus you need not only your code, but the interpreter program to be running, for your program to execute successfully.

The C programming language is a compiled language. This means that a special program called a compiler takes your program and turns it into machine code. When you want to run your program you don't need any other program, your program is in machine code in a run-able state.

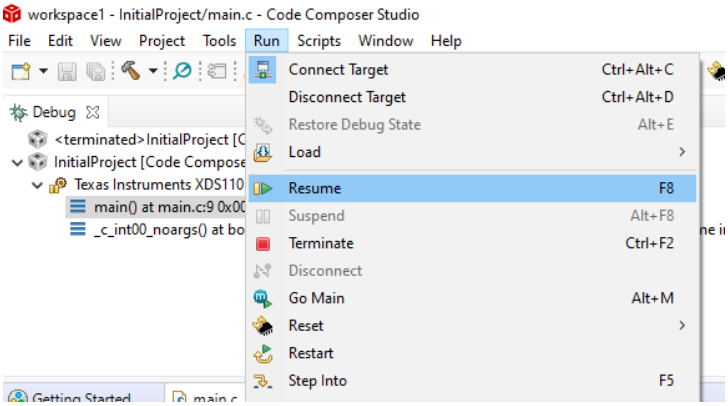
You can actually now run your program by right clicking on the project and hitting the Debug As ->Code Composer Debug Session;



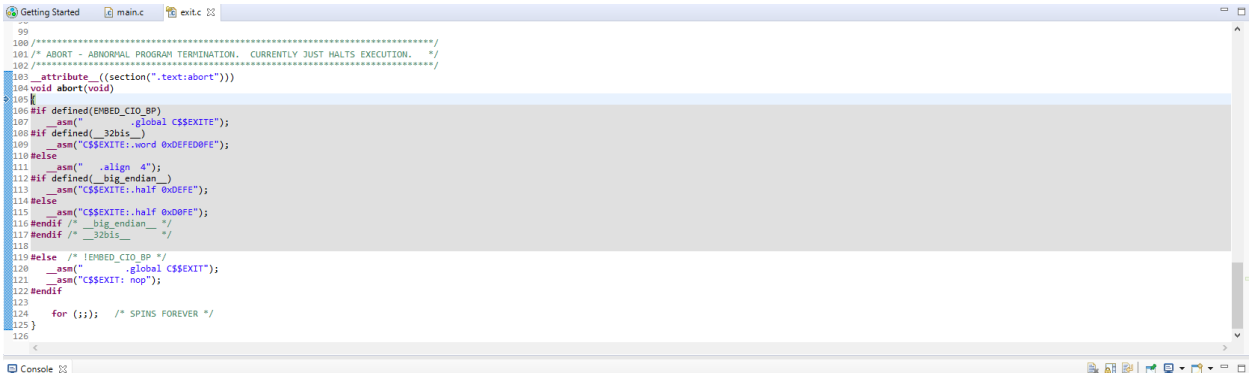

The program will now be uploaded to the MSP432P401R through the USB cable. The view will also switch to the Debug View, and you should see this:



The program is on the MSP432P401R hardware, but you now need to press Resume to actually run the code:




In this case the program will run, and the reach the end of the main function, and abort:



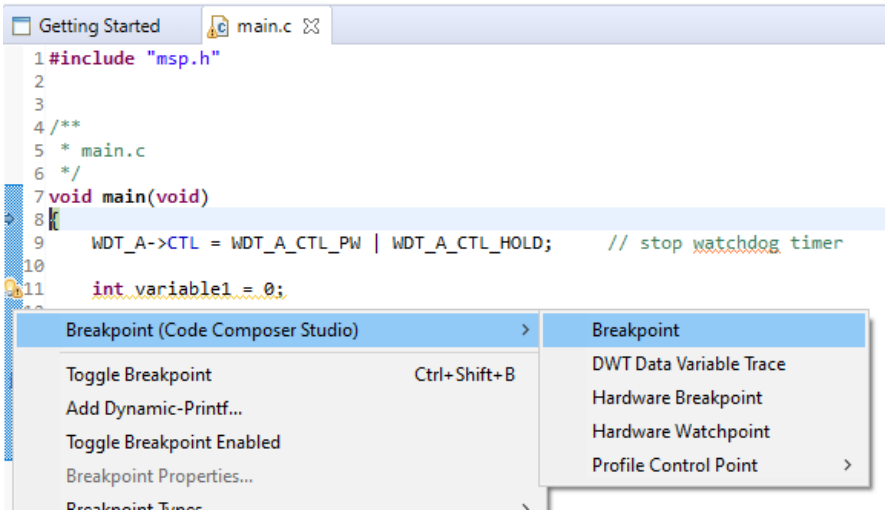

Now that you have a framework, you can add code.

