



BLM203 I

Structured Programming

Zeyneb KURT
25.09.2018



Contact

- Contact info
 - office : D-219
 - e-mail zeynebkurt@gmail.com , zeyneb@yildiz.edu.tr
- When to contact
 - e-mail first,
 - take an appointment
- What to expect
 - help on your problems about the course
 - you should have “spent some time” on it !!!



General Information

- **Lecture Hours:**

- Tuesday between 09:00-12:00

- **Lab Hours:**

- The schedule will be announced. Please check webpage at <http://avesis.yildiz.edu.tr/zeyneb/documents>



Course Outline

- **Review of “Introduction to Procedural Programming” course**
 - Data types, control flow, and so on...
- **C preprocessor**
- **Storage classes**
 - Scope & duration
- **Arrays**
 - Strings, multi-dimensional arrays, pointers
- **Dynamic memory allocation**
 - Multi-dimensional



Course Outline Cont'd

- **Functions**

- parameter passing, return values
- recursion
- function pointers

- **Structures**

- structures, unions
- linked lists

- **FILE I/O**

- high level operations
- file descriptors, low level functions



Course Metarial

- **Lecture notes**

- Slides will be available at <http://avesis.yildiz.edu.tr/zeyneb/documents>

- **Web resources**

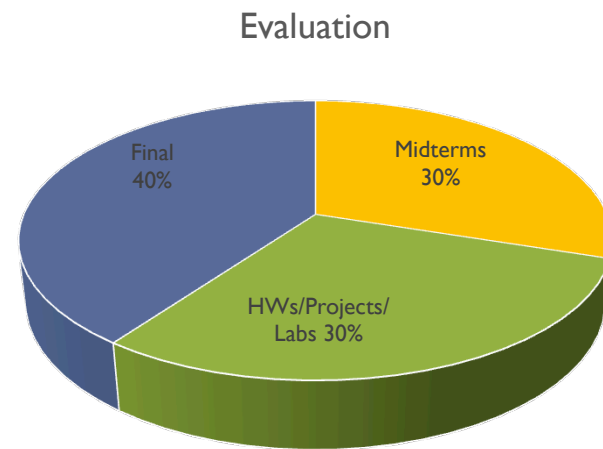
- Always cite your resources!

- **Book :**

- Peter A. Darnell and Philip E. Margolis, *C:A Software Engineering Approach*, Springer-Verlag, 1996 (3rd) edition.

Grading Policy

- Two midterms
 - 8th week
 - 12th or 13th week
- Homeworks
 - a few
- Labs (4+1)
 - **will be announced**
- One final exam
- You have to attend 70% of the theoretical lectures and 80% of the labs





Please Do NOT !!!

- try to take credit for work done by someone else.
- reuse the same work for more than one class.
- use any resource not permitted by the instructor during an exam.
- communicate with another person during an exam.



A fast review of C

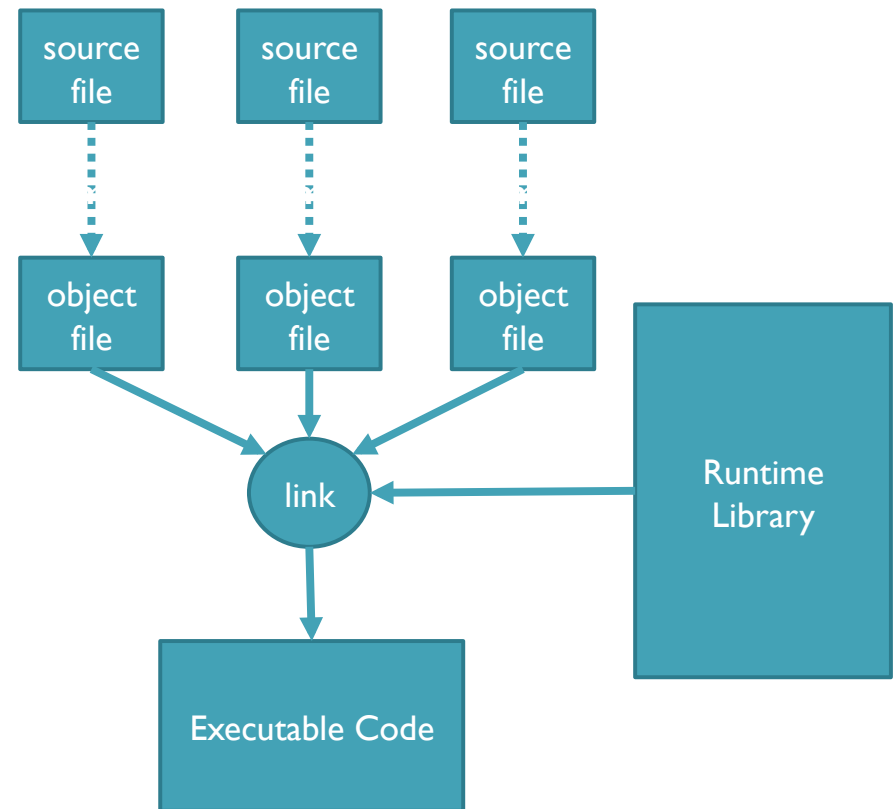


Outline

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

program development

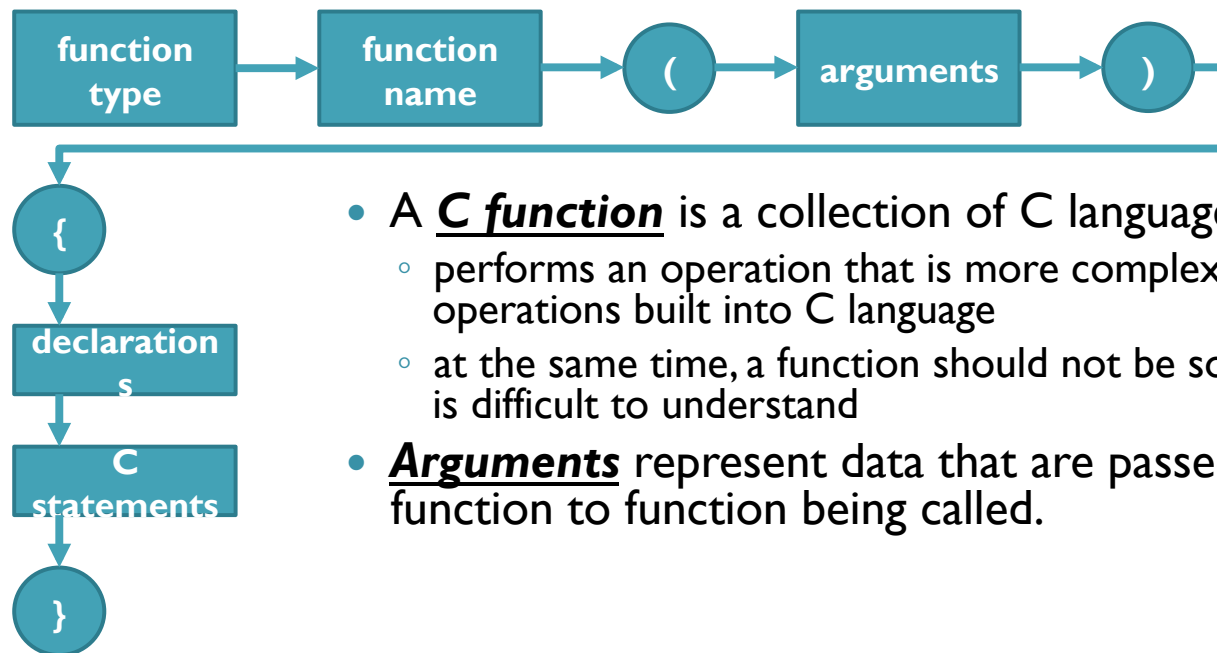
- The task of compiler is to translate source code into machine code
- The compiler's input is **source code** and output is **object code**.
- The linker combines separate object files into a single file and forms an exe file
- The linker also links in the functions from the runtime library, if necessary.
- 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>`

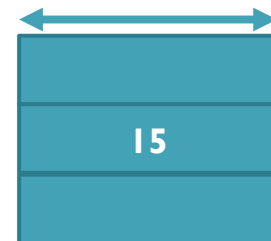
C essentials



- 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.

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
- __system_name
- sesquipedalial_name
- UpPeR_aNd_LoWeR_cAsE_nAmE

- **ILLEGAL NAMES**

- 5j
- \$name
- int
- bad%#*@name

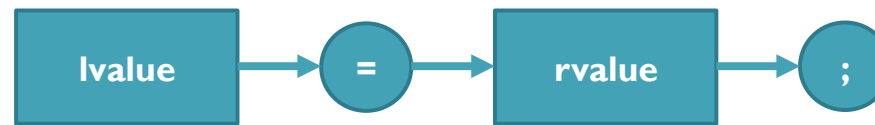
names cont'd - reserved keywords

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 denotes the computation of a value.
- **Examples**
 - 5 A constant
 - 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;

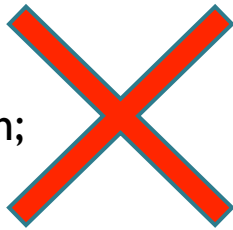


num * num = answer;

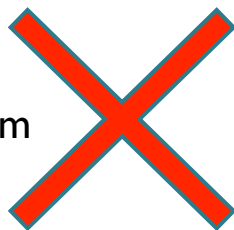


formