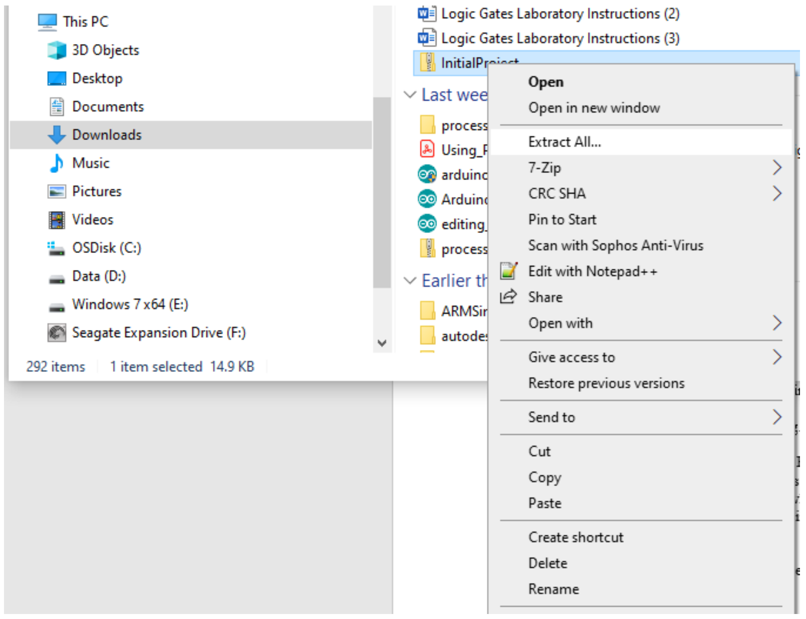
Introduction to Conditionals and Loops

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 functions and libraries 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:.

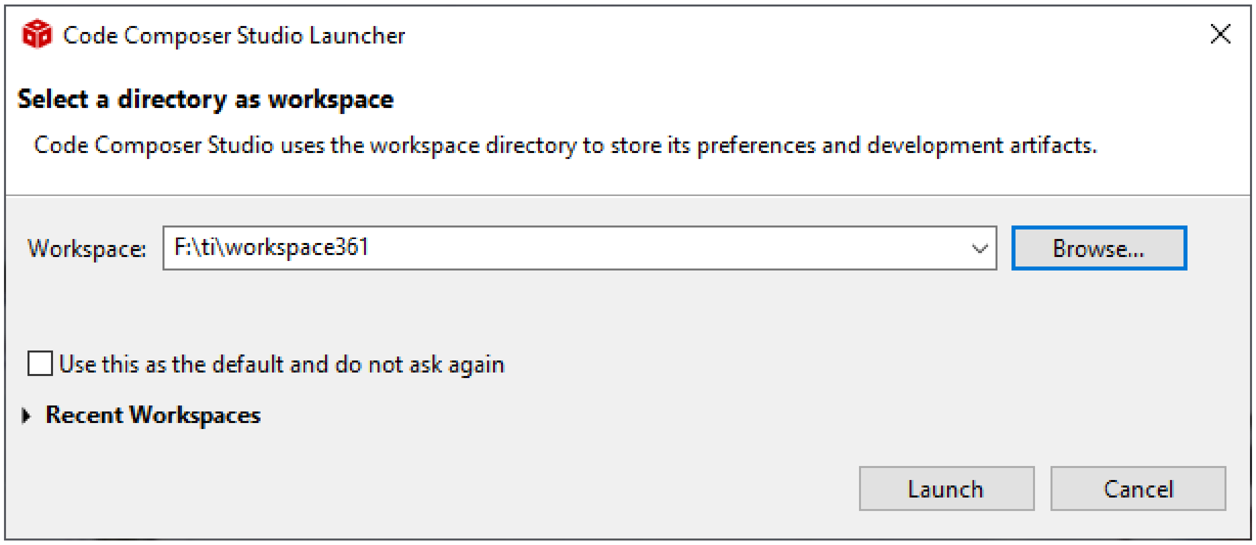


When you have the file unarchived, you can now import it into Code Composer.

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.


Code Composer Studio Launcher

Select a directory as workspace

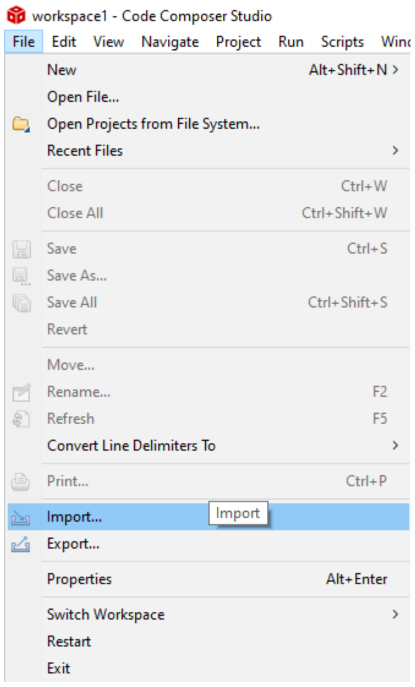
Code Composer Studio uses the workspace directory to store its preferences and development artifacts.

Workspace: F:\ti\workspace361

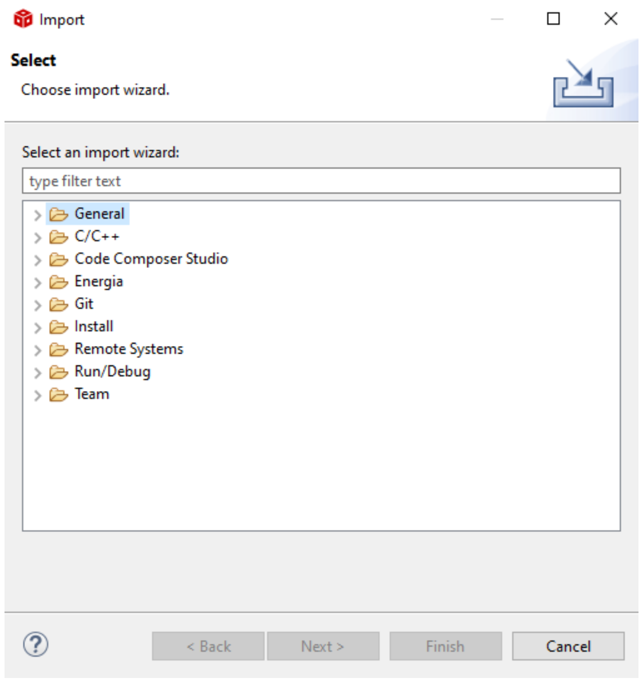
☐ Use this as the default and do not ask again

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.)

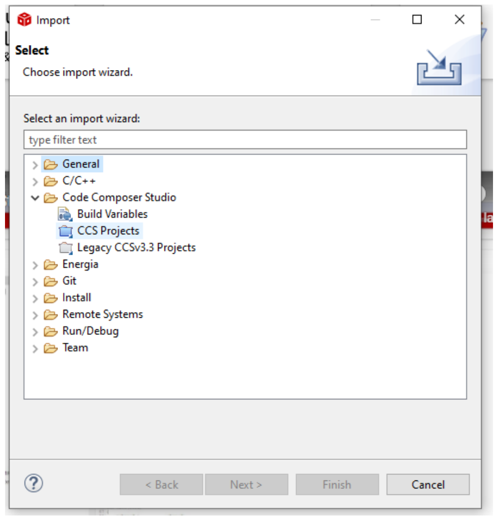
If you haven't already connect your MSP432P401R to the computer via a USB cable. Now let's import the project into Code Composer. To do this select File->Import



Then you should see this dialogue box:



Now select Code Composer Studio, then CCS Projects:



Import

Select

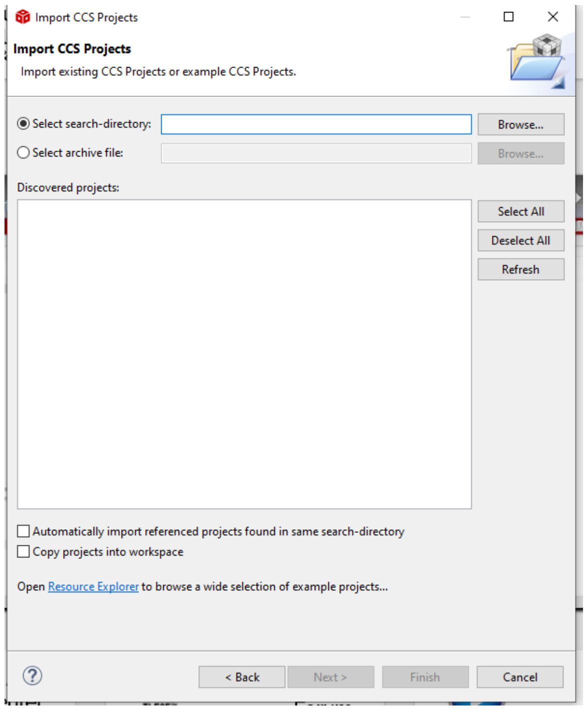
Choose import wizard.

Select an import wizard:

type filter text

- General
- C/C++
- Code Composer Studio
  - Build Variables
  - CCS Projects
  - Legacy CCSv3.3 Projects
- Energia
- Git
- Install
- Remote Systems
- Run/Debug
- Team

And you should see this dialogue box:



Import CCS Projects

Import CCS Projects

Import existing CCS Projects or example CCS Projects.