## Simple Typed Variables

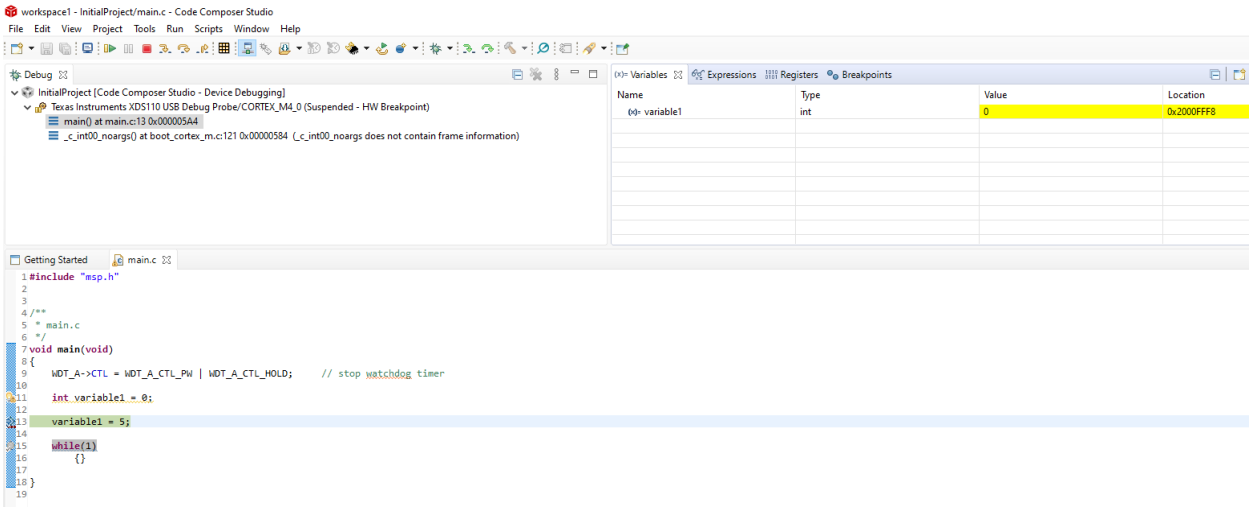
So let's understand typed variables in the C coding language. Change your main.c add a variable, like this:

By the way, you've also added a while forever loop, just so the program keep running forever, and not reach the abort code. The declaration for any variable is simple, first you detail the type, then the name of the variable, and then, optionally, you'll can initialize the variable to a value.

Compile and load the code. Before running the code, however, let's put some break points in the code to see what is happening with our variable. To add a break point, after you've reached the debug view, go to the start of line 13, then right click and you should see the dialogue box pop up to add a breakpoint:

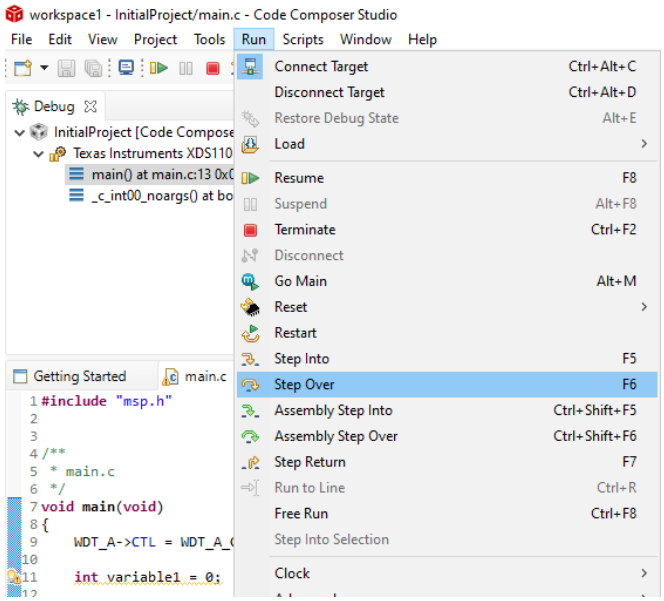



This will stop the execution of the code just before line 13. Now go ahead and hit resume. Your code should run, but stop on line 13. Code Composer provides a way for you to look at the value associated with your variable. Up in the right hand corner of the display you can see the value of the variable:

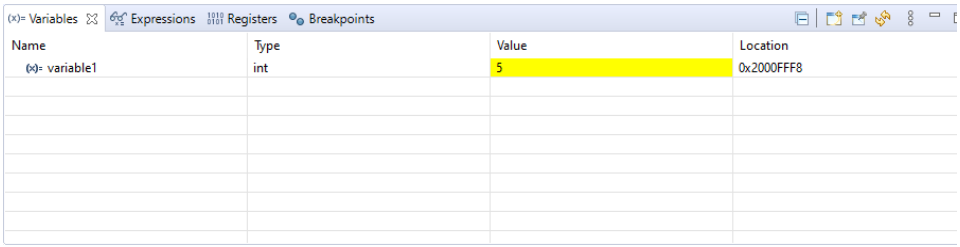



In this case you have a variable named variable1, it is of type int, it's current value is 0, and its Location is 0x2000FFF8 (your actual value may be different depending on where Code Composer has decided to allocate memory for your variable.)

Now use the Step Over command to step over line 13:

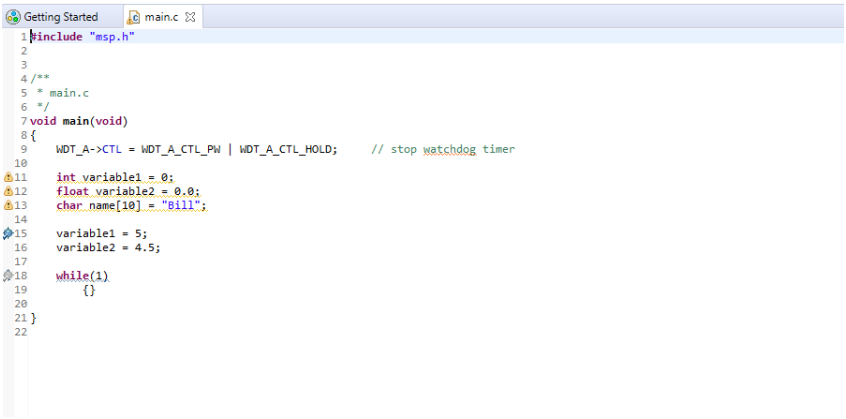



Now you should see this:



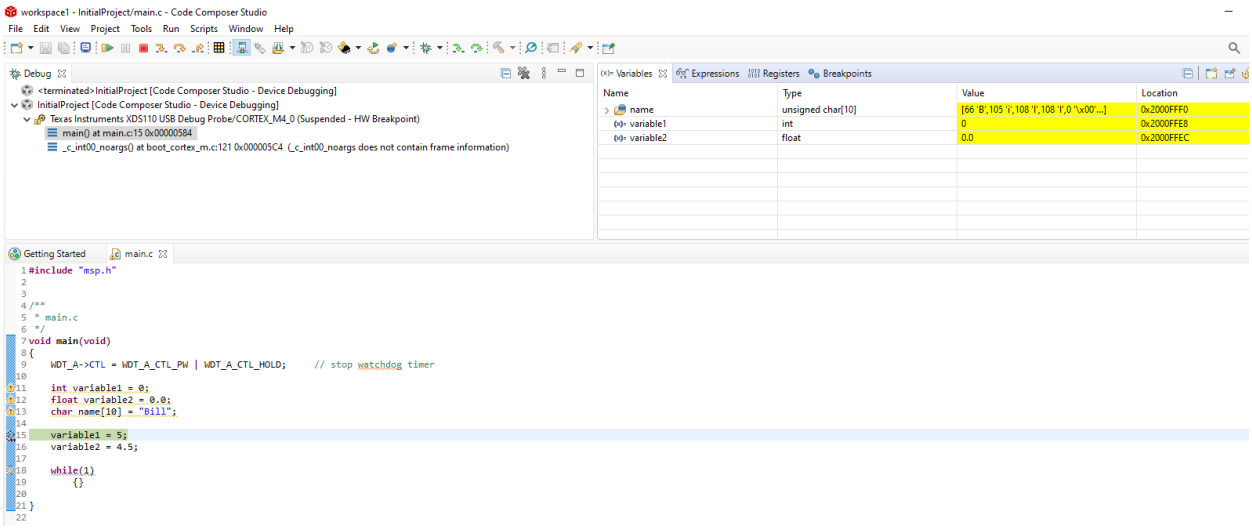

Notice the value has changed to 5.

Let’s explore some other variable types. Change the code in main.c to look like this:



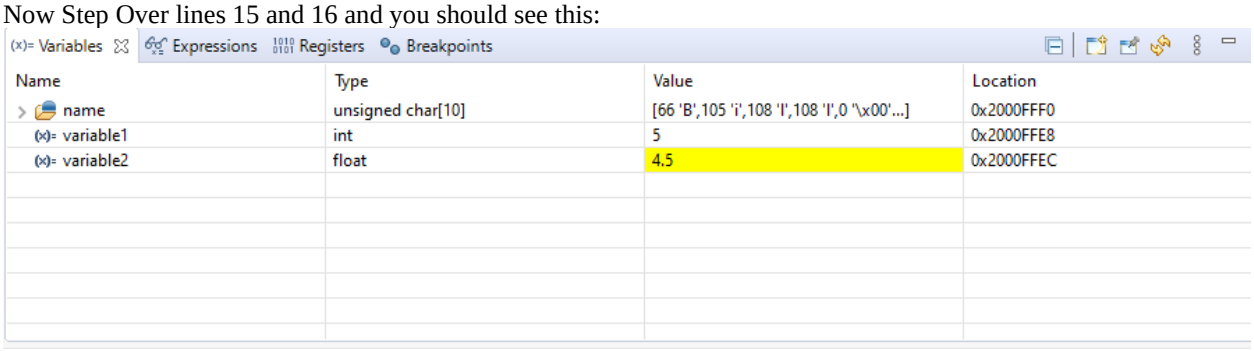
This introduces two new types of variable, a float and character array. The float will hold values that have values after the decimal point, and the character array holds character values. In this case you've allocated up to 10 characters for the name array. Important to note is the NULL character, of the 0 value, must always be at the end of an array of characters, so if you want to store names of up to 10 characters in length, you'd actually need to allocate 11 characters for the array.

Now compile and upload the code. If you haven't changed it your breakpoint should still be at the variable = 5; statement, so the Resume the code and it should stop at that line. You should see something similar to this:




Notice all of our variables are shown in the Variables space. Each has a type, and an address. If you look closely, you'll see that each character in the name array is stored as a numeric value as represented by the ASCII table. You'll also note that after the last "l" of the name is a NULL character to indicate that this particular name stops with the fourth character.

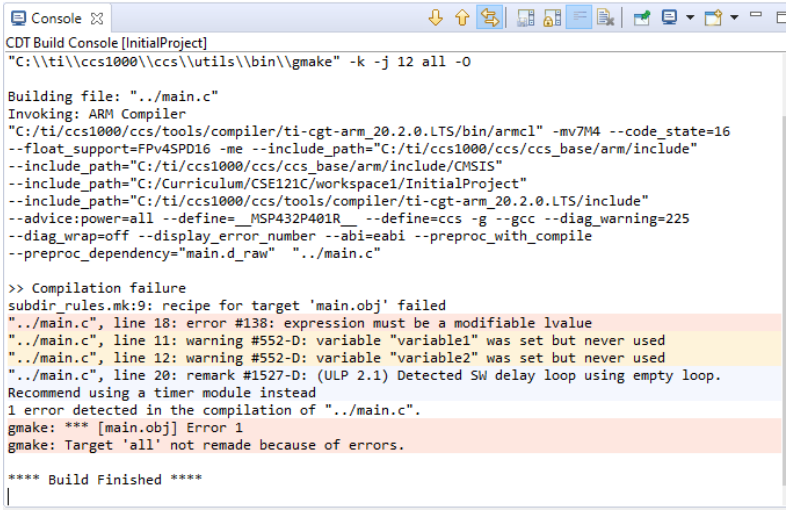
Now Step Over lines 15 and 16 and you should see this:



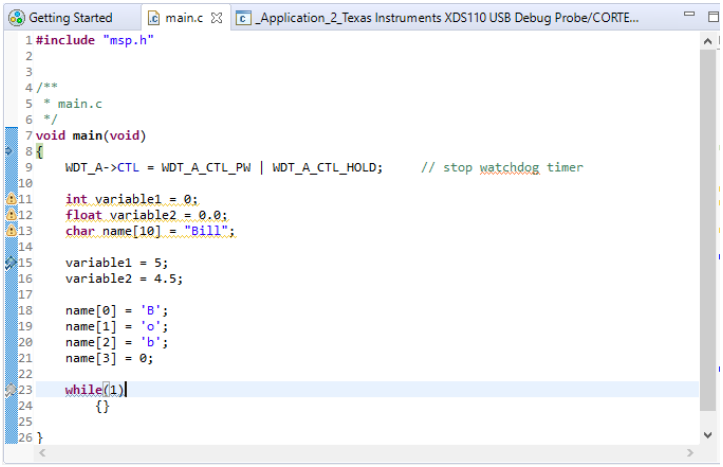

Now the obvious question is why didn't you add a statement like this:

name = "Bob";

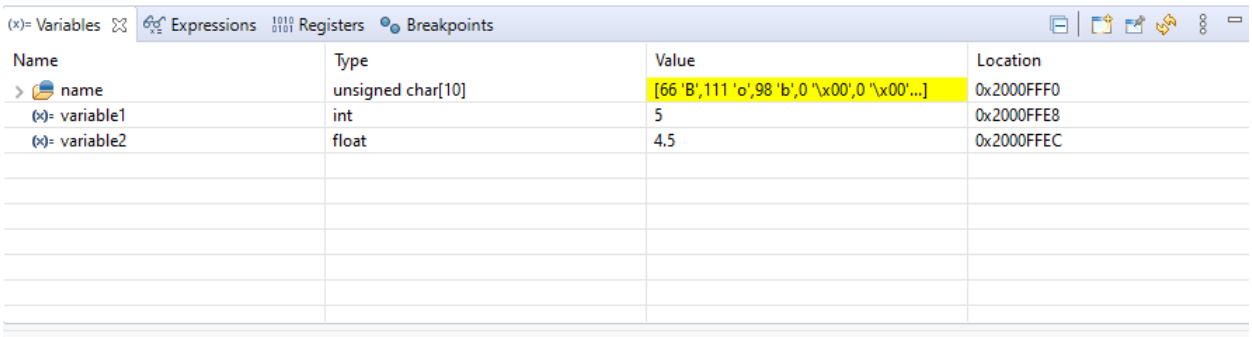
And the answer is this would have caused a compile error, like this:




