# BLM2031 YAPISAL PROGRAMLAMA – EKİM 2023
# Dr. Ögr. Üyesi H. Irem Türkmen
# GENEL BİLGİLER

## İLETİŞİM

- İletişim bilgileri
  - Oda : D-033
  - e-mail: irem@yildiz.edu.tr
  - İletişim için öncelikle e-mail gönderiniz, yüz yüze görüşme için randevu isteyiniz.

## DERS NOTLARI ve KAYNAKLAR

- Önceki katkıları için Z. Cihan Tayşi, Zeyneb Kurt ve Ahmet Elbir hocalarımıza teşekkür ederim.
- Darnell P. A. and Margolis P. E., C: A Software Engineering Approach, 3rd ed., Springer-Verlag, 1996 (notların oluşturulduğu asıl kaynaktır).

# BLM2031 YAPISAL PROGRAMLAMA – GENEL BİLGİLER

## BAŞARIM DEĞERLENDİRME

- Uygulama ve lab. çalışmaları:
  - Ekim ayı sonunda başlar, dönüşümlü yapılır.
  - Dersi alan tüm öğrenciler lab. çalışmalarına katılmak zorundadır.
- Ara sınav: 8.hafta
- Proje ödevi:  Ayrıntılar ileride duyurulacak
- Ara sınav mazereti: 14.hafta (yönetmelik kuralları uyarınca)
- Final sınavı: Final haftasında
- Bölümün sayfasında duyuracağı vize ve final programlarına göre, haftalar ve hatta günler ile saatler değişebilir.
- Puanlama (değişebilir):
  - 1. Ara sınav %25, Lab. %15, Proje %20, Final %40
  - Yapıl(a)mayan değerlendirmenin not ağırlığı yapılanlara paylaştırılır.

# BLM2031 YAPISAL PROGRAMLAMA – GENEL BİLGİLER

## DERS İÇERİĞİ

- Hatırlatma: C'de veri tipleri, Bitsel işlemler, Kontrol deyimleri, Döngüler, Diziler
- İşaretçiler: İşaretçiler Aritmetiği, diziler ve işaretçiler, İşaretçi Dizileri, Karakter Dizileri, İşaretçilerin İşaretçisi
- Dinamik Bellek Yönetimi ve Fonksiyonlar, Fonksiyon İşaretçileri, Özyineleme
- Yerel ve Global Değişkenler, Depolayıcı Sınıflar, Yapılar, Birlikler
- Dosya işlemleri
- C Önişlemcileri ve Makrolar
- Statik ve Dinamik Kütüphaneler

# BLM2031 YAPISAL PROGRAMLAMA – GENEL BİLGİLER

## ÖNEMLİ SENATO KARARLARI

- Öğrencinin ara sınav notunun %60'ı + Finalin %40'ı eğer "sayısal olarak" **40'ın altı**nda kalıyorsa öğrenci doğrudan "**FF** notu" **ile dersten kalmış sayılacaktır** (YN-027-YTÜ Önlisans ve Lisans Eğitim-Öğretim Yönetmeliği, Md. 26.e).
- Yarıyıl sonu sınavına girmeyen öğrenciler vize notuna bakılmaksızın ilgili dersten başarısız (FF) sayılırlar (YÖ-075-YTÜ Sınav Yönergesi, Md. 4.2.k).
- Bütün öğrencilere derslere devam zorunluluğu gelmiştir (dersi tekrar alanların önceki notu ne olursa olsun).
  - Derslere ait devam durumu ilgili öğretim üyesi tarafından yarıyıl sonu sınavları başlamadan önce öğrenci bilgi sisteminde ilan edilir.
  - Devamsızlıktan kalan öğrenciler yarıyıl sonu sınavına giremezler ve bu öğrencilerin ilgili derse ait başarı notu (F0) olarak bilgi sistemine işlenir (YÖ-075-YTÜ Sınav Yönergesi, Md. 4.2.h).

