

Debug

Last updated 6/19/23

Debug

- Easy Debug Tactics

- Print out intermediate information

```
printf("I reached this point");  
printf("foo = %i\n", foo);
```

- Break problems into pieces

```
foo = a | b << c * d++ - 3 / b % 6;
```

→

```
foo = d++;  
printf("foo = %i\n", foo);  
foo = c * d++ ;  
printf("foo = %i\n", foo);
```

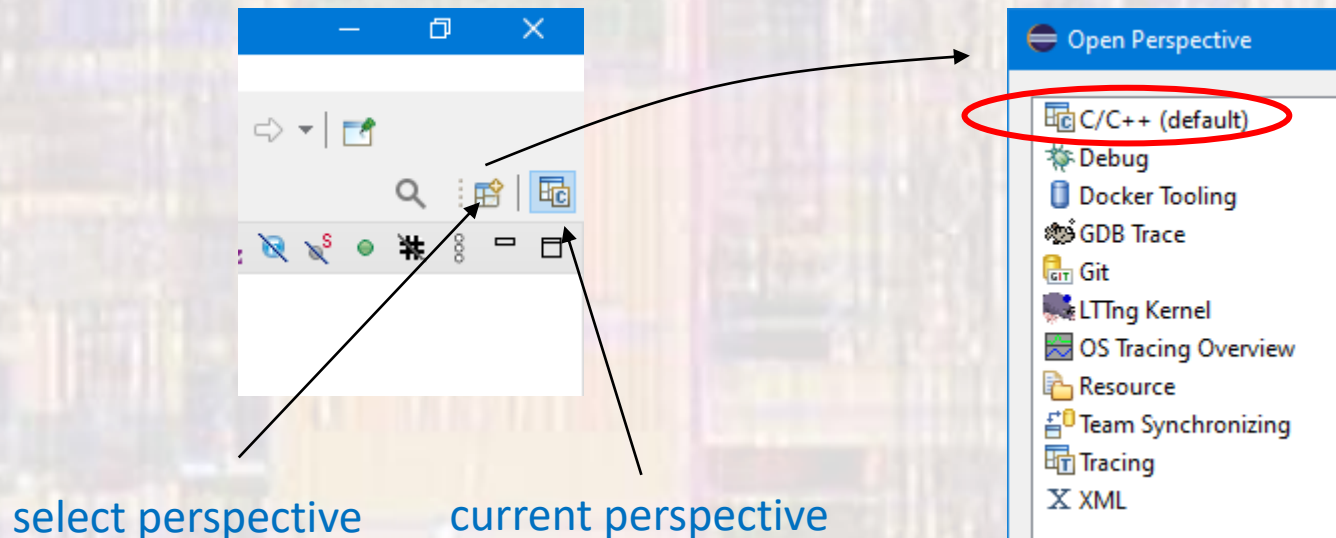
...

Debug

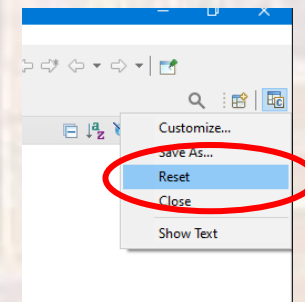
- Debugger
 - Most C tool chains include a debugger
 - The debugger allows
 - Stopping execution
 - Stepping line – by – line
 - Tracking variable values
 - Follow execution into and out of functions

Debug

- Debugger - Eclipse Perspectives
 - Eclipse has a series of pre-defined window configurations
 - Each configuration is optimized for a specific purpose

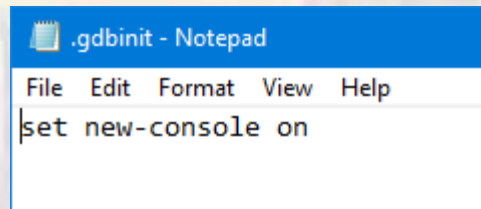


- If your windows get messed up
 - **rt-click** on the current perspective → **Reset**



Debug

- Eclipse Debugger – Work-around
 - The debugger does not work in the UI console window
 - To work-around this issue we will use an external terminal window
 - **rt-click** the project name **New** → **File**
 - **provide** the file name **.gdbinit**
 - In the opened file **type** **set new-console on**
 - **save**
 - Note – the file will not show up in the Project Explorer list



```
.gdbinit - Notepad
File Edit Format View Help
set new-console on
```