Character arrays are a bit tricky, in C you can't just use the assignment operator to change them. You can either write individual characters, like this:




Now when you run and step through the program you should see this in the variable space:



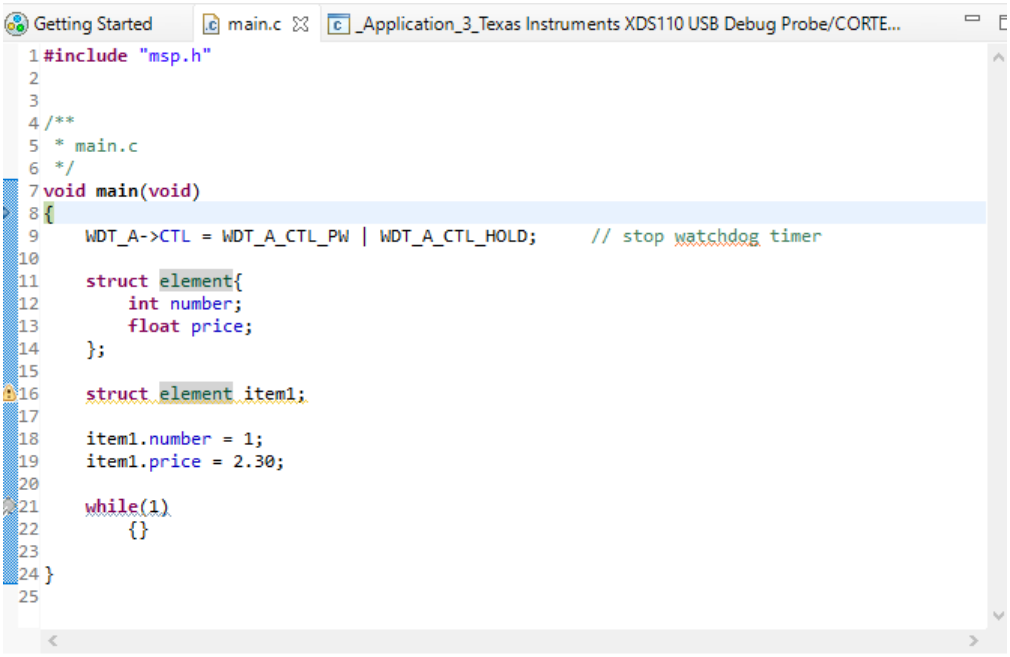
Notice line 21. It is important that you end your character array with the NULL character (value 0) or the computer won't know how many valid characters the array contains.

The Code Composer compiler supports a significant number of other standard data types, including these:

int1
Defines a 1 bit number
int8
Defines an 8 bit number
int16
Defines a 16 bit number
int32
Defines a 32 bit number
char
Defines a 8 bit character
float
Defines a 32 bit floating point number
short
By default the same as int1
Int
By default the same as int8
long
By default the same as int16
void
Indicates no specific type

## Combining Data Types using Data Structures

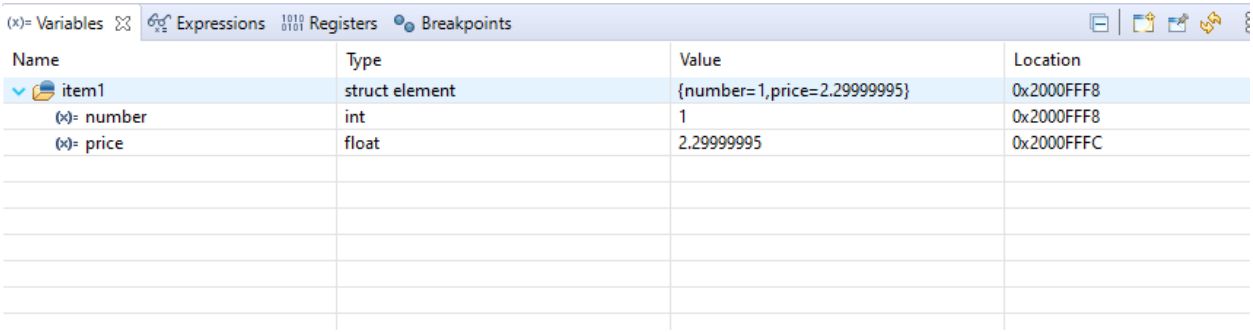
There are times when a data item might have more than one number associated with it. For example, you might be building an inventory of electronic parts. You might want to capture not only the number of parts available, but also the price that each part cost. To do this you can create a new data structure that contains both elements. To do this create the following code:



The struct keyword tells the compiler that you are going to create a combined data type. In this case it is going to have two elements, an integer with the name number, and a float with the name price. Now the struct key work used like this does not create a variable, it simply creates a definition of a type you can use later.