# A Fast Review of C Essentials Part I

Structural Programming

by Z. Cihan TAYSI

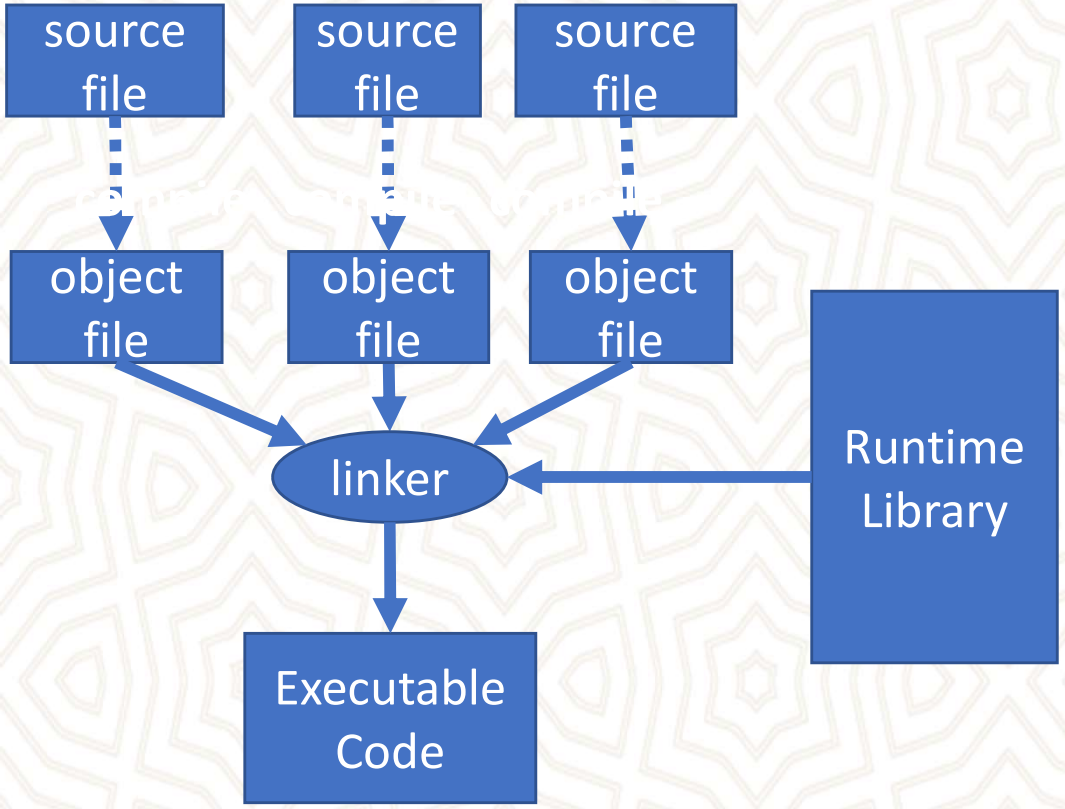
additions by Yunus Emre SELÇUK

# Outline

- Program development
- C Essentials
  - Variables & constants
  - Names
  - Functions
  - Formatting
  - Comments
  - Preprocessor
- Data types
- Mixing types

# Program Development

- The task of compiler is to translate source code into machine code
- The compiler's input is ***source code*** and its output is ***object code***.
- The linker combines separate object files into a single file
- The linker also links in the functions from the runtime library, if necessary.
- Linking usually handled automatically.

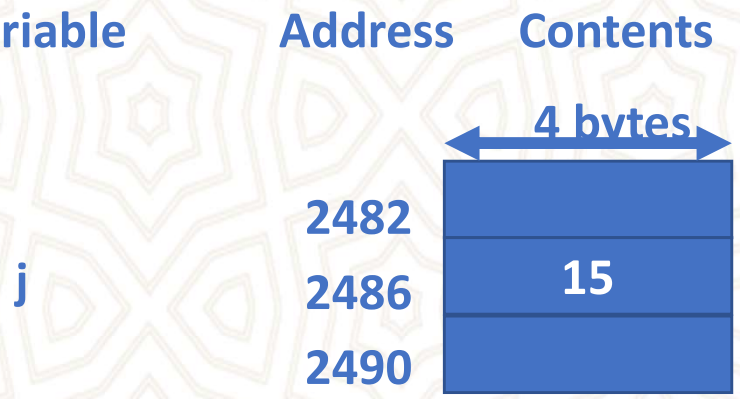


# Program Development CONT'D

- One of the reasons C is such a small language is that it defers many operations to ***a large runtime library.***
- The runtime library is a collection of object files
  - Each file contains the machine instructions for a function that performs one of a wide variety of services
    - The functions are divided into groups, such as I/O, memory management, mathematical operations, and string manipulation.
  - For each group there is a source file, called a ***header file***, that contains information you need to use these functions
    - by convention , the names for header files end with *.h* extention
- For example, one of the I/O runtime routines, called ***printf()***, enables you to display data on your terminal. To use this function you must enter the following line in your source file
  - #include <stdio.h>

# Variables & Constants

- The statement
  - j = 5 + 10;
- ***A constant*** is a value that never changes
- ***A variable*** achieves its variableness by representing a location, ***or address***, in computer memory.



# Names

- In the C language, you can name just about anything
  - variables, constants, functions, and even location in a program.
- Names may contain
  - letters, numbers, and the underscore character ( _ )
  - **but must start with a letter or underscore**...
- The C language is ***case sensitive*** which means that it differentiates between lowercase and uppercase letters
  - VaR, var, VAR
- A name **can NOT be** the same as one of the ***reserved keywords***.

# Names cont'd

- **LEGAL NAMES**
  - j
  - j5
  - __sesquipedalial_name_system_name
  - UpPeR_aNd_LoWeR_cAsE_nAmE
- **ILLEGAL NAMES**
  - 5j
  - $name
  - int
  - bad%#*@name

# Names cont'd

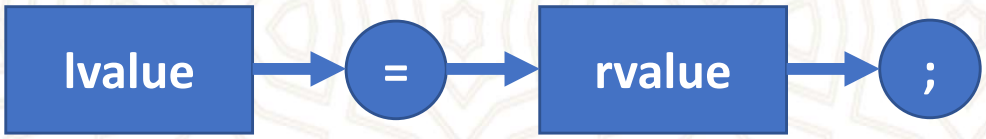
- reserved keywords = illegal names cont.'d:

| | | | |
|---|---|---|---|
| auto | double | int | struct |
| break | else | long | switch |
| case | enum | register | typedef |
| char | extern | return | union |
| const | float | short | unsigned |
| continue | for | signed | void |
| default | goto | sizeof | volatile |
| do | if | static | while |

# Expressions

- An ***expression*** is any combination of operators, numbers, and names that donates the computation of a value.
- **Examples**
  - 5 A literal
  - j A variable
  - 5 + j A constant plus a variable
  - f() A function call
  - f()/4 A function call, whose result is divided by a constant

# Assignment Statements




- The left hand side of an assignment statement, called an ***lvalue***, must evaluate to a memory address that can hold a value.
- The expression on the right-hand side of the assignment operator is sometimes called an ***rvalue***.

answer = num * num;

