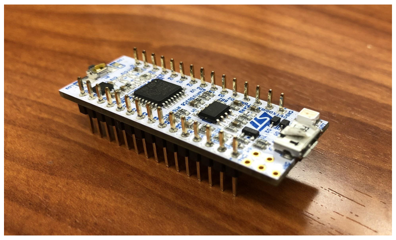
E85: Digital Electronics and Computer Engineering
Lab 6: Linear Algebra in C on a Microcontroller

Objective

The purpose of this lab is to familiarize yourself with programming and debugging in C on an embedded microcontroller. Specifically, you will write some linear algebra routines that will help you become comfortable with loops, arrays, and pointers.

1. Welcome to the STM32 Nucleo

The STM32 Nucleo board is a circuit board smaller than 2 x 5 cm at a cost of $11 that has a complete 32-bit ARM Cortex-M0 microcontroller with a wide assortment of peripherals inside. With this board, you could add a microprocessor to a system for comparable cost to a piece of lumber or pipe.

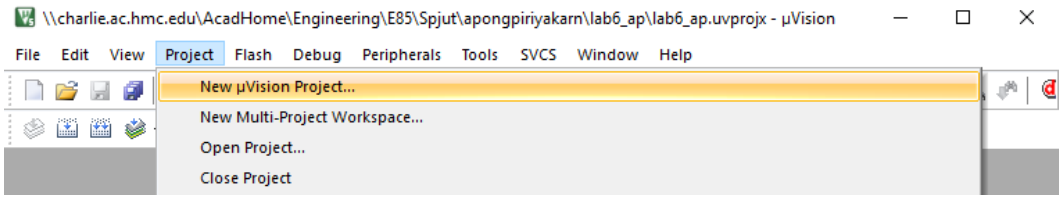


The Nucleo works with the industry-standard Keil Microcontroller Development Kit (MDK). You can use the tools in the E85 lab, or can purchase your own STM32F042 and download the free unlicensed version of the MDK-ARM software to use on your own Windows, Linux, or Mac computer (microUSB cable required). If you install yourself, make sure you install the STM32F0 device pack during the installation process. The MDK-ARM software used during Fall 2018 is version 5.25 which can be found here. The MDK5 Device Family Pack v2.0 can be found here.

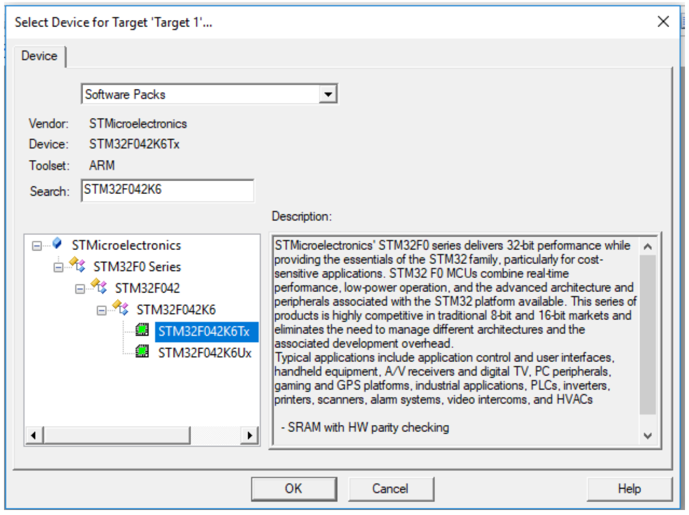
2. Keil Tutorial

In this section, you will learn to write, compile, and debug programs with the Keil MDK.

- Launch **Keil uVision** from the start menu.
- Choose **Project -> New uVision Project** to create a new project named 'tutorial'. Save it in a new folder in your Charlie directory.


\\charlie.ac.hmc.edu\AcadHome\Engineering\E85\Spjut\apongpiriyakarn\lab6_ap\lab6_ap.uvprojx - µVision
File
Edit
View
Project
Flash
Debug
Peripherals
Tools
SVCS
Window
Help
New µVision Project...
New Multi-Project Workspace...
Open Project...
Close Project

- Select the ‘STM32F042K6Tx’ device target using the search box or under **STMicroelectronics -> STM32F0 Series -> STM32F042 -> STM32F042K6 -> STM32F042K6Tx**.