The struct element item1 creates a combined variable of type element with the name item1. Now to use the variable you will use the . notation. To set the number of item1 you will access it using item1.number and to access the price you'll use item1.price.

Compile and run this code. Put a breakpoint on line 18. Then Resume the code, and step over lines 18 and 19. In the variable selection click on the arrow key just to the left of the item1 variable name, and you should be able to expand the item to see its individual parts.




| Name | Type | Value | Location |
|---|---|---|---|
| item1 | struct element | {number=1,price=2.29999995} | 0x2000FFF8 |
| number | int | 1 | 0x2000FFF8 |
| price | float | 2.29999995 | 0x2000FFFC |

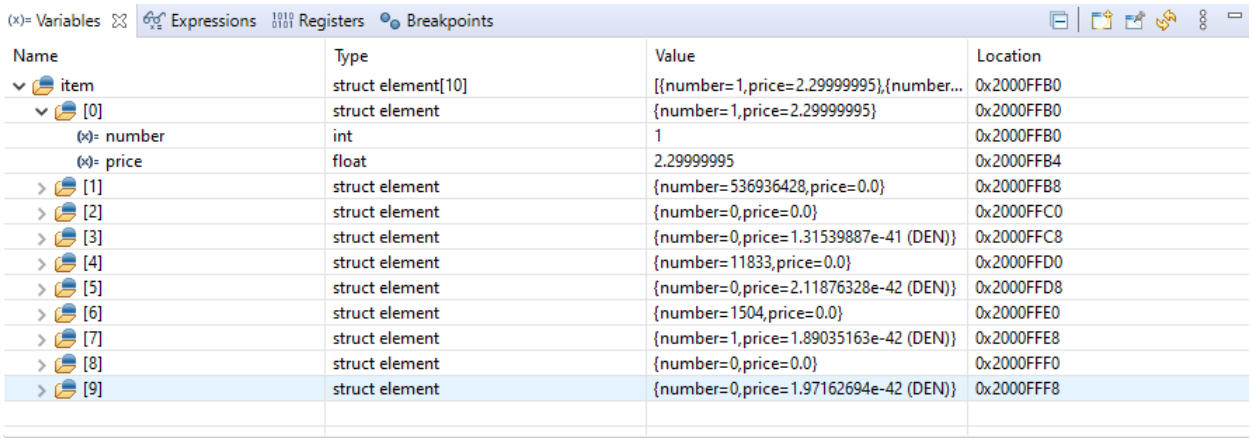
Interesting to note that price is not stored as exactly 2.30. It is stored as closely as it can get based on the accuracy of the storage location.

Now an inventory of a single item wouldn't make much sense, So let's create an array of items:

main.c

```
1  #include "msp.h"
2
3
4  /**
5   * main.c
6   */
7  void main(void)
8  {
9      WDT_A->CTL = WDT_A_CTL_PW | WDT_A_CTL_HOLD;     // stop watchdog timer
10
11     struct element{
12         int number;
13         float price;
14     };
15
16     struct element item[10];
17
18     item[0].number = 1;
19     item[0].price = 2.30;
20
21     while(1)
22         {}
23
24 }
25
```

Now compile and debug the program. If you step over to line 21 and look at the variables you should see something like this:

