1. #include <stdio.h>
2. int main()
3. {
4.     int i, j, k;
5.     int myA[2][2] = {{1, 2}, {3, 4}};
6.     int myB[2][2] = {{5, 6}, {7, 8}};
7.     int myC[2][2] = {{0, 0}, {0, 0}};
8.     for (i=0; i<2; i++)
9.         for (j=0; j<2; j++)
10.            for (k=0; k <2; k++){
11.                 myC[i][j] = myC[i][j] + myA[i][k]*myB[k][j];
12.             }
13.         }
14.     }
15. }
14. for(i=0; i<2; i++){
15.     for(j=0; j<2; j++){
16.         printf(" %d ", myC[i][j]);
17.     }
18.     printf("\n");
19. }
20. return 0;
21.}
Overview of the Course

• Writing a Basic C Program
• Understanding Errors
• Comments Keywords, Identifiers, Variables
• Operators
• Standard Input and Output (Basic)
• Control Structures
• Standard Input and Output
• Arrays, Structures
• Functions in C
• Pointers
• Working with Files

All the concepts will be accompanied with examples.
#include <stdio.h>

void add(int a, int b){
    int c;
    c = a + b;
    printf("
Addition is : %d",c);
}

int main(){
    int a, b;
    printf("
Enter Any 2 Numbers : ");
    fflush(stdout);
    scanf("%d %d",&a,&b);
    add(a, b);
    return 0;
}

Note: The variables used as formal and actual parameters can have different names.
Passing Values to Functions: passValue2.c

```c
#include <stdio.h>
#include <stdlib.h>

void add(int a, int b){
    //same code as in the previous slide
}

int main(int argc, char *argv[]){
    int a, b;
    if ( argc != 3 ){
        printf("\nInsufficient num. of arguments.\n");
        printf( "\nUsage:%s <firstNum> <secondNum>" , argv[0]);
    }else{
        a = atoi(argv[1]);
        b = atoi(argv[2]);
        add(a, b);
    }

    return 0;
}
```
```c
int main(int argc, char *argv[]){
    int a, b;
    if ( argc != 3 ){
        printf("Insufficient num. of arguments.\n");
        printf("Usage:%s <firstNum> <secondNum>", argv[0]);
    } else{
        a = atoi(argv[1]);
        b = atoi(argv[2]);
        add(a, b);
    }
    return 0;
}
```


```c
#include <stdio.h>

int add(int a, int b) { <-- Notice the return type
    int c;
    c = a + b; a=c; b=c;
    printf("\n Addition is : %d",c);
    return c; <-- Return value: c
}

int main(){
    int a, b, c;
    printf("\n Enter Any 2 Numbers : ");
    scanf("%d %d",&a,&b);
    printf("a is: %d, b is: %d\n", a, b);
    c = add(a, b); <-- Value returned from add stored in c
    printf("a is: %d, b is: %d\n", a, b);
    return 0;
}
```
Note: the values of $a$ and $b$ remained the same when accessed from function main.
Pointers

• A pointer is a variable that stores an address in memory - address of another variable

• For instance, the value of a pointer may be 42435. This number is an address in the computer's memory which is the start of some data

• We can dereference the pointer to look at or change the data

• Like variables, you have to declare pointers before you use them

• The data type specified with pointer declaration is the data type of the variable the pointer will point to
Revisiting Variable Declaration

- Consider the declaration
  
  ```c
  int i = 3;
  ```

- This declaration tells the C compiler to:
  - Reserve space in memory to hold the integer value
  - Associate the name `i` with this memory location
  - Store the value `3` at this location

  ![Diagram showing variable declaration](chart.png)

  ```plaintext
  i ←----- Location name
  3 ←----- Value at location
  6485 ←----- Location number (Address)
  ```
‘Address of’ Operator

```c
#include <stdio.h>

int main(){
    int i=3;
    printf("\nAddress of i = \%p", &i);
    printf("\nValue of i = \%d", i);
    return 0;
}
```

Output:
Address of i = 0x22ff0c
Value of i = 3

Note:
&i Returns the address of variable i
`Value at Address’ Operator: printAddress.c

```c
#include <stdio.h>

int main()
{
    int i = 3;
    printf("\nAddress of i = %u", &i);
    printf("\nValue of i = %d", i);
    printf("\nValue of i = %d", *(&i));
    return 0;
}
```

Output:  
Address of i = 2293532  
Value of i = 3  
Value of i = 3

Note:  
`&i` returns the address of variable `i`  
`*(&i)` returns the value at address of `i`
Pointer Expressions

• In the previous example, the expression \&i returns the address of i.