Software Packs

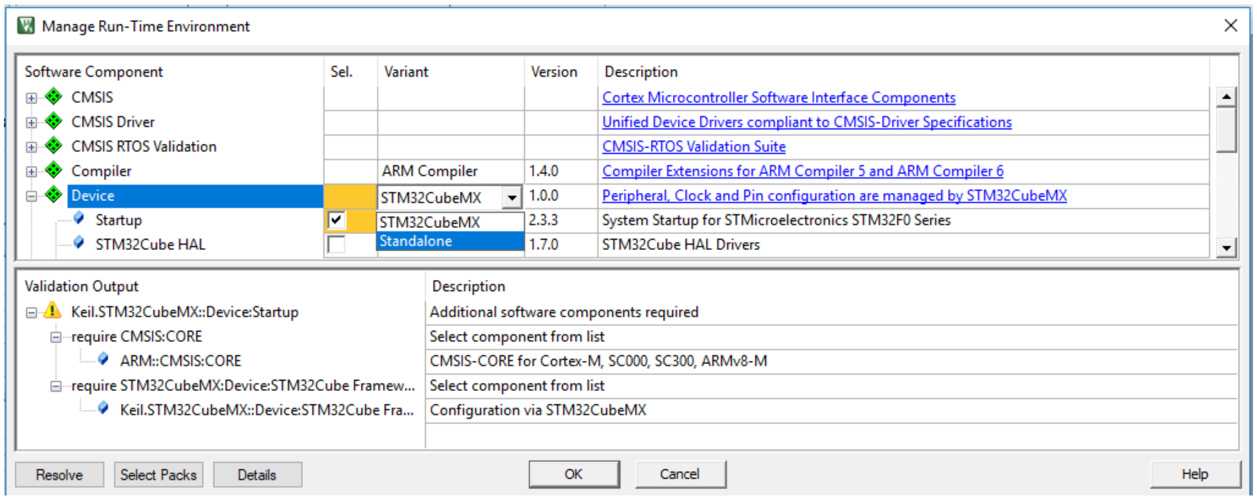
Vendor: STMicroelectronics
Device: STM32F042K6Tx
Toolset: ARM
Search: STM32F042K6

STMicroelectronics
STM32F0 Series
STM32F042
STM32F042K6
STM32F042K6Tx
STM32F042K6Ux

Description:
STMicroelectronics' STM32F0 series delivers 32-bit performance while providing the essentials of the STM32 family, particularly for cost-sensitive applications. STM32 F0 MCUs combine real-time performance, low-power operation, and the advanced architecture and peripherals associated with the STM32 platform available. This series of products is highly competitive in traditional 8-bit and 16-bit markets and eliminates the need to manage different architectures and the associated development overhead.
Typical applications include application control and user interfaces, handheld equipment, A/V receivers and digital TV, PC peripherals, gaming and GPS platforms, industrial applications, PLCs, inverters, printers, scanners, alarm systems, video intercoms, and HVACs