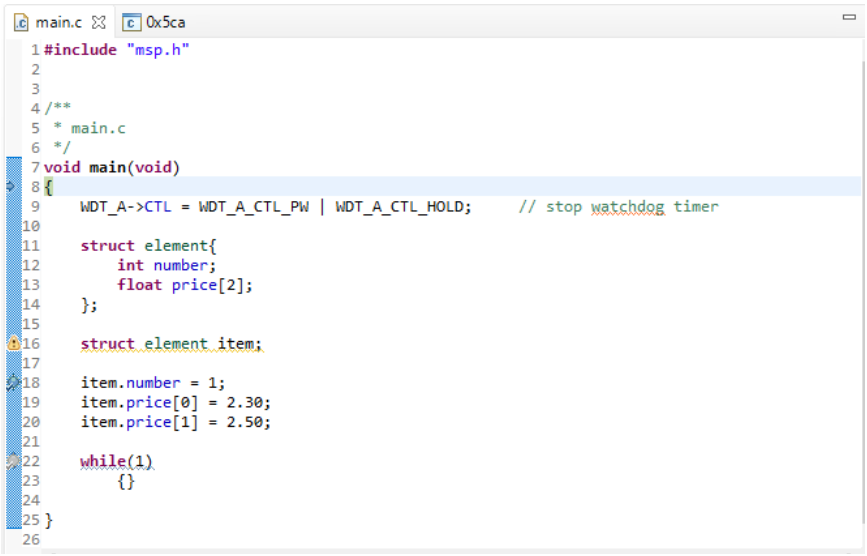


Variables | Expressions | Registers | Breakpoints

| Name | Type | Value | Location |
|---|---|---|---|
| item | struct element[10] | [{number=1,price=2.29999995},{number... | 0x2000FFB0 |
| [0] | struct element | {number=1,price=2.29999995} | 0x2000FFB0 |
| number | int | 1 | 0x2000FFB0 |
| price | float | 2.29999995 | 0x2000FFB4 |
| [1] | struct element | {number=536936428,price=0.0} | 0x2000FFB8 |
| [2] | struct element | {number=0,price=0.0} | 0x2000FFC0 |
| [3] | struct element | {number=0,price=1.31539887e-41 (DEN)} | 0x2000FFC8 |
| [4] | struct element | {number=11833,price=0.0} | 0x2000FFD0 |
| [5] | struct element | {number=0,price=2.11876328e-42 (DEN)} | 0x2000FFD8 |
| [6] | struct element | {number=1504,price=0.0} | 0x2000FFE0 |
| [7] | struct element | {number=1,price=1.89035163e-42 (DEN)} | 0x2000FFE8 |
| [8] | struct element | {number=0,price=0.0} | 0x2000FFF0 |
| [9] | struct element | {number=0,price=1.97162694e-42 (DEN)} | 0x2000FFF8 |

The first element has been set. Notice the other elements are not zero, but rather whatever the memory happened to hold when it was allocated. Important to note is that the C programming language does not initialize variables. This can cause a real problem when running programs if you don't initialize the variables as the actual execution of the code will seem random, when really it is simply because you are relying on memory that wasn't cleaned up from the last time it was used.

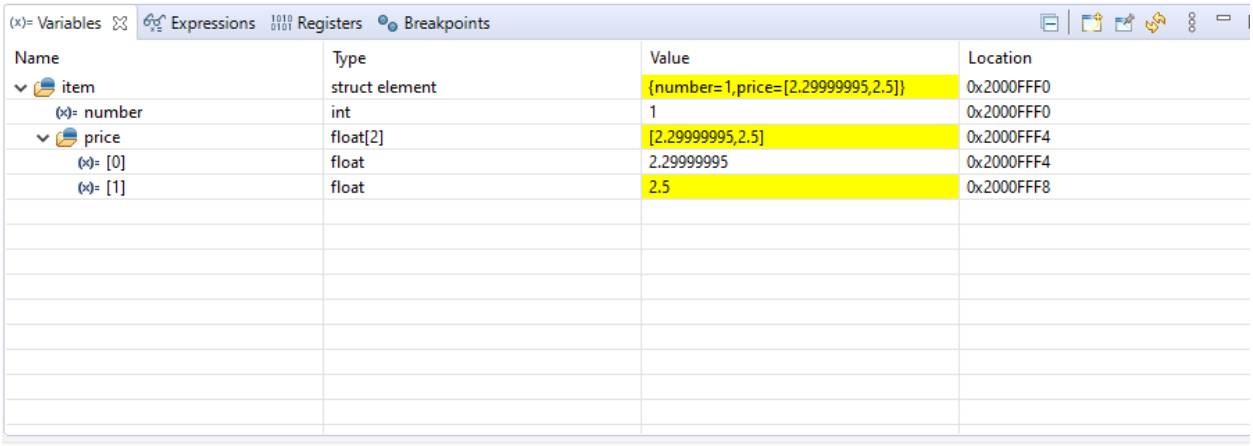
Also notice where the [] are in the array access. It is after the item, not after the entire item.number, as item is the array. This is important as you can also declare arrays as a part of the structure, like this:



main.c | 0x5ca

```
1  #include "msp.h"
2
3
4  /**
5   * main.c
6   */
7  void main(void)
8  {
9      WDT_A->CTL = WDT_A_CTL_PW | WDT_A_CTL_HOLD;     // stop watchdog timer
10
11     struct element{
12         int number;
13         float price[2];
14     };
15
16     struct element item;
17
18     item.number = 1;
19     item.price[0] = 2.30;
20     item.price[1] = 2.50;
21
22     while(1)
23         {}
24
25 }
26
```