Debug

- Debugger Tool - Example program

```
/*
 * debug_demo.c
 *
 * Created on: Jan 24, 2021
 * Author: johnsontimoj
 */

#include <stdio.h>

void splash(void);
void read_input(int* intval_ptr, float* floatval_ptr, char* charval_ptr);
int ifelsefn(int val);
int casefn(int* intval_ptr, float* floatval_ptr);

int main(void){
    setbuf(stdout, NULL);

    int x;
    int y;
    char aa;
    char bb;
    float one;
    float two;

    x = 3;
    y = 4;
    aa = 'f';
    bb = 'g';
    one = 1.003;
    two = 2.222;

    printf("%i %i\n", x, y);

    // splash screen
    splash();

    // input values
    read_input(&x, &one, &aa);

    // ifelse function
    y = ifelsefn(x);

    // case function
    y = casefn(&x, &two);

    // port manipulation
    printf("%i %i\n", x, y);
    printf("%f %f\n", one, two);
    printf("%c %c\n", aa, bb);

    return 0;
} // end main
```

```
52
53@ //////////////////////////////////////////////////
54 // splash
55 //
56 // code to print splash screen
57 //
58 // input: none
59 // output - prints message to screen
60 // return - void
61 //////////////////////////////////////////////////
62@ void splash(void){
63     printf("\nProgram to demonstrate the debugger\n\n");
64     return;
65 } // end splash
66
67
68@ //////////////////////////////////////////////////
69 // read_input
70 //
71 // read in an int, a float, and a char
72 //
73 // inputs - none
74 // output - int/float/char via pointers
75 // return - void
76 //////////////////////////////////////////////////
77@ void read_input(int* intval_ptr, float* floatval_ptr, char* charval_ptr){
78     printf("Please enter an int, a float, and a character: \n");
79     scanf("%i %f %c", intval_ptr, floatval_ptr, charval_ptr);
80
81     return;
82 } // end read_input
83
84@ //////////////////////////////////////////////////
85 // ifelsefn
86 //
87 // selects an output for a given input
88 //
89 // inputs - int to select on
90 // output - none
91 // return - random value based on input
92 //////////////////////////////////////////////////
93@ int ifelsefn(int val){
94     int result;
95     if(val == 0)
96         result = 5;
97     else if(val == 4)
98         result = 9;
99     else if(val >=5)
100        result = 12;
101     else
102        result = -2;
103
104     return result;
105 } // end ifelsefn
106
```

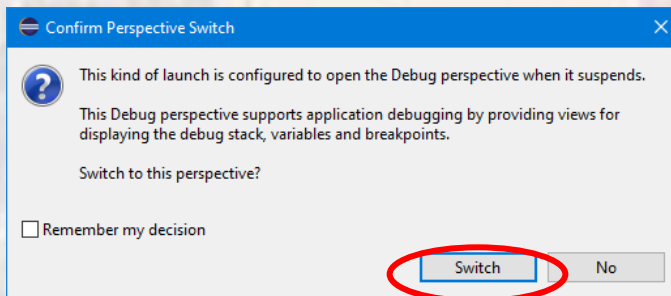
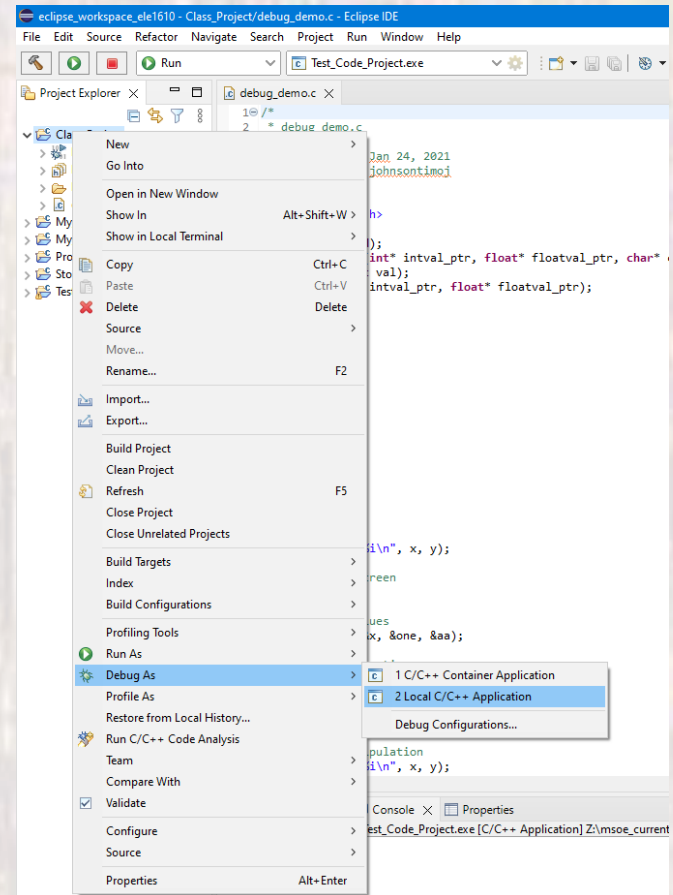
```
106
107@ //////////////////////////////////////////////////
108 // casefn
109 //
110 // selects an output for a given input
111 //
112 // input - pointer to value to change (and switch on), and float to multiply by
113 // output - modifies the pointer value
114 // return - original value input
115 //////////////////////////////////////////////////
116@ int casefn(int* intval_ptr, float* floatval_ptr){
117     int inputval;
118     int tmpval;
119     inputval = *intval_ptr;
120     tmpval = ifelsefn(*intval_ptr);
121     switch(tmpval){
122     case 5:
123         *intval_ptr = 6;
124         *floatval_ptr *= 6;
125         break;
126     case 9:
127         *intval_ptr = 10;
128         *floatval_ptr *=10;
129         break;
130     default:
131         *intval_ptr = 0;
132         *floatval_ptr = 0;
133         break;
134     } // end switch
135
136     return inputval;
137 } // end casefn
138
```