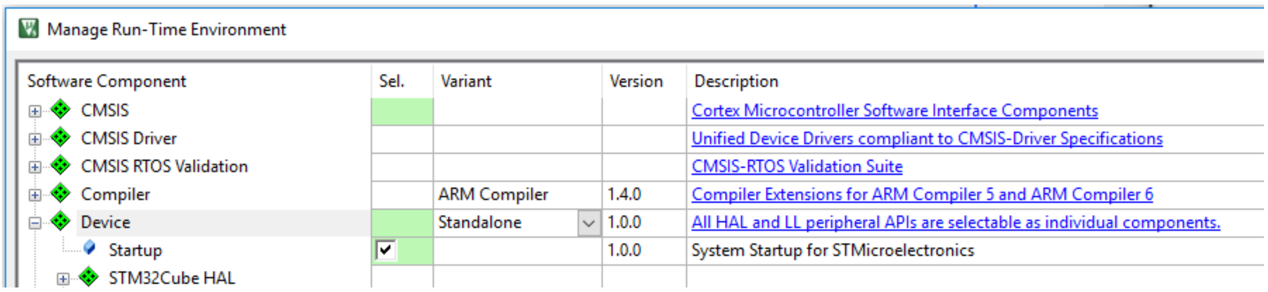
- SRAM with HW parity checking

- In the **Manage Run-Time Environment** window, expand **Device.** In the Device Variant column, choose **Standalone**. Then check the **Startup** box. Click on the **Resolve** button at the bottom to resolve dependencies and you’ll see that **CMSIS** (the Cortex Microcontroller Software Interface System) is selected too. Choose **OK**.



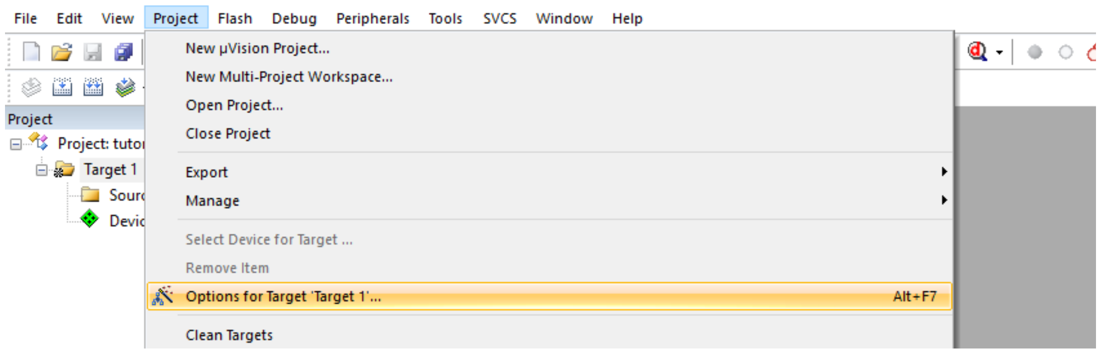
Software Component	Sel.	Variant	Version	Description
CMSIS				Cortex Microcontroller Software Interface Components
CMSIS Driver				Unified Device Drivers compliant to CMSIS-Driver Specifications
CMSIS RTOS Validation				CMSIS-RTOS Validation Suite
Compiler		ARM Compiler	1.4.0	Compiler Extensions for ARM Compiler 5 and ARM Compiler 6
Device		STM32CubeMX	1.0.0	Peripheral, Clock and Pin configuration are managed by STM32CubeMX
Startup	✓	STM32CubeMX	2.3.3	System Startup for STMicroelectronics STM32F0 Series
STM32Cube HAL		Standalone	1.7.0	STM32Cube HAL Drivers

Validation Output	Description
Keil.STM32CubeMX::Device:Startup	Additional software components required
require CMSIS:CORE	Select component from list
ARM::CMSIS:CORE	CMSIS-CORE for Cortex-M, SC000, SC300, ARMv8-M
require STM32CubeMX:Device:STM32Cube Framew...	Select component from list
Keil.STM32CubeMX::Device:STM32Cube Fra...	Configuration via STM32CubeMX

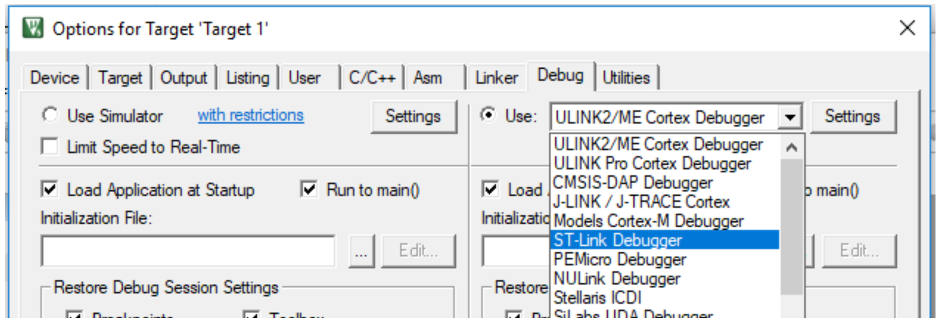


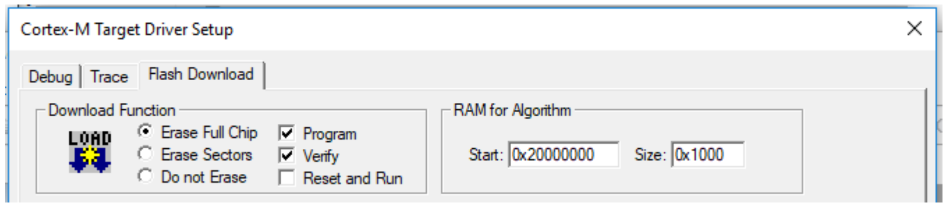
Software Component	Sel.	Variant	Version	Description
CMSIS				Cortex Microcontroller Software Interface Components
CMSIS Driver				Unified Device Drivers compliant to CMSIS-Driver Specifications
CMSIS RTOS Validation				CMSIS-RTOS Validation Suite
Compiler		ARM Compiler	1.4.0	Compiler Extensions for ARM Compiler 5 and ARM Compiler 6
Device		Standalone	1.0.0	All HAL and LL peripheral APIs are selectable as individual components.
Startup	✓		1.0.0	System Startup for STMicroelectronics
STM32Cube HAL				

- In the project pane, click to expand **Target 1**. You will see **Source Group 1**. Click on the **Target1** item, then choose **Project -> Options for Group ‘Target 1…’**.



- In the **Debug** tab, choose **Use: ST-Link Debugger** (top right area), then click on **Settings**, go to the **Flash Download** tab and choose **Erase Full Chip**. Click **OK** then **OK** to close the dialog boxes.





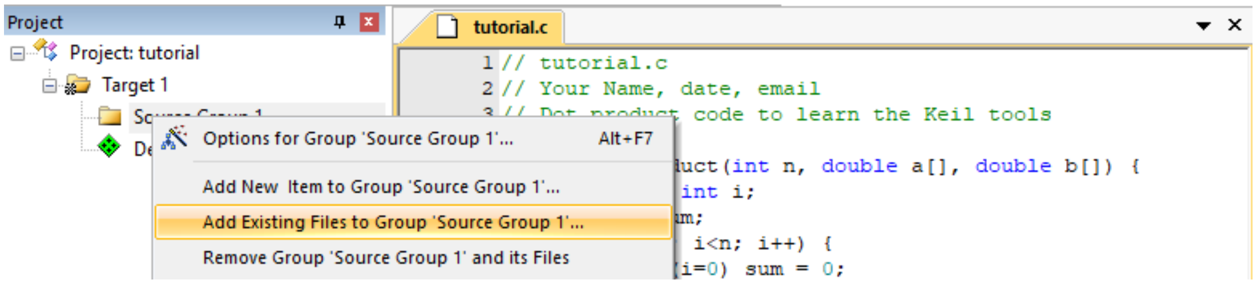
Now, we will create a program that computes the dot product of two vectors.

- Choose **File -> New...**. Enter the following code below. Note that it intentionally contains some bugs. Save your program as 'tutorial.c'.

```
// tutorial.c
// Your Name, date, email
// Dot product code to learn the Keil tools
#define DIM 3
double dotproduct(int n, double a[], double b[]) {
	volatile int i;
	double sum;
	for (i=0; i<n; i++) {
		if (i=0) sum=0;
		sum += a[i]*a[i];
	}
	return sum;
}
int main(void) {
	double x[DIM] = {3, 4, 5}; // x is an array of size 3(DIM)
```

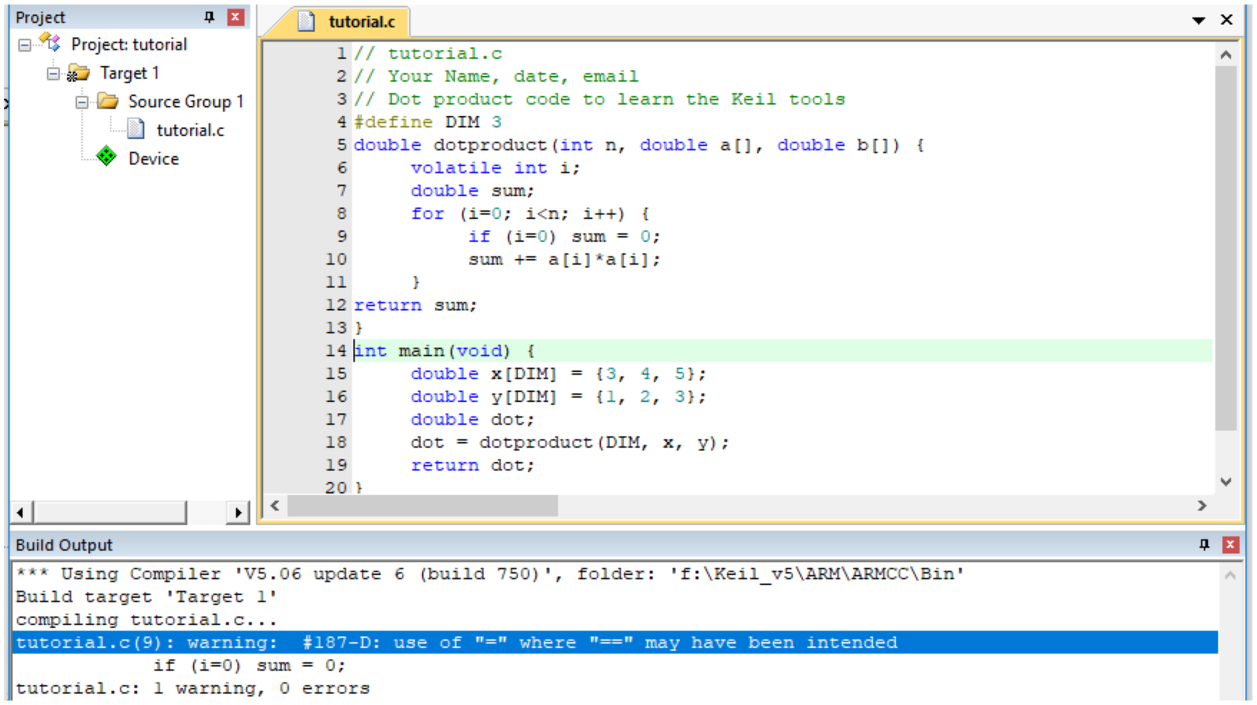
```
        double y[DIM] = {1, 2, 3}; // same as y
        double dot;
        dot = dotproduct(DIM, x, y);
        return dot;
}
```

- Calculate the dot product of [3 4 5] and [1 2 3] to predict the output of your program.
- Right-click on the **Source Group 1** and "**Add Existing Files to Group 'Source Group 1'...**" and add tutorial.c. You should now see **tutorial.c** when expand **Source Group 1**.


Project
Project: tutorial
Target 1
Source Group 1
De
tutorial.c
Options for Group 'Source Group 1'... Alt+F7
Add New Item to Group 'Source Group 1'...
Add Existing Files to Group 'Source Group 1'...
Remove Group 'Source Group 1' and its Files

```
1 // tutorial.c
2 // Your Name, date, email
3 // Dot product code to learn the Keil tools
uct(int n, double a[], double b[]) {
int i;
um;
 i<n; i++) {
(i=0) sum = 0;
```

- Choose **Project -> Build Target** to compile (or click on the top panel; or press F7). Scroll through the **Build Output** panel at the bottom to look for warnings and errors. You will see a warning that the `(i=0)` comparison should have been `(i==0)`. Change your code to correct this and rebuild and ensure there are no errors or warnings.