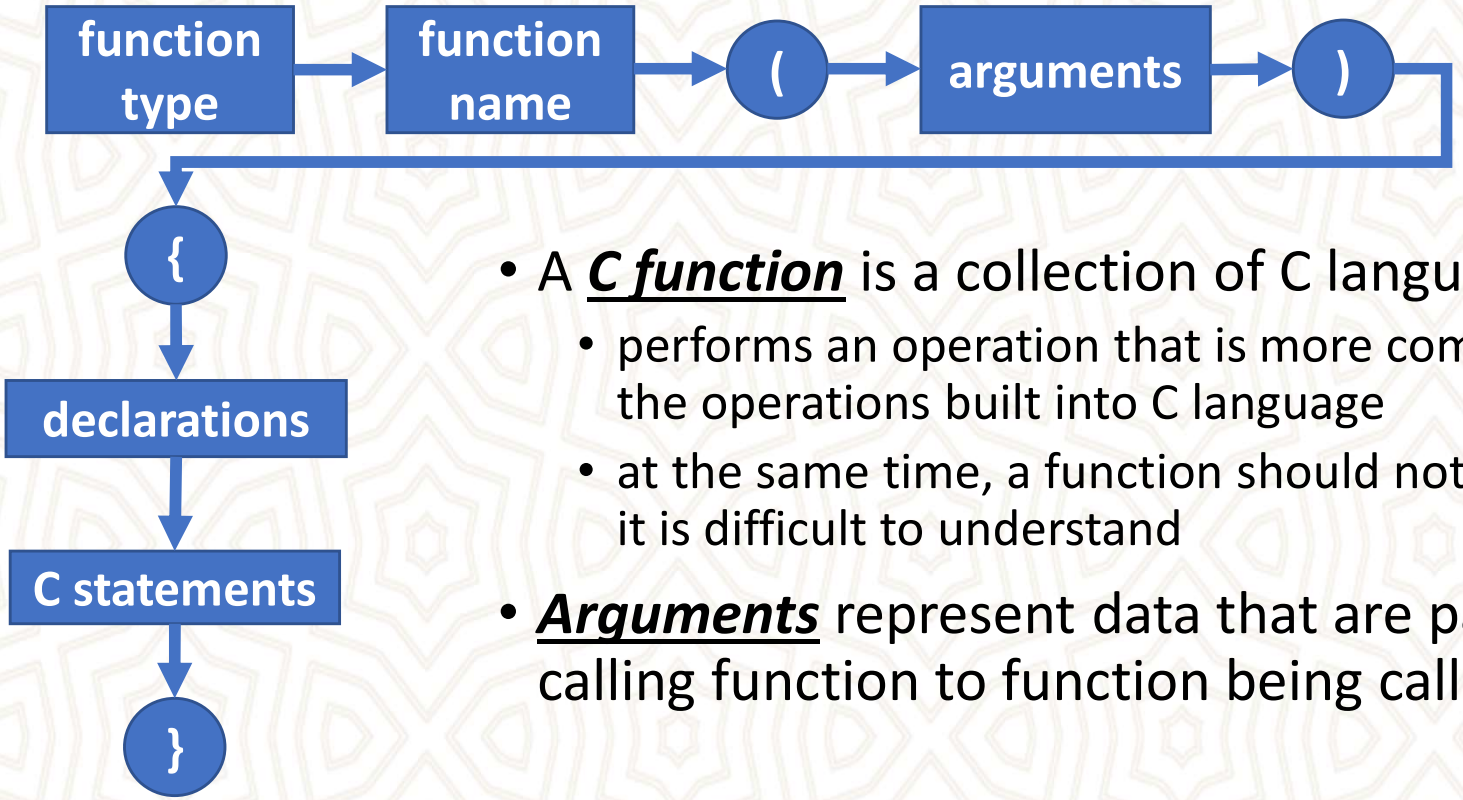
num * num = answer;

# Comments

- A comment is text that you include in a source file to explain what the code is doing!
  - Comments are for human readers – compiler ignores them!
- The C language allows you to enter comments between the symbols /* and */
- Nested comments are NOT supported
- **What to comment ?**
  - Function header
  - changes in the code

```
/* square()
 * Author : P. Margolis
 * Initial coding : 3/87
 * Params : an integer
 * Returns : square of its parameter
 */
```

# Functions




- A ***C function*** is a collection of C language operations.
  - performs an operation that is more complex than any of the operations built into C language
  - at the same time, a function should not be so complex that it is difficult to understand
- ***Arguments*** represent data that are passed from calling function to function being called.

# Functions

- You can write your own functions and you should do so!
  - Grouping statements that execute a sub-task under a function leads to modular software
  - You can reuse functions in different programs
  - Functions avoid duplicate code that needs to be corrected in multiple places of the entire program if a bug removal or change request emerges.
    - Bugs and requirement changes are inevitable in software development!