(●) Select search-directory:

( ) Select archive file:

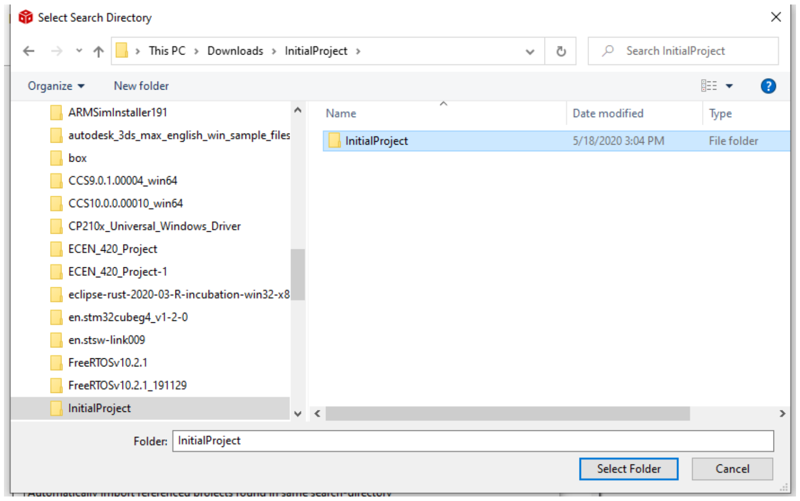
Discovered projects:

☐ Automatically import referenced projects found in same search-directory

☐ Copy projects into workspace

Open Resource Explorer to browse a wide selection of example projects...

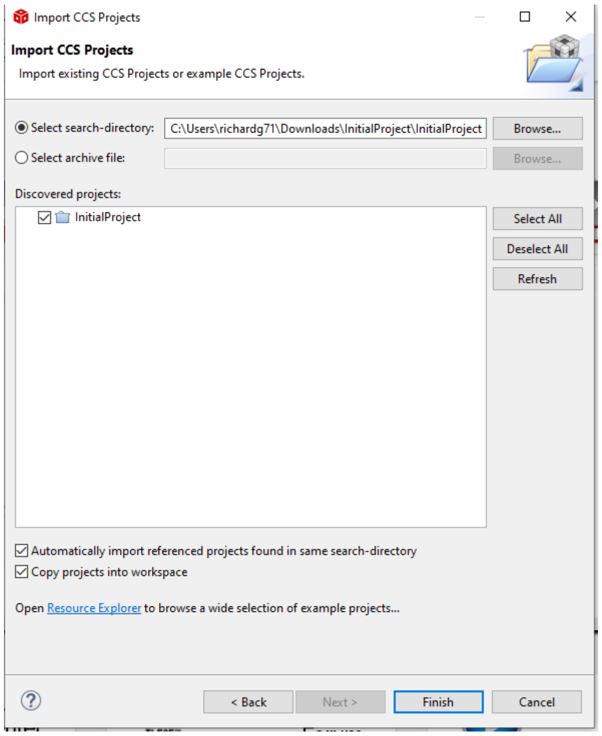
Browse to the directory where you unzip the archive. In my case it was my Downloads directory:



Name	Date modified	Type
InitialProject	5/18/2020 3:04 PM	File folder

Folder: InitialProject

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



Import CCS Projects

Import existing CCS Projects or example CCS Projects.

(●) Select search-directory: C:\Users\richardg71\Downloads\InitialProject\InitialProject

( ) Select archive file:

Discovered projects:

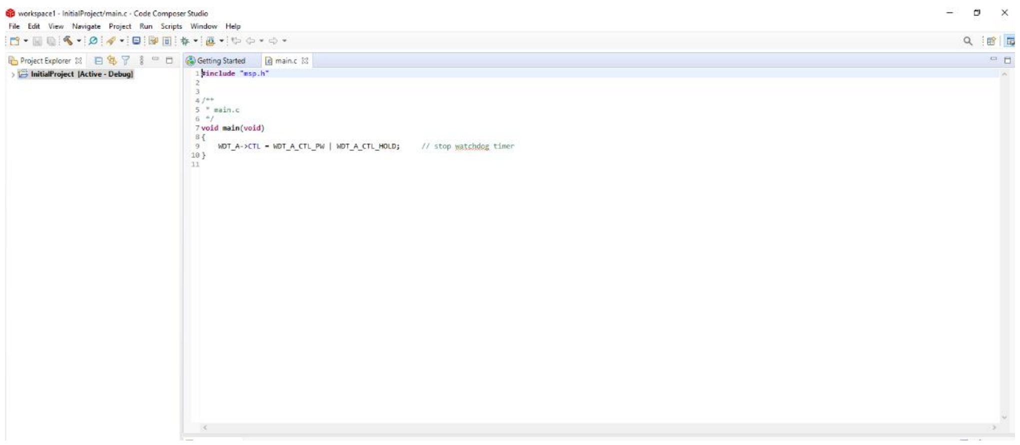
☑ InitialProject

☑ Automatically import referenced projects found in same search-directory

☑ Copy projects into workspace

Open Resource Explorer to browse a wide selection of example projects...