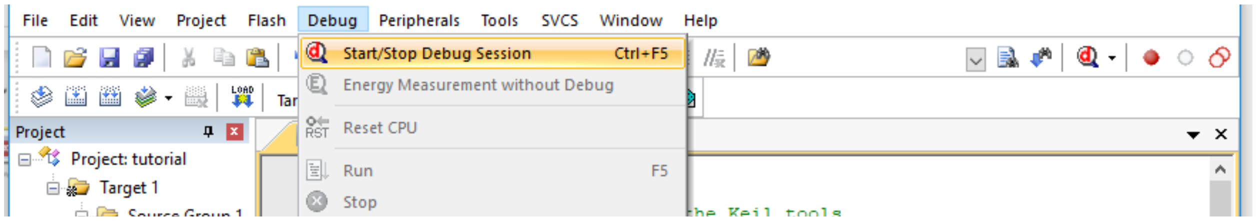


```
 1 // tutorial.c
 2 // Your Name, date, email
 3 // Dot product code to learn the Keil tools
 4 #define DIM 3
 5 double dotproduct(int n, double a[], double b[]) {
 6       volatile int i;
 7       double sum;
 8       for (i=0; i<n; i++) {
 9             if (i=0) sum = 0;
10             sum += a[i]*a[i];
11       }
12 return sum;
13 }
14 int main(void) {
15       double x[DIM] = {3, 4, 5};
16       double y[DIM] = {1, 2, 3};
17       double dot;
18       dot = dotproduct(DIM, x, y);
19       return dot;
20 }
```

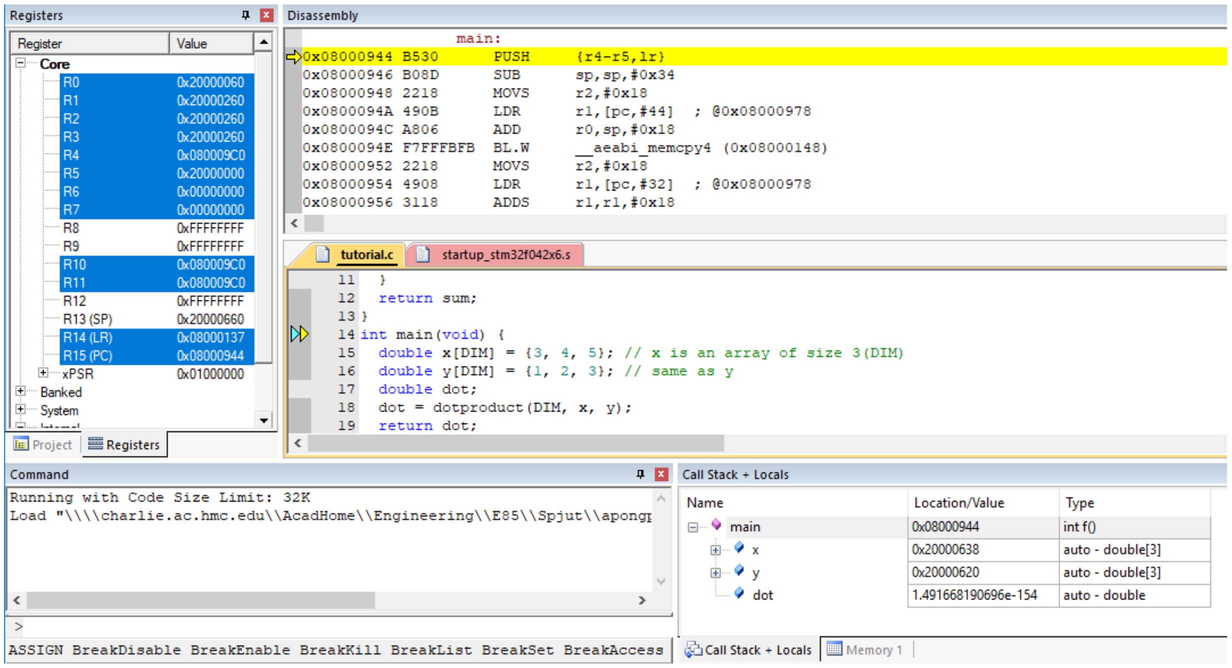
```
*** Using Compiler 'V5.06 update 6 (build 750)', folder: 'f:\Keil_v5\ARM\ARMCC\Bin'
Build target 'Target 1'
compiling tutorial.c...
tutorial.c(9): warning:  #187-D: use of "=" where "==" may have been intended
             if (i=0) sum = 0;
tutorial.c: 1 warning, 0 errors
```

- Make sure a Nucleo board is plugged and connected via a USB port.