# Functions

- You should declare a function before it can be used …

```
int combination( int, int ); //This is also called allusion
void aTaskThatNeedsCombination( ) {
      //some code
      c = combination(a, b);
      //more code
}
int combination( int a, int b ) {
      //necessary code
}
```

# Functions

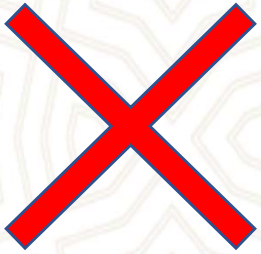
- … or the required function should be completely coded before it is called from another function.

```
int combination( int a, int b ) {
    //necessary code
}
void aTaskThatNeedsCombination( ) {
    //some code
    c = combination(a, b);
    //more code
}
```



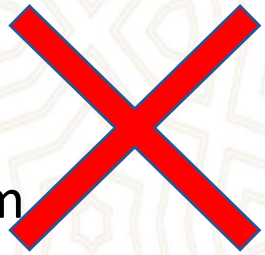
# Formatting Source Code

```
int square (int num)  {
int answer;
answer = num * num;
return answer;
}
```



```
int square (int num)  {
        int answer;
        answer = num * num;
        return answer;
}
```

```
int square (int num)  {
 int
answer;
          answer   = num
* num;
return answer;
}
```



```
int square ( int num )
{
        int answer;
        answer = num * num;
        return answer;
}
```

# The main() Function

- All C programs must contain a function called ***main()***, which is always the first function executed in a C program.
- It can take two arguments but we need to learn much more before going into details.
- When ***main()*** returns, the program is done.

```
int main ( ) {
        extern int square();
        int solution;
        solution = square(5);
        exit(0);
}
```

- The ***exit()*** function is a runtime library routine that causes a program to end, returning control to operating system.
  - If the argument to exit() is zero, it means that the program is ending normally without errors.
  - Non-zero arguments indicate abnormal termination of the program.
  - Calling exit() from main() is exactly the same as executing ***return*** statement.

# printf() and scanf() Functions

```
int num;
scanf("%d", &num);
printf("num : %d\n", num);
```

- The printf() function can take any number of arguments.
  - The first argument called the ***format string***. It is enclosed in double quotes and **may contain** text and ***format specifiers***
- The scanf() function is the mirror image of printf(). Instead of printing data on the terminal, it reads data entered from keyboard.
  - The first argument is a format string.
  - **The major difference between scanf() and printf() is that the data item arguments must be lvalues**
  - **Scanf requires a memory address as 2nd parameter, hence comes the &**

# Preprocessor

- The preprocessor executes automatically, when you compile your program
- All preprocessor directives begin with pound sign (#), which must be the first non-space character on the line.
  - unlike C statements a preprocessor directive ends with a newline, **NOT a semicolon**
- It is also capable of
  - macro processing
  - conditional compilation
  - debugging with built-in macros

# Preprocessor cont'd

- The ***define*** facility
  - it is possible to associate a name with a constant
    - #define NOTHING 0
  - It is a common practice to all uppercase letters for constants
  - naming constants has two important benefits
    - it enable you to give a descriptive name to a nondescript number
    - it makes a program easier to change
  - be careful NOT to use them as variables
    - **NOTHING = j + 5**

# Preprocessor cont'd

- The ***include*** facility
  - #include directive causes the compiler to read source text from another file as well as the file it is currently compiling
  - the #include command has two forms
    - #include <filename>
      - **the preprocessor looks in a special place designated by the operating system. This is where all system include files are kept.**
    - #include "filename"
      - **the preprocessor looks in the directory containing the source file. If it can not find the file, it searches for the file as if it had been enclosed in angle brackets!!!**

# hello world!!!

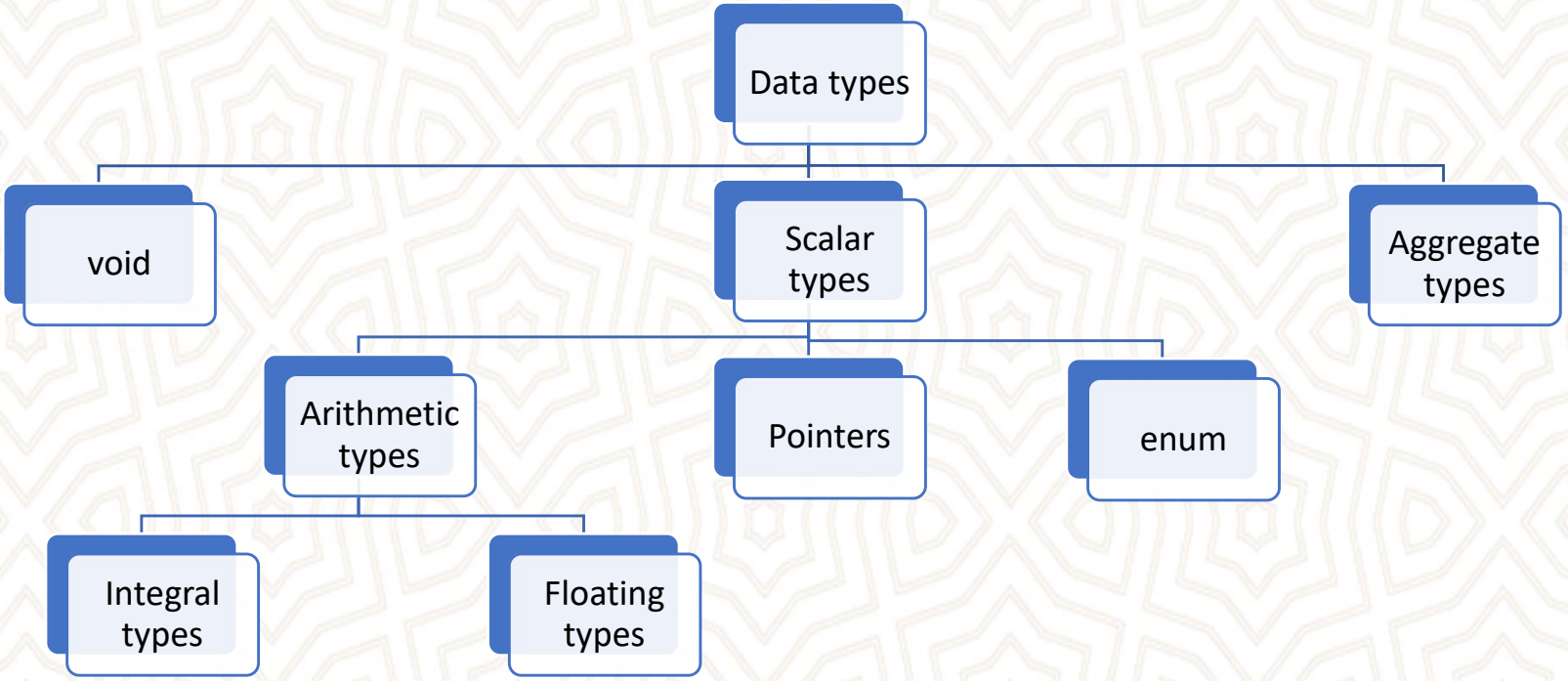
```
#include <stdio.h>

int main ( void ) {

        printf("Hello World...\n");
        return 0;
}
```

- include standard input output library
- start point of your program
- return a value to calling program
  - in this case 0 to show success?

# Data Types


Data types
void
Scalar types
Aggregate types
Arithmetic types
Pointers
enum
Integral types
Floating types

# Data Types cont'd

- There are 9 reserved words for scalar data types
- Basic types
  - char, int, float, double, enum
- Qualifiers
  - long, short, signed, unsigned
- To declare j as an integer
  - int j;

| | | | |
|---|---|---|---|
| **char** | **double** | **short** | **signed** |
| **int** | **enum** | **long** | **unsigned** |
| **float** | | | |

- You can declare variables that have the same type in a single declaration
  - int j,k;
- ***All declarations in a block must appear before any executable statements***

# Different Types of Integers

- The only requirement that the ANSI Standard makes is that a byte must be ***at least 8 bits long***, and that ints must be ***at least 16 bits long*** and must represent the "***natural***" size for computer.
  - natural: the number of bits that the CPU usually handles in a single instruction

| Type | Size (in bytes) | Value Range | Format String |
|---|---|---|---|
| int | 4 | $-2^{31}$ to $2^{31}-1$ | %d |
| unsigned int | 4 | 0 to $2^{32}-1$ | %u |
| short int | 2 | $-2^{15}$ to $2^{15}-1$ | %hi |
| long int | 4 | $-2^{31}$ to $2^{31}-1$ | %li |
| unsigned short int | 2 | 0 to $2^{16}-1$ | %hu |
| unsigned long int | 4 | 0 to $2^{32}-1$ | %lu |
| signed char | 1 | $-2^{7}$ to $2^{7}-1$ | %c |
| unsigned char (rather meaningless) | 1 | 0 to $2^{8}-1$ | %uc |