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


workspace1 - InitialProject/main.c - Code Composer Studio
File Edit View Navigate Project Run Scripts Window Help
Project Explorer
InitialProject [Active - Debug]
Getting Started
main.c

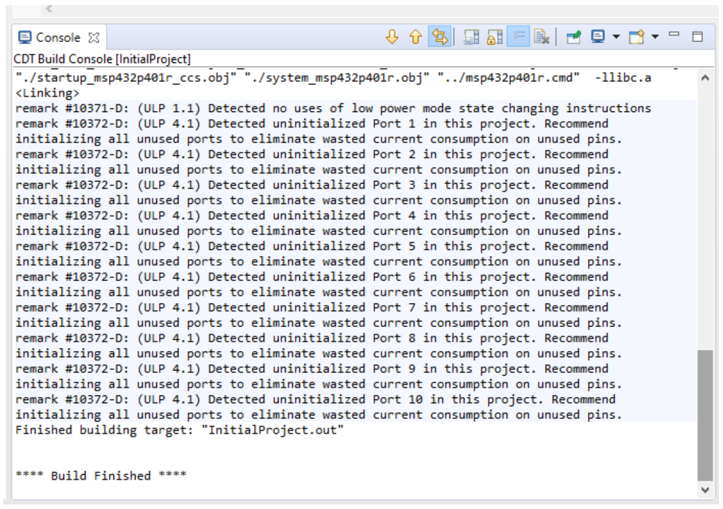
```
#include "msp.h"


/**
 * main.c
 */
void main(void)
{
    WDT_A->CTL = WDT_A_CTL_PW | WDT_A_CTL_HOLD;     // stop watchdog timer
}

```

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:



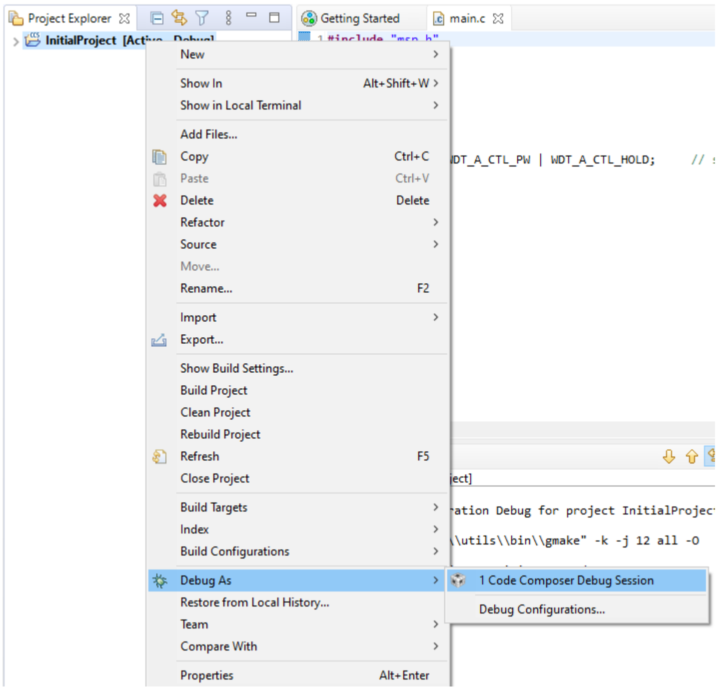
```
"./startup_msp432p401r_ccs.obj" "./system_msp432p401r.obj" "../msp432p401r.cmd"  -llibc.a
<Linking>
remark #10371-D: (ULP 1.1) Detected no uses of low power mode state changing instructions
remark #10372-D: (ULP 4.1) Detected uninitialized Port 1 in this project. Recommend
initializing all unused ports to eliminate wasted current consumption on unused pins.
remark #10372-D: (ULP 4.1) Detected uninitialized Port 2 in this project. Recommend
initializing all unused ports to eliminate wasted current consumption on unused pins.
remark #10372-D: (ULP 4.1) Detected uninitialized Port 3 in this project. Recommend
initializing all unused ports to eliminate wasted current consumption on unused pins.
remark #10372-D: (ULP 4.1) Detected uninitialized Port 4 in this project. Recommend
initializing all unused ports to eliminate wasted current consumption on unused pins.
remark #10372-D: (ULP 4.1) Detected uninitialized Port 5 in this project. Recommend
initializing all unused ports to eliminate wasted current consumption on unused pins.
remark #10372-D: (ULP 4.1) Detected uninitialized Port 6 in this project. Recommend
initializing all unused ports to eliminate wasted current consumption on unused pins.
remark #10372-D: (ULP 4.1) Detected uninitialized Port 7 in this project. Recommend
initializing all unused ports to eliminate wasted current consumption on unused pins.
remark #10372-D: (ULP 4.1) Detected uninitialized Port 8 in this project. Recommend
initializing all unused ports to eliminate wasted current consumption on unused pins.
remark #10372-D: (ULP 4.1) Detected uninitialized Port 9 in this project. Recommend
initializing all unused ports to eliminate wasted current consumption on unused pins.
remark #10372-D: (ULP 4.1) Detected uninitialized Port 10 in this project. Recommend
initializing all unused ports to eliminate wasted current consumption on unused pins.
Finished building target: "InitialProject.out"


**** Build Finished ****
```

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.

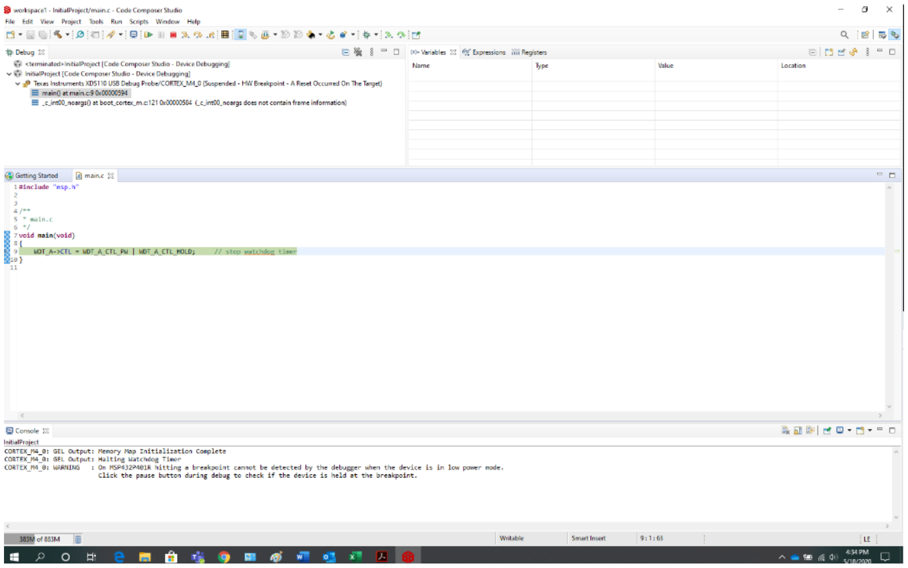
The C programming language is a compiled language. This means that a special program called a compiler takes your program and turns it into machine code. When you want to run your program you don't need any other program, your program is in machine code in a run-able state.

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



```
1 #include "msp.h"
DT_A_CTL_PW | WDT_A_CTL_HOLD;     // s
ject]
ration Debug for project InitialProjec
\utils\\bin\\gmake" -k -j 12 all -O
```

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:

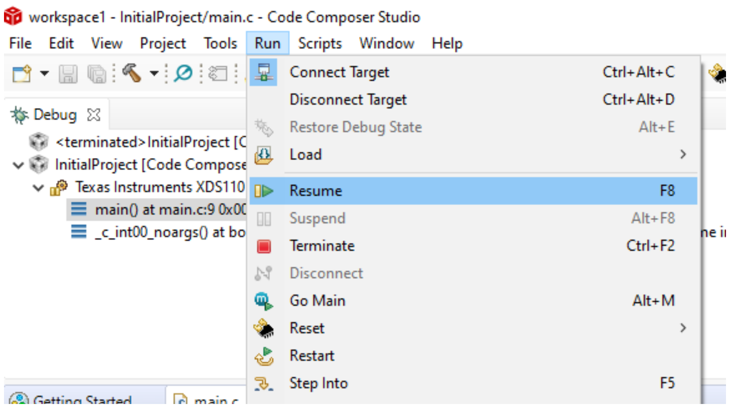


```
<terminated>InitialProject [Code Composer Studio - Device Debugging]
InitialProject [Code Composer Studio - Device Debugging]
  Texas Instruments XDS110 USB Debug Probe/CORTEX_M4_0 (Suspended - HW Breakpoint - A Reset Occurred On The Target)
    main() at main.c:9 0x00000594
    _c_int00_noargs() at boot_cortex_m.c:121 0x00000584  (_c_int00_noargs does not contain frame information)
```

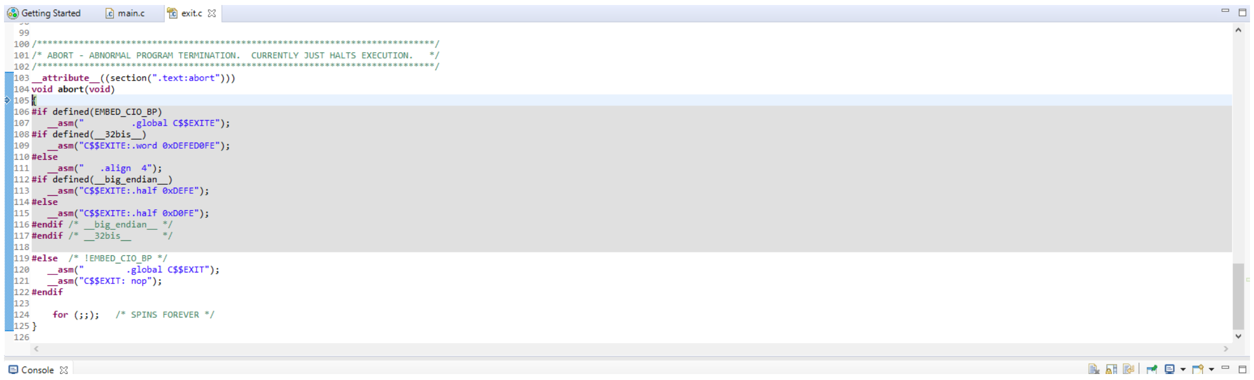
```c
#include "msp.h"


/**
 * main.c
 */
void main(void)
{
    WDT_A->CTL = WDT_A_CTL_PW | WDT_A_CTL_HOLD;     // stop watchdog timer
}
```

```
InitialProject
CORTEX_M4_0: GEL Output: Memory Map Initialization Complete
CORTEX_M4_0: GEL Output: Halting Watchdog Timer
CORTEX_M4_0: WARNING   : On MSP432P401R hitting a breakpoint cannot be detected by the debugger when the device is in low power mode.
                         Click the pause button during debug to check if the device is held at the breakpoint.
```

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


workspace1 - InitialProject/main.c - Code Composer Studio
File Edit View Project Tools Run Scripts Window Help
Debug
<terminated>InitialProject [C
InitialProject [Code Compose
Texas Instruments XDS110
main() at main.c:9 0x00
_c_int00_noargs() at bo
Connect Target Ctrl+Alt+C
Disconnect Target Ctrl+Alt+D
Restore Debug State Alt+E
Load
Resume F8
Suspend Alt+F8
Terminate Ctrl+F2
Disconnect
Go Main Alt+M
Reset
Restart
Step Into F5
Getting Started
main.c

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



```

/****************************************************************************/
/* ABORT - ABNORMAL PROGRAM TERMINATION.  CURRENTLY JUST HALTS EXECUTION.   */
/****************************************************************************/
__attribute__((section(".text:abort")))
void abort(void)
{
#if defined(EMBED_CIO_BP)
   __asm("         .global C$$EXITE");
#if defined(__32bis__)
   __asm("C$$EXITE:.word 0xDEFED0FE");
#else
   __asm("   .align  4");
#if defined(__big_endian__)
   __asm("C$$EXITE:.half 0xDEFE");
#else
   __asm("C$$EXITE:.half 0xD0FE");
#endif /* __big_endian__ */
#endif /* __32bis__      */

#else  /* !EMBED_CIO_BP */
   __asm("        .global C$$EXIT");
   __asm("C$$EXIT: nop");
#endif

    for (;;);   /* SPINS FOREVER */
}

```

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

Conditionals in C

In the last lab you learned the basics of structures and pointers. This lab will introduce you to some basic concepts that you should already be familiar with if you have used any programming language; conditionals and Loops. Let’s start with conditionals.

The most basic of conditionals using if statements is shown in this code:

```
#include "msp.h"


/**
 * main.c
 */
void main(void)
{

    WDT_A->CTL = WDT_A_CTL_PW | WDT_A_CTL_HOLD;     // stop watchdog timer

    int id = 1;
    int myId = 0;
    if (id == 1)
        myId = id;

   return;

}
```

Notice the use of the == sign to check to see if two values are equal. We cannot use the = sign, it has already by used as the assignment operator.

The allowed Boolean checks in the if statement are standard:

>	greater than
<	less than
>=	greater than or equal
<=	less than or equal
==	equal to
!=	not equal to

As in most programming languages an optional else class is also allowed, as shown in this code:

```
#include "msp.h"


/**
 * main.c
 */
void main(void)
{

    WDT_A->CTL = WDT_A_CTL_PW | WDT_A_CTL_HOLD;     // stop watchdog timer

    int id = 1;
    int myId = 0;

    if (id == 1)
        myId = id;
    else
        myId = 2;

   return;

}
```

Often there are times when you want more than one command statement to be part of the if statement structure. In the C programming construct such groups of statements are surrounded by {}. Here is some example code:

```
#include "msp.h"


/**
 * main.c
 */
void main(void)
```

```
{

    WDT_A->CTL = WDT_A_CTL_PW | WDT_A_CTL_HOLD;     // stop watchdog timer

    int id = 1;
    int myId = 0;
    int myId1 = 0;

    if (id == 1)
    {
        myId = id;
        myId1 = id;
    }
    else
    {
        myId = 2;
        myId1 = 1;
    }

   return;

}
```

You can also have the Boolean expression in an if statement contain more than one clause. Here is an example:

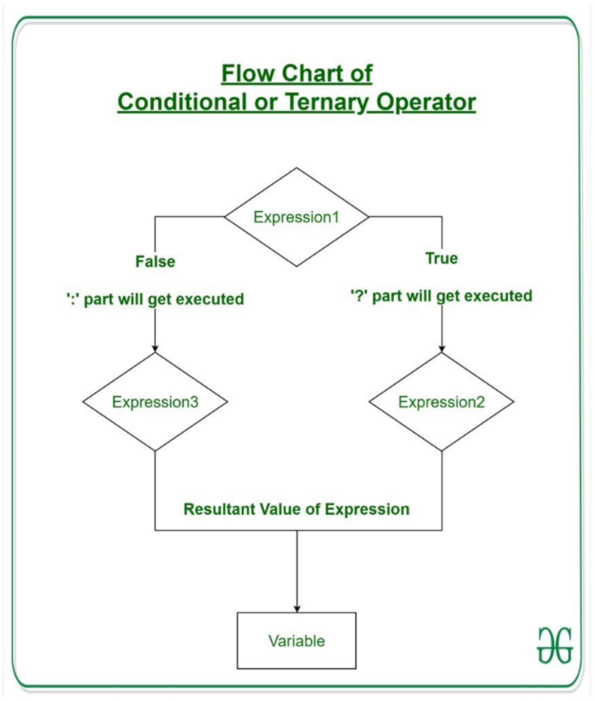
```
#include "msp.h"


/**
 * main.c
 */
void main(void)
{

    WDT_A->CTL = WDT_A_CTL_PW | WDT_A_CTL_HOLD;     // stop watchdog timer

    int id = 1;
    int myId = 0;
    int myId1 = 0;

    if (id == 1 && myId == 0 && myId1 == 0)
    {
        myId = id;
        myId1 = id;
    }
    else
    {
        myId = 2;
        myId1 = 1;
    }

   return;

}
```

The && is equivalent to the AND operation of the two expressions. The || is the equivalent to the OR operation. If statements can also be nested inside of if statements. Just to note, these use the values and the AND operation. If you wish to use a Bitwise AND or a Bitwise OR, you can use a single & or a single | operator.