• This address can be collected in a variable as 

  \( j = \&i; \)

• j is a variable which contains the address of another variable and is declared as int *j;

\[
\begin{array}{c}
i \leftarrow \text{Location name} \rightarrow j \\
3 \leftarrow \text{Value at location} \rightarrow 6485 \\
6485 \leftarrow \text{Location number} \rightarrow 3276 \quad \text{(Address)}
\end{array}
\]
Pointers:
pointerExample2.c

#include <stdio.h>

int main()
{
    int i = 3;
    int *j;
    j = &i;

    printf("\nAddress of i = %u", &i);
    printf("\nAddress of i = %u", j);
    printf("\nAddress of j = %u", &j);
    printf("\nValue of j = %u", j);
    printf("\nValue of i = %d", i);
    printf("\nValue of i = %d", *(&i));
    printf("\nValue of i = %d", *j);
    return 0;
}

Output:
Address of i = 2293532
Address of i = 2293532
Address of j = 2293528
Value of j = 2293532
Value of i = 3
Value of i = 3
Value of i = 3
Key Concepts Related to Pointers

- Declaring a pointer

```c
define *myIntPtr;
define int* myIntPtr;
```

- Getting the address of a variable

```c
int age = 3;
myIntPtr = &age;
```

- Dereferencing a pointer

```c
*myIntPtr = 5;
```

Note: We just changed the value of age!
Pointers Example 2: ptrExample.c

```c
#include <stdio.h>

int main()
{
    int myValue;
    int *myPtr;
    myValue = 15;
    myPtr = &myValue;
    printf("myValue is equal to : %d\n", myValue);
    *myPtr = 25;
    printf("myValue is equal to : %d\n", myValue);
}
```

Output:
myValue is equal to : 15
myValue is equal to : 25
Pointers and Arrays

The square-bracket array notation is a short cut to prevent you from having to do pointer arithmetic

```c
char array[5];
array[2] = 12;

array is a pointer to array[0]

array[2] = 12; is therefore equivalent to
*(array+2) = 12;
```
#include <stdio.h>

void addUpdate(int *a, int *b){
    int c;
    c = *a + *b;
    printf("Addition is : %d
", c);
    *a = c;
    *b = c;
}

int main(){
    int a, b;
    printf("Enter Any 2 Numbers : ");
    scanf("%d %d", &a, &b);
    printf("a is: %d, b is: %d\n", a, b);
    addUpdate(&a, &b);
    printf("a is: %d, b is: %d\n", a, b);
    return 0;
}

Note: The values of a and b changed in addUpdate function.
Output of passValue3.c

• Output:
Enter Any 2 Numbers : 2 8
a is: 2, b is: 8
Addition is : 10
a is: 10, b is: 10
Dynamic Memory Allocation

• Dynamic allocation is the automatic allocation of memory at run-time

• It is accomplished by two functions: `malloc` and `free`

• These functions are defined in the library file `stdlib.h`

• `malloc` allocates the specified number of bytes and returns a pointer to the block of memory

• When the memory is no longer needed, the pointer is passed to `free` which deallocates the memory

• Other functions:
  – `calloc` allocates the specified number of bytes and initializes them to zero
  – `realloc` increases the size of the specified chunk of memory

Note: With arrays, static memory allocation takes place, that is at compile-time.
#include<stdio.h>
#include<stdlib.h>

int main()
{
    int numStudents, avg, *ptr, i, sum = 0;
    printf("Enter the num of students :");
    scanf("%d",&numStudents);
    ptr=(int *)malloc(numStudents*sizeof(int));
    if(ptr == NULL){
        printf("\n\nMemory allocation failed!");
        exit(1);
    }
    for (i=0; i<numStudents; i++){
        printf("\nEnter the marks for the student %d\n", i+1);
        scanf("%d",(ptr+i));
    }
}
Example: dynMemAlloc.c (2)

```c
... for (i=0; i<numStudents; i++) {
    sum = sum + *(ptr + i);
}
avg = sum/numStudents;
printf("\nAvg marks = %d ",avg);
return 0;
} // end of main function
```

Output:
Enter the num of students : 3
Enter the marks for the student 1
10
Enter the marks for the student 2
20
Enter the marks for the student 3
30
Avg marks = 20
Consider the problem specified in Homework 5. Redo it by writing a user-defined function of your choice. For example,

- you could write a `printMatrix` function that can print the contents of a two-dimensional matrix of type double, and with any number of rows and columns.

- You could also write a `myMatMul` function for multiplying matrices. This function can accepts two integer arrays of same size, say \( P \times Q \), where \( P \) is the number of rows and \( Q \) is the number of columns.
References

• C Programming Language, Brian Kernighan and Dennis Ritchie

• Let Us C, Yashavant Kanetkar

• C for Dummies, Dan Gookin

• http://cplusplus.com