- Choose **Debug -> Start/Stop Debug Session** to invoke the debugger (or press Ctrl then F5). You should see a yellow arrow pointing to the beginning of the main function.



- Use the **Step** command (on the icon bar, or press F11) to step through your code. You can watch the variable addresses and values in the **Call Stack + Locals** pane in the lower right. Make it bigger so you can see everything. Expand the x and y arrays so you can see their values. As you step through the first two lines, you should see arrays getting initialized. The compiler aggressively optimizes the code, so sometimes you will see weird things in the debugger or variables not changing when you expect them to change. i is declared as volatile in this code to force it to be preserved by the optimizer and appear in the debugger.



Register	Value
Core	
R0	0x20000060
R1	0x20000260
R2	0x20000260
R3	0x20000260
R4	0x080009C0
R5	0x20000000
R6	0x00000000
R7	0x00000000
R8	0xFFFFFFFF
R9	0xFFFFFFFF
R10	0x080009C0
R11	0x080009C0
R12	0xFFFFFFFF
R13 (SP)	0x20000660
R14 (LR)	0x08000137
R15 (PC)	0x08000944
xPSR	0x01000000
Banked	
System	

```
                    main:
0x08000944 B530       PUSH      {r4-r5,lr}
0x08000946 B08D       SUB       sp,sp,#0x34
0x08000948 2218       MOVS      r2,#0x18
0x0800094A 490B       LDR       r1,[pc,#44]  ; @0x08000978
0x0800094C A806       ADD       r0,sp,#0x18
0x0800094E F7FFFBFB   BL.W      __aeabi_memcpy4 (0x08000148)
0x08000952 2218       MOVS      r2,#0x18
0x08000954 4908       LDR       r1,[pc,#32]  ; @0x08000978
0x08000956 3118       ADDS      r1,r1,#0x18
```

```
  }
  return sum;
}
int main(void) {
  double x[DIM] = {3, 4, 5}; // x is an array of size 3(DIM)
  double y[DIM] = {1, 2, 3}; // same as y
  double dot;
  dot = dotproduct(DIM, x, y);
  return dot;
```

```
Running with Code Size Limit: 32K
Load "\\\\charlie.ac.hmc.edu\\AcadHome\\Engineering\\E85\\Spjut\\apongp
>
ASSIGN BreakDisable BreakEnable BreakKill BreakList BreakSet BreakAccess
```

Name	Location/Value	Type
main	0x08000944	int f()
x	0x20000638	auto - double[3]
y	0x20000620	auto - double[3]
dot	1.491668190696e-154	auto - double

- Find the bug that causes the dot product to be incorrect. Fix your code. Use the debugger (**Debug -> Start/Stop Debug session**), then rebuild the code, download it again.

3. Linear Algebra

The goal in this section is to write a library of linear algebra routines in C. This will help you get accustomed to loops, arrays, and pointers in C, and it is good to understand these routines because they are fundamental building blocks of signal processing code.

Mathematical operations that you will be writing include matrix addition, linear combination of matrices, matrix transpose, matrix equality, and matrix multiplication.

The following functions operate on matricies of m rows and n columns (m x n). You may assume that the result matrix have already been allocated prior to the function call. The transpose function produces an n x m result. Function declarations are given below:

```
void add(int m, int n, double *A, double *B, double *Y); //Y=A+B

void linearcomb(int m, int n, double sa, double sb, double *A, double *B, double *Y);
//Y=sa*A + sb*B

void transpose(int m, int n, double *A, double *A_t);
      //A_t=transpose(A)

int equal(int m, int n, double *A, double *B); //returns 1 if equal, 0 if not
```

The last function multiplies an m1 x n1m2 matrix A by an n1m2 x n2 matrix B to produce an m1 x n2 matrix Y. Y should already be allocated and the contents will be overwritten. The following is the function declaration.

```
void mult(int m1, int n1m2, int n2, double *A, double *B, double *Y); //Y=A*B
```

Now is your turn to program.