Finally, there is also a unique way to write a special if statement in C so that it takes up only one line of code. It is called the Ternary Operator. Here is some code that illustrates using the ternary operator:

```
#include "msp.h"


/**
 * main.c
 */
void main(void)
{

    WDT_A->CTL = WDT_A_CTL_PW | WDT_A_CTL_HOLD;     // stop watchdog timer

    int id = 1;
    int myId = 2;
    int myId1 = 0;

    myId1 = (id > myId) ? id:  myId;

   return;

}
```

The ternary statement has three parts. The first is the assignment operator, the second is the logical expression, and the third is the set of statement that can be executed and the values that can be returned. Here is a diagram that shows the execution flow of the ternary statement:


Flow Chart of
Conditional or Ternary Operator
Expression1
False
True
':' part will get executed
'?' part will get executed
Expression3
Expression2
Resultant Value of Expression
Variable

Loops in C

As in most computer languages, C comes with three types of loops; the for loop, the while loop, and the do-while loop. Here is an overview of the three loops:

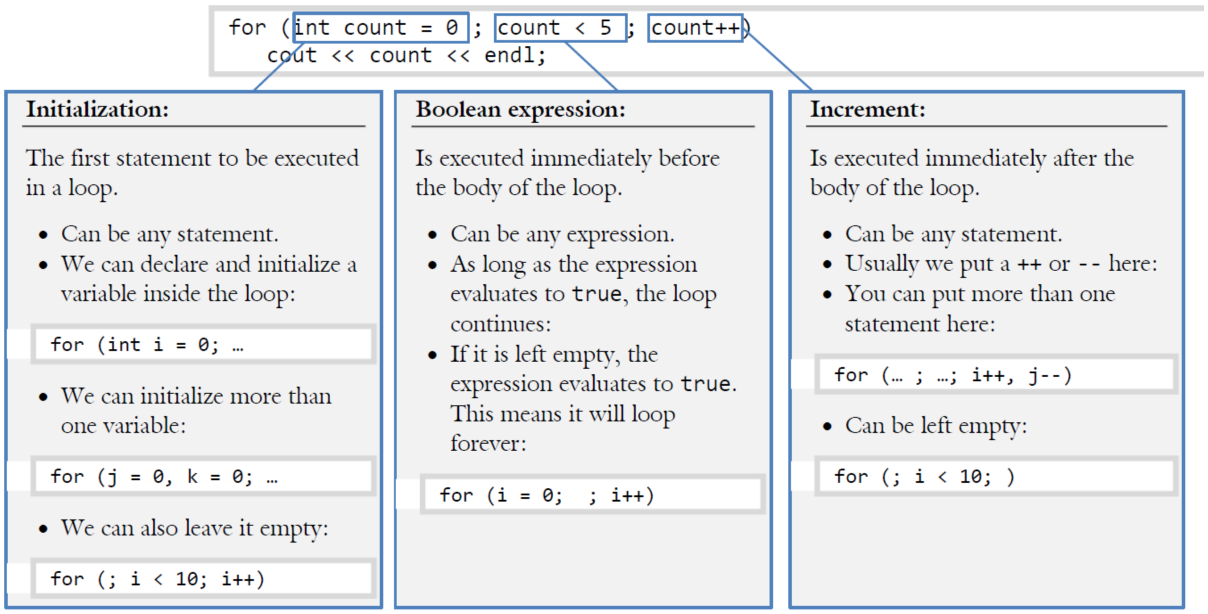
while	do-while	for
A WHILE loop is good for repeating through a given block of code multiple times.	Same as WHILE except we always execute the body of the loop at least once.	Designed for counting, usually meaning we know where we start, where we end and what changes.

`{`
`   while (x > 0)`
`   {`
`      x--;`
`      cout << x << endl;`
`   }`
`}`
	`{`
	`   do`
	`   {`
	`      x--;`
	`      cout << x << endl;`
	`   }`
	`   while (x > 0);`
	`}`
		`{`
		`   for (x = 10;`
		`        x > 0;`
		`        x--)`
		`   {`
		`      cout << x << endl;`
		`   }`
		`}`

The for loop syntax structure is fairly simple, as shown in this code example:

```
#include "msp.h"


/**
 * main.c
 */
void main(void)
{

    WDT_A->CTL = WDT_A_CTL_PW | WDT_A_CTL_HOLD;     // stop watchdog timer

    int i;
    int value = 0;

    for (i = 0; i < 5; i++)
        value = i;

   return;

}
```

The format of the for statement is similar to other programming languages. In the c programming language there is a great deal of flexibility in what is placed in the for loop. Here is the overall context of the statement:



```
for (int count = 0; count < 5; count++)
   cout << count << endl;
```

Initialization:

The first statement to be executed in a loop.

- Can be any statement.
- We can declare and initialize a variable inside the loop:

```
for (int i = 0; …
```

- We can initialize more than one variable:

```
for (j = 0, k = 0; …
```

- We can also leave it empty:

```
for (; i < 10; i++)
```

Boolean expression:

Is executed immediately before the body of the loop.

- Can be any expression.
- As long as the expression evaluates to `true`, the loop continues:
- If it is left empty, the expression evaluates to `true`. This means it will loop forever:

```
for (i = 0;  ; i++)
```

Increment:

Is executed immediately after the body of the loop.

- Can be any statement.
- Usually we put a `++` or `--` here:
- You can put more than one statement here:

```
for (… ; …; i++, j--)
```

- Can be left empty:

```
for (; i < 10; )
```

The While loop, and the do-While loop, are both related. They use a statement to decide whether or not to execute the loop.

Perhaps the simplest loop is the WHILE statement. The WHILE loop will continue executing the body of the loop until the controlling Boolean expression evaluates to false. The syntax is:

```
while (<Boolean expression>)
   <body statement>;
```

As with the IF statement, we can always have more than one statement in the body of the loop by adding curly braces {}s:

```
while (<Boolean expression>)
  {
     <body statement1>;
     <body statement2>;
      ...
  }
```

The DO-WHILE loop is the same as the WHILE loop except the controlling Boolean expression is checked after the body of the loop is executed. This is called a trailing-condition loop, while the WHILE loop is a leading-condition loop.

As with the WHILE statement, the loop will continue until the controlling Boolean expression evaluates to false. The syntax is:

```
do
  <body statement>;
while (<Boolean expression>);
```

DO-WHILE loops are used far less frequently than the aforementioned WHILE loops. Those scenarios when the DO-WHILE loop would be the tool of choice center around the need to ensure the body of the loop gets executed at least once.

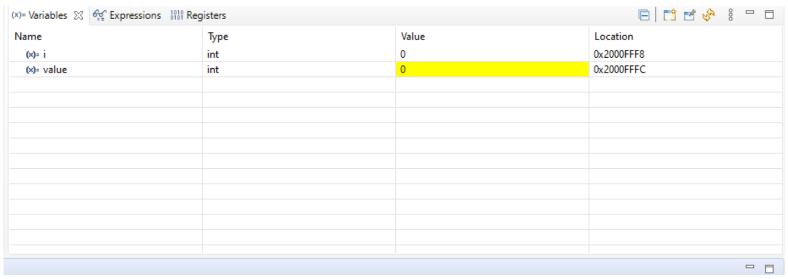
Debugging Loops using the Code Composer IDE

One of the nice features of the Code Composer IDE is the ability to step code. This is particularly helpful in debugging defects associated with loops. Let's take this example code:

```c
#include "msp.h"


/**
 * main.c
 */
void main(void)
{

    WDT_A->CTL = WDT_A_CTL_PW | WDT_A_CTL_HOLD;     // stop watchdog timer

    int i;
    int value = 0;

    for (i = 0; i < 5; i++)
        value = i;

   return;

}
```

This is a fairly simple loop, but let's assume that for some reason I thought the loop would start at i = 1 and go until i = 5. So now debug the code.

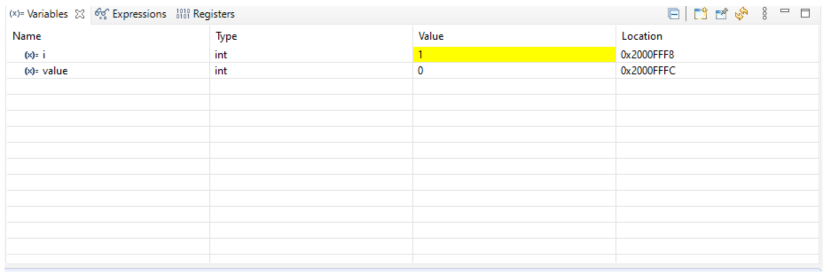
We're going to use the Step Into command (F5) to step through the program and watch the variables. Before the loop is begun the variables should look like this:



Name	Type	Value	Location
(x)= i	int	0	0x2000FFF8
(x)= value	int	0	0x2000FFFC

This makes sense, the loop hasn't started. Now hit the F5 key, and you will step into the loop. However, the variables values still stay the same. We can see the first defect, the loop doesn't start with the value 1, it starts with the value 0. To make the loop start at one you would need to change the initializer statement to i = 1;

Now hit the F5key again, and then once more. You should see this:



Name	Type	Value	Location
i	int	1	0x2000FFF8
value	int	0	0x2000FFFC

The value of i has now change to 1. You can continue to step through the loop, the number stored in value will change to 1, then to 2 and so on.

When you get to i = 5 you will notice the loop will no longer continue, but will exit. Thus you can tell that your loop will not include the value i = 5 in the execution. To make this happen you’ll need to change the Boolean expression that controls the loop to include the value 5 by using i <= 5.