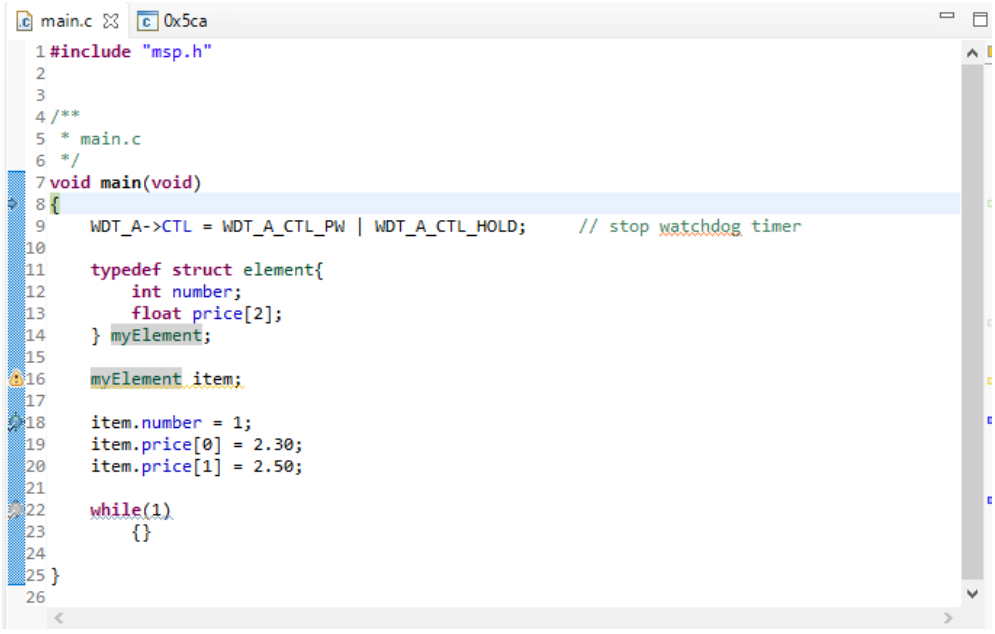
You might want to do this if you wanted to capture the max and min price of the item. In this case, because the array is part of the structure, you put the [] after the item.price, indicating that price is the array. Build and debug this code, step over the assignment operators, and you should see this in the variable space:




| Name | Type | Value | Location |
|---|---|---|---|
| item | struct element | {number=1,price=[2.29999995,2.5]} | 0x2000FFF0 |
| number | int | 1 | 0x2000FFF0 |
| price | float[2] | [2.29999995,2.5] | 0x2000FFF4 |
| [0] | float | 2.29999995 | 0x2000FFF4 |
| [1] | float | 2.5 | 0x2000FFF8 |

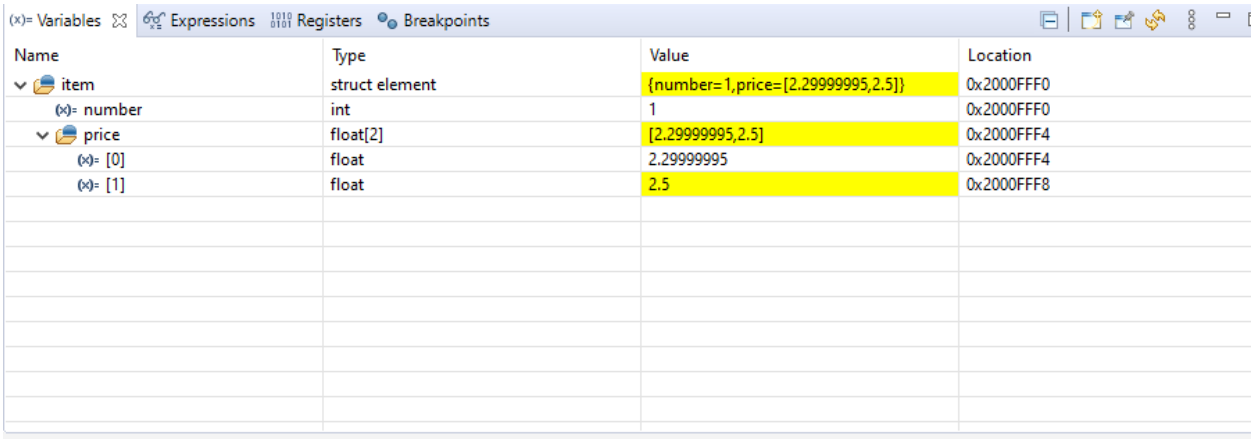
## Creating Your Own Data Types using typedef

You can also create your own data types using typedef to make your programming easier to read. This is often used in conjunction with the struct command. If you change your code to look like this:




```
1 #include "msp.h"
2
3
4 /**
5  * main.c
6  */
7 void main(void)
8 {
9     WDT_A->CTL = WDT_A_CTL_PW | WDT_A_CTL_HOLD;     // stop watchdog timer
10
11    typedef struct element{
12        int number;
13        float price[2];
14    } myElement;
15
16    myElement item;
17
18    item.number = 1;
19    item.price[0] = 2.30;
20    item.price[1] = 2.50;
21
22    while(1)
23        {}
24
25 }
26
```

You will now notice that you've create a new type called myElement. It is associated with the data structure element, and you can now use it as you would any data type to declare new variables. Now build and run this code. Resume the code and step over the assignments and you should see this in the variable space:




| Name | Type | Value | Location |
|---|---|---|---|
| item | struct element | {number=1,price=[2.29999995,2.5]} | 0x2000FFF0 |
| number | int | 1 | 0x2000FFF0 |
| price | float[2] | [2.29999995,2.5] | 0x2000FFF4 |
| [0] | float | 2.29999995 | 0x2000FFF4 |
| [1] | float | 2.5 | 0x2000FFF8 |