Debug

- Eclipse Debugger

- **rt-click** on your project → **Debug As** → **Local C/C++ Application**

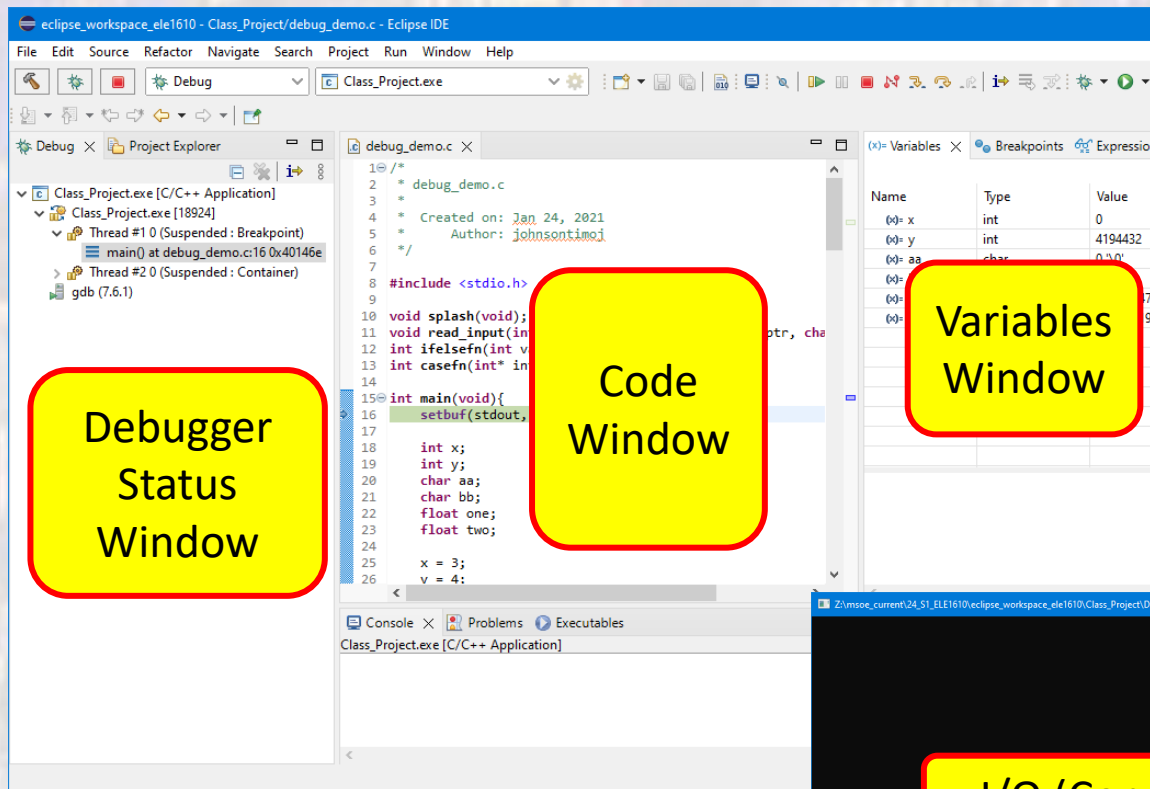
- **Select Switch** to change the perspective