- Write a C program to complete five operations given above
- Test your program with the following code, using the `newMatrix` and `newIdentityMatrix` code from lecture. Remember that you will need to include the standard library that has the `malloc` function using the statement: `#include <stdlib.h>` at the beginning of your code.

```
#include <stdlib.h> // for malloc

double* newMatrix(int m, int n) {
      double *mat;
      mat = (double*)malloc(m*n*sizeof(double));
      return mat;
}

double* newIdentityMatrix(int n) {
      double *mat = newMatrix(n, n);
      int i, j;
      for (i=0; i<n; i++)
            for (j=0; j<n; j++) {
                  mat[j+i*n] = (i==j);
      return mat;
}
int main(void) {
  double v1[3] = {4, 2, 1};                         // 1x3 vector
  double v2[3] = {1, -2, 3};                        // 1x3 vector
  double dp = dotproduct(3, v1, v2);                      // compute v1 dot v2
  double m1[9] = {0, 0, 2, 0, 0, 0, 2, 0, 0};      // 3x3 matrix
  double *m2 = newIdentityMatrix(3);                      // 3x3 identity matrix
  double *m3 = newMatrix(3, 3);                     // 3x3 matrix
  double m4[6] = {2, 3, 4, 5, 6, 7};                      // 3x2 matrix
  double *m5 = newMatrix(3, 2);                     // 3x2 matrix
  double m6[6] = {6, 2, 5, 8, 2, 7};                      // 2x3 matrix
  double *m7 = newMatrix(3, 2);                     // 3x2 matrix
  double *m8 = newMatrix(3, 2);                     // 3x2 matrix
  double expected[6] = {2, 1, 0, 1, 0, -1};        // expected result matrix
  int eq;
  add(3, 3, m1, m2, m3);                            // m3= m1+m2
  mult(3, 3, 2, m3, m4, m5);                  // m5= m3*m4 (3x2 result matrix)
  transpose(2, 3, m6, m7);                    // m7= m6^t
  linearcomb(3, 2, 1, 1-dp, m5, m7, m8);      // m8= 1*m5 + (1-dp)*m7
  eq = equal(3, 2, m8, expected);             // check if m8 is as expected
  return eq;                                  // return 1 if so; 0 otherwise
}
```

- Predict what each of the matrices should be and particularly check that m8 matches your expectations.
- You'll find it frustrating in the debugger watch window that when you look at a pointer variable such as m2, you only see the 0th element. However, the watch window (open from View menu) allows you to enter expressions. You can type in particular elements such as m2[4] to have them displayed for you.

4. Extra Credit: Solving Systems of Linear Equations

Write a C function to invert an n x n matrix. If the matrix is singular, the function should return 0 and Y is a don't care; otherwise, the function should return 1 and Y is A^{-1}. Use the following function declaration:

```
      int invert(int n, double *A, double *Y);
```

Then write a function to solve for x in Ax=b for a system of n variables.

```
      int solve(int n, double *A, double *b, double *x);
```

The function should again return 0 if A is singular.

Use your function to solve the following system of linear equations:

```
3a  +  b +  c + d = 6
2a  + 3b + 4c – d = 12
-4a           + d = 8
      3b – 2c     = 0
```

Note that your new program may hang from lack of memory. Comment out the test code from the previous part or use free to free up some space. Be careful in your memory allocation and dellocation, particularly if you are using a recursive determinant function. A malloc that fails for lack of memory will return a NULL pointer (with an address of 0).

What to Turn In

1. Please indicate how many hours you spent on this lab. This will be helpful for calibrating the workload for next time the course is taught.
2. Code for `add`, `linearcomb`, `transpose`, `equal`, and `mult`.
3. What does your code produce for m8? Does it match your expectations?
4. Extra credit, if applicable. Give your code and a, b, c, and d.

Please indicate any bugs you found in this lab manual, or any suggestions you would have to improve the lab.