# Format Strings for Integers

- A format string determines the representation of a value in output (printf) and the interpretation of a value in input (scanf).
- Try the following code with different values:

```c
#include <stdio.h>

int main()
{

    int sayiN = -65, sayiP=65;
    printf("     int \t%d\n",sayiN);     //-65
    printf(" uns.int \t%u\n",sayiN);     //4294967231
    printf(" srt.int \t%hi\n",sayiN);    //-65


    printf(" lng.int \t%li\n",sayiP);    //warning --- 65
    printf("     int \t%d\n",sayiP);     //65
    printf("     char\t%c\n",sayiP);     //A

    printf("     char\t%d\n",'a');       //97

    return 0;
}
```

# Different Types of Integers cont'd

- Integer literals
  - ***d***ecimal (%d), ***o***ctal (%o), He***x***adecimal (%x)

| Decimal | Octal (leading 0 zero) | Hexadecimal (leading 0x zeroX, case insensitive) |
|---|---|---|
| 3 | 003 | 0x3 |
| 8 | 010 | 0x8 |
| 15 | 017 | 0xF |
| 16 | 020 | 0x10 |
| 21 | 025 | 0x15 |
| -87 | -0127 | -0x57 |
| 255 | 0377 | 0xFF |

- In general, an integer constant has type int, if its value can fit in an int. Otherwise it has type long int.
- Suffixes
  - u or U (for unsigned)
  - l or L (for long)

```
printf("    octal\t%d\n",010);    //8

printf("    octal\t%o\n",010);    //8
```

# Floating Point Types

- to declare a variable capable of holding floating-point values
  - ***float (%f)***
  - ***Double (%lf)***
- The word ***double*** stands for double-precision
  - it is capable of representing about twice as much precision as a ***float***
  - A float generally requires ***4 bytes***, and a double generally requires ***8 bytes***
  - ***read more about limits in <limits.h>***
- Decimal point
  - 0.356
  - 5.0
  - 0.000001
  - .7
  - 7.
- ***Scientific notation (%e)***
  - 3e2
  - 5E-5
- Long double can be defined but they can become plain double in some computer platforms
- Refer to the source book and the Internet for different representation format modifiers (such as %5.7f)

# Format Strings for Real Numbers

```
#include <stdio.h>

int main(int argc, char *argv[]){
        float ondalikli = 700.555;
        printf("     dbl \t%.3f\n",ondalikli);
        printf("     exp \t%e\n",ondalikli);

        system("PAUSE");
        return 0;
}
```

# Initialization

- A declaration allocates memory for a variable, but it does not necessarily store an initial value at the location
  - ***If you read the value of such a variable before making an explicit assignment, the results are unpredictable***
- To initialize a variable, just include an assignment expression after the variable name
  - char ch = 'A' ;
- It is same as
  - char ch;
  - ch = 'A';

# Enumeration Data Type

```
enum { red, blue, green, yellow } color;
enum { bright, medium, dark } intensity;

color = yellow;          // OK
color = bright;          // Type conflict
intensity = bright;      // OK
intensity = blue;        // Type conflict
color = 1;               // Type conflict
color = green + blue; // Misleading usage
```

- ***Enumeration types*** enable you to declare variables and the set of named constants that can be legally stored in the variable.
- You can override default values by specifying other values

# void Data Type

- The void data type has two important purposes.
- The first is to indicate that a function does not return a value
  - void func (int a, int b);
- The second is to declare a generic pointer
  - **We will discuss it later !**



# typedef

- ***typedef*** keyword lets you create your own names for data types.
- Semantically, the variable name becomes a synonym for the data type.
- By convention, typedef names are capitalized.

```
typedef long int INT32;

long int j;
INT32 j;
```