Debug

- Eclipse Debugger

- The Eclipse perspective will be changed



- A console window will be opened

Debug

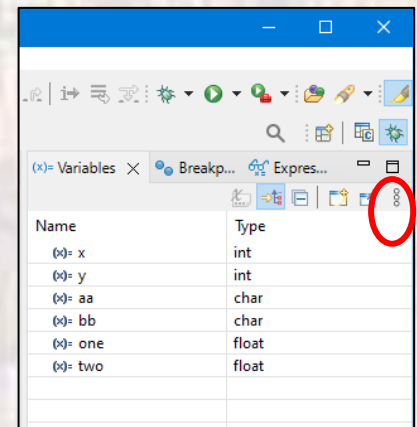
The screenshot shows the Eclipse IDE with a C++ application named 'Class_Project.exe' running in debug mode. The source code for 'debug_demo.c' is displayed in the center. The program is currently halted at line 16, which is the first executable line in the 'main' function. The toolbar at the top contains several buttons for debugging, with a red circle highlighting the 'Next' button (a right-pointing arrow). A yellow callout box with a red border points to this button with the text 'Arrow points to the NEXT line to be executed'. Another yellow callout box with a red border points to the 'Play/Pause/Stop' buttons (a play icon, a pause icon, and a stop icon) with the text 'Play Pause Stop'. A third yellow callout box with a red border points to line 16 of the code with the text 'Program is halted at first executable line in main'. The right-hand side of the IDE shows the 'Variables' view, which lists several variables: 'x' (int, 0), 'y' (int, 4194432), 'aa' (char, 0 '\0'), 'bb' (char, 0 '\0'), 'one' (float, 5.88737473e-), and 'two' (float, 8.99963199e-).

```
1 /*
2  * debug_demo.c
3  *
4  * Created on:
5  * Author:
6  */
7
8 #include <stdio.h>
9
10 void splash(void)
11 void read_input(int* floatval_ptr, char* floatval_ptr)
12 int ifelsefn(int)
13 int casefn(int* intval_ptr, float* floatval_ptr);
14
15 int main(void){
16     setbuf(stdout, NULL);
17
18     int x;
19     int y;
20     char aa;
21     char bb;
22     float one;
23     float two;
24
25     x = 3;
26     v = 4;
```

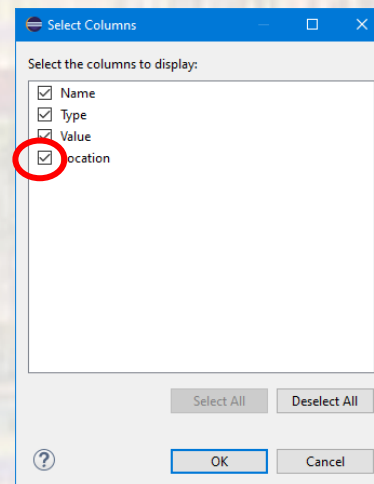
Name	Type	Value
(x)= x	int	0
(x)= y	int	4194432
(x)= aa	char	0 '\0'
(x)= bb	char	0 '\0'
(x)= one	float	5.88737473e-
(x)= two	float	8.99963199e-

Debug

- Eclipse Debugger – default data change
 - The debugger defaults to not-showing the memory location of variables
 - To modify this to show the memory location of variables
 - Click the 3-vertical-dots in the variable window



- Select Layout → Select Columns
- Check the Location box



Debug

Variables:
Only those in the current scope

Values
currently garbage

Memory
Location

Name	Type	Value	Location
(x)= x	int	0	0x61ff14
(x)= y	int	4194432	0x61ff1c
(x)= aa	char	0 '\0'	0x61ff13
(x)= bb	char	0 '\0'	0x61ff1b
(x)= one	float	5.88737473e-039	0x61ff0c
(x)= two	float	8.99963199e-039	0x61ff08

Debug

The screenshot shows the Eclipse IDE interface with a C program being debugged. The toolbar at the top contains various icons, with the 'Step Into' icon (a magnifying glass over a code line) circled in red. A yellow callout box with a red border provides the following information:

- Step Into:** Execute the current instruction
Go into a function
- Step Over:** Execute an entire function call
- Step Return:** Complete the current element and return
Complete a function
Finish a For/Switch/While statement

The background shows the Eclipse IDE with the following details:

- Window title: eclipse_workspace_ele1610 - Class_Project/debug_demo.c - Eclipse IDE
- Menu bar: File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, Help
- Toolbar: Includes icons for Run, Debug, Step Into, Step Over, Step Return, and other debugging actions.
- Editor: Shows the source code for debug_demo.c with the following content:

```
7
8 #include <stdio.h>
9
10 void splash(void);
11 void read_input(int* intval_ptr, float* floatval_ptr, char* charval_ptr);
12 int ifelsefn(int val);
13 int casefn(int* intval_ptr, float* floatval_ptr);
14
15 int main(void){
16
17     int x;
18     int y;
19     char aa;
20     char bb;
21     float one;
22     float two;
23
24     x = 3;
25     y = 4;
26     aa = 'a';
27     bb = 'b';
28     one = 1.0;
29     two = 2.0;
30
31     printf("x: %d, y: %d, aa: %c, bb: %c, one: %f, two: %f\n", x, y, aa, bb, one, two);
32
33     // splash()
34     splash();
```
- Variables window: Shows a table of variables with their names, types, and values.

Name	Type	Value
(x)= x	int	0
(x)= y	int	4194432
(x)= aa	char	0 '\0'
(x)= bb	char	0 '\0'
(x)= one	float	5.88732989e-039
(x)= two	float	8.99963199e-039

Debug

The screenshot shows the Eclipse IDE with a C program being debugged. The program is paused at line 25. The code is as follows:

```
12 int ifelsefn(int val);
13 int casefn(int* intval_ptr, float* floatval_ptr);
14
15 int main(void){
16     setbuf(stdo
17
18     int x;
19     int y;
20     char aa;
21     char bb;
22     float one;
23     float two;
24
25     x = 3;
26     y = 4;
27     aa = 'f';
28     bb = 'g';
29     one = 1.003;
30     two = 2.222;
31
32     printf("%i %i\n", x, y);
33
```

The IDE shows the following state:

- Debugger:** Thread #1 0 (Suspended : Step) is paused at line 25.
- Variables:** A table showing the current state of variables.
- Console:** The console is empty.

Name	Type	Value	Location
(x)= x	int	0	0x61ff14
(x)= y	int	4194432	0x61ff1c
(x)= aa	char	0 '\0'	0x61ff13
(x)= bb	char	0 '\0'	0x61ff1b
	float	5.88737473e-039	0x61ff0c
	float	8.99963199e-039	0x61ff08

Annotations in the image:

- step over – so we don't go into setbuf function:** Points to the 'step over' button in the debugger toolbar.
- Executes the setbuf function:** Points to the 'setbuf' function call in the code.
- No change → white:** Points to the console area, which is white, indicating no output.
- jumps to next executable command:** Points to the 'x = 3;' line in the code.

Debug

The screenshot shows the Eclipse IDE with a C++ project named 'Class_Project'. The main window displays the source code for 'debug_demo.c'. The code is as follows:

```
10  
11  
12 int ifelsefn(int val);  
13 int casefn(int* intval_ptr, float* floatval_ptr);  
14  
15 int main(void){  
16     setbuf(stdout, NULL);  
17  
18     int x;  
19     int y;  
20     char aa;  
21     char bb;  
22     float one;  
23     float two;  
24  
25     x = 3;  
26     y = 4;  
27  
28     aa = 'f';  
29     bb = 'g';  
30     one = 1.003;  
31     two = 2.222;  
32  
33     printf("%i %i\n", x, y);  
34  
35     // splash screen
```

The 'Variables' view on the right shows the current state of the program:

Name	Type	Value	Location
(x)= x	int	3	0x61ff14
(y)= y	int	4194432	0x61ff1c
(a)= aa	char	0'0'	0x61ff13
(b)= bb	char	0'0'	0x61ff1b
(o)= one	float	5.88737473e-039	0x61ff0c
(t)= two	float	8.99963199e-039	0x61ff08

Annotations in the image explain the following actions and states:

- Step into or step over:** A green box with an arrow pointing to the 'Step into' button in the toolbar.
- Executes the x=3 statement:** A yellow box with an arrow pointing to line 25 of the code.
- Value change highlighted in yellow:** A yellow box with an arrow pointing to the value '3' in the Variables view.
- jumps to next executable command:** A yellow box with an arrow pointing to line 27 of the code.

Debug

The screenshot shows the Eclipse IDE with a C program being debugged. The code in the editor is as follows:

```
15  
16  
17  
18  
19 char aa;  
20 char bb;  
21 float one;  
22 float two;  
23  
24 x = 3;  
25 y = 4;  
26 aa = 'f';  
27 bb = 'g';  
28 one = 1.00300002;  
29 two = 2.222;  
30  
31  
32 printf("%i %i\n", x, y);  
33  
34 // splash screen  
35 splash();  
36  
37 // input values  
38 read_input(&x, &one, &aa);  
39  
40 // ifelse function
```

The Variables window shows the following data:

Name	Type	Value	Location
(x)= x	int	3	0x61ff14
(x)= y	int	4	0x61ff1c
(x)= aa	char	102 'f'	0x61ff13
(x)= bb	char	103 'g'	0x61ff1b
one	float	1.00300002	0x61ff0c
two	float	2.22199988	0x61ff08

Annotations in the image provide the following information:

- Step into or step over 5 times**: Points to the first five lines of code (lines 19-23).
- Executes the y, aa, bb, one, and two statements**: Points to the assignment statements from line 24 to 29.
- Note: Chars show value and ascii**: Points to the character values 'f' and 'g' in the Variables window.
- Note: 2.222 cannot be represented exactly in floating point**: Points to the value 2.22199988 for the variable 'two' in the Variables window.
- jumps to next executable command**: Points to the printf statement on line 32.

Debug

The screenshot shows the Eclipse IDE with a C program being debugged. The code in the editor is as follows:

```
18
19
20
21
22
23
24
25 x = 3;
26 y = 4;
27 aa = 'f';
28 bb = 'g';
29 one = 1.003;
30 two = 2.222;
31
32 printf("%i %i\n", x, y);
33
34 // splash screen
35 splash();
36
37 // input values
38 read_input(&x, &one, &aa);
39
40 // ifelse function
41 y = ifelsefn(x);
42
43 // case function
```

The IDE interface includes a Project Explorer on the left, a Debug console at the bottom, and a Variables window on the right. The Variables window shows the following data:

Name	Type	Value	Location
(x)= x	int	3	0x61ff14
(x)= y	int	4	0x61ff1c
(x)= aa	char	102 'f'	0x61ff13
(x)= bb	char	103 'g'	0x61ff1b
		1.00300002	0x61ff0c
		2.22199988	0x61ff08

Annotations in the image provide the following information:

- Step Over – we do not want to go into the printf()**: A green callout box with an arrow pointing to the `printf` statement on line 32.
- Executes the printf() statement**: A yellow callout box with an arrow pointing to the `printf` statement on line 32.
- jumps to next executable command**: A yellow callout box with an arrow pointing to the `splash();` statement on line 35.
- Console window shows printf() results**: A yellow callout box with an arrow pointing to the console window at the bottom, which displays the output `3 4`.

Debug

The screenshot shows the Eclipse IDE with a C++ project named 'Class_Project'. The code in the editor is as follows:

```
21  
22  
23  
24  
25  
26  
27 aa = 1;  
28 bb = 'g';  
29 one = 1.003;  
30 two = 2.222;  
31  
32 printf("%i %i\n", x  
33  
34 // splash screen  
35 splash();  
36  
37 // input values  
38 read_input(&x, &one, &aa);  
39  
40 // ifelse function  
41 y = ifelsefn(x);  
42  
43 // case function  
44 y = casefn(&x, &two);  
45  
46 // port manipulation
```

The 'Variables' window shows the following data:

Name	Type	Value	Location
(x)= x	int	3	0x61ff14
(x)= y	int	4	0x61ff1c
(x)= aa	char	102 'f'	0x61ff13
(x)= bb	char	103 'g'	0x61ff1b
		1.00300002	0x61ff0c
		2.22199988	0x61ff08

The console window shows the output of the program:

```
3 4  
Program to demonstrate the debugger
```

Annotations in the image explain the following actions:

- Step Over – we do not want to go into the splash()**: Points to the 'Step Over' button in the IDE toolbar.
- Executes the splash() statement**: Points to the `splash();` line in the code.
- jumps to next executable command**: Points to the `read_input(&x, &one, &aa);` line in the code.
- Console window shows splash() results**: Points to the console output showing the numbers '3 4'.

Debug

Step Into – we are now in the read_input function

The screenshot shows the Eclipse IDE with a C++ project. The main editor displays the source code of 'debug_demo.c', with the 'read_input' function highlighted. The left sidebar shows the Project Explorer and Debug Console. The right sidebar shows the Variables view with a table of current variables.

Name	Type	Value	Location
intval_ptr	int *	0x61ff14	0x61fee0
*intval_p	int	3	0x61ff14
floatval_ptr	float *	0x61ff0c	0x61fee4
charval_ptr	char *	0x61ff13 "f\003"	0x61fee8

Entered read_input and created storage for the formal parameters

Note: only in-scope variables are visible

Note: pointers are indicated with an arrow and show the pointer value (memory location pointed to)

Expanding a pointer shows the value it points to

Debug

The screenshot shows the Eclipse IDE with a C++ application named 'Class_Project.exe' running in debug mode. The code editor displays the following code:

```
71 //  
72 // read in an int, a float, and a char  
73 //  
74 // inputs - none  
75 // output - int/float/char via pointers  
76 // return - void  
77 ///////////////////////////////////////////////////////////////////  
78 void read_input(int* intval_ptr, float* floatval_ptr, cha  
79     printf("Please enter an int, a float, and a character  
80     scanf("%i %f %c", intval_ptr, floatval_ptr, charval_p  
81  
82     return;
```

The IDE interface includes a Project Explorer on the left, a toolbar at the top, and a Variables/Expressions window on the right. The Variables window is currently empty.

Annotations in the image provide the following information:

- Step over the printf()**: A green box with a red border pointing to the `printf` statement on line 79.
- Step over the scanf()**: A green box with a red border pointing to the `scanf` statement on line 80.
- On scanf() the debugger stops and waits for user input (via the separate console window)**: A yellow box with a red border.
- output suspended**: A yellow box with a red border, with an arrow pointing to the console window.

The console window at the bottom shows the following output:

```
Z:\msoe_current\24_S1_ELE1610\eclipse_workspace_ele1610\Class_Project\Debug\Class_Project.exe  
3 4  
  
Program to demonstrate the debugger  
Please enter an int, a float, and a character:
```

Debug

type in values and hit return

```
73 //
74 // inputs - none
75 // output - int/float/char via pointers
76 // return - void
77 ///////////////////////////////////////////////////////////////////
78 void read_input(int* intval_ptr, float* floatval_ptr, cha
79     printf("Please enter an int, a float, and a character
80     scanf("%i %f %c", intval_ptr, floatval_ptr, charval_p
81
82     return;
83 } // end read_input
84
85 ///////////////////////////////////////////////////////////////////
86 // ifelsefn
87 //
88 // selects an output for a given input
89 //
90 // inputs - int to select on
91 // output - none
92 // return - random value based on input
93 ///////////////////////////////////////////////////////////////////
94 int ifelsefn(int val){
95     int result;
96     if(val == 0)
```

Name	Type	Value	Location
intval_ptr	int *	0x61ff14	0x61fee0
*intval_p	int	7	0x61ff14
floatval_ptr	float *	0x61ff0c	0x61fee4
charval_ptr	char *	0x61ff13 "T,a"	0x61fee8

values updated

jumps to next executable command

Debug

Step into or step over

The screenshot shows the Eclipse IDE with a C++ project named 'Class_Project'. The main window displays the source code for 'debug_demo.c'. The function 'read_input' is highlighted in blue, and the cursor is positioned at its closing brace on line 83. A yellow callout box with a red border points to this brace, containing the text 'jumps to the closing brace on the function'. The 'Variables' window on the right shows the following data:

Name	Type	Value	Location
intval_ptr	int *	0x61ff14	0x61fee0
*intval_ptr	int	7	0x61ff14
floatval_ptr	float *	0x61ff0c	0x61fee4
charval_ptr	char *	0x61ff13 "T\a"	0x61fee8

The console at the bottom shows the output of the program: 'Please enter an int, a float, and a character' followed by '7' and 'T\a'.

jumps to the closing brace on the function

Debug

The screenshot shows the Eclipse IDE with a C++ application named 'Class_Project.exe' being debugged. The code is paused at line 41, which is `y = ifelsefn(x);`. The 'Thread #1 0 (Suspended : Step)' is visible in the Debug console. The 'Variables' window on the right shows the current state of variables: `x` is 7, `y` is 4, `aa` is '84 T', `bb` is '103 g', `one` is 3.14159012, and `two` is 2.22199988. Annotations explain the 'Step into' and 'Step over' actions, the return to main, and how variable values are updated in the main scope.

Step into or step over

Returns to main

Values that were updated via pointer show updates in the main() scope

jumps to next executable command

Name	Type	Value	Location
(x)= x	int	7	0x61ff14
(x)= y	int	4	0x61ff1c
(x)= aa	char	84 'T'	0x61ff13
(x)= bb	char	103 'g'	0x61ff1b
(x)= one	float	3.14159012	0x61ff0c
(x)= two	float	2.22199988	0x61ff08

Additional Things we can do in the debugger

Debug

Instead of single stepping
Right click on a line and select run_to

Workspace_V10_EE1910 - Class_MSP_Project/debug_example.c - Code Composer Studio

File Edit View Project Tools Run Scripts Window Help

Debug

Class_MSP_Project [Code Composer Studio - Device Debugging]

Texas Instruments XDS110 USB Debug Probe/CORTEX_M4_0 (Suspended - HW Breakpoint)

main() at debug_example.c:33 0x00002F14

._c_int00_noargs() at boot_cortex_m.c:121 0x000044E8 (_c_int00_noargs does not contain fi

Name	Type	Value	Location
(*) a	int	2	0x2000FFE8
(*) b	int	4	0x2000FFEC
(*) c	float	2.5	0x2000FFF0
(*) d	unsigned char	115 's'	0x2000FFF4

```
17
18 int main(void){
19     setbuf(stdout, NULL); // added to force printing to flush during debug
20
21     int a;
22     int b;
23     float c;
24     char d;
25     a = 2;
26     c = 2.5;
27     d = 's';
28
29     b = 2 * a;
30
31     printf("%c\n",d);
32
33     d = d + 1;
34
35     printf("Enter a new character: ");
36     scanf("%c", &d);
37
38     c = doublef(c);
39
40     doublei(&b);
41 }
```

debug_example.c setbuf.c

Executes all the commands up-to but not including the line selected

Console

Class_MSP_Project:CIO
[CORTEX_M4_0] s

Updates Available

Updates are available for your software. Click to review and install updates. Set up [Reminder options](#)

281M of 851M

Debug

Instead of single stepping

Right click on the blue area next to a line and select toggle breakpoint

Then hit run

Right click on the blue area next to a line and select toggle breakpoint – to turn it off again

Executes all the commands up-to but not including the line with the breakpoint

Breakpoint bubble

The screenshot shows the Code Composer Studio interface. The top menu bar includes File, Edit, View, Project, Tools, Run, Scripts, Window, and Help. The Debug console shows the project 'Class_MSP_Project' and the device 'Texas Instruments XDS110 USB Debug Probe/CORTEX_M4_0 (Suspended - HW Breakpoint)'. The main window displays the source code for 'debug_example.c' with a breakpoint set on line 33. The code includes a main function with variables a, b, c, and d, and a printf statement. The console output shows 'Memory Map Initialization Complete', 'Halting watchdog Timer', and a 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.' An 'Updates Available' notification is visible in the bottom right corner.

Debug

```
/*
 * debug_example_bp.c
 *
 * Created on: Dec 17, 2020
 * Author: johnsontimoi
 */
////////////////////////////////////
//
// Program to demonstrate debugger
//
////////////////////////////////////
#include "msp432.h"
#include <stdio.h>

int main(void){
    setbuf(stdout, NULL); // added to force printing to flush during debug

    int a;
    char d;

    while(1){
        printf("\nEnter a new character: ");
        scanf("%c", &d);


        switch(d){
            case 'a':
                a = 5;
                break;
            case 'c':
                a = 7;
                break;
            case 'd':
                a = 9;
                break;
            default:
                a = 0;
                break;

        } // end switch
        printf("a is: %i\n", a);

    } // end while

    return 0;
} // end main
```

Would like to stop
when a c is entered



Debug

Instead of single stepping

Right click on the blue area next to a line and select toggle breakpoint (a = 7)

Then hit run

Right click on the blue area next to a line and select toggle breakpoint – to turn it off again

Executes until the breakpoint is encountered – then stops in debug mode

Breakpoint bubble

Runs through the loop for a and t
Encounters the breakpoint for c

Updates Available
Updates are available for your software. Click to review and install updates.
Set up [Reminder options](#)

The screenshot shows the Code Composer Studio interface. The main window displays a C program with a switch statement. A breakpoint is set on line 32, which is highlighted in blue. The console window at the bottom shows the program's output: "Enter a new character: a", "a is: 5", "Enter a new character: t", "a is: 0", and "Enter a new character: c". The program has stopped at the breakpoint on line 32. A yellow callout box points to the breakpoint bubble on line 32, and another yellow callout box points to the console output for the character 'c'.

```
24 scanf("%c", &d);
25
26 switch(d){
27     case 'a':
28     case 'b':
29         a = 5;
30         break;
31     case 'c':
32         a = 7;
33         break;
34     case 'd':
35         a = 9;
36         break;
37     default:
38         a = 0;
39         break;
40
41 } // end switch
42 printf("a is: %i\n", a);
43
44 } // end while
45
46 return 0;
47 } // end main
48
```

Class_MSP_Project:CIO
Enter a new character: a
a is: 5
Enter a new character: t
a is: 0
Enter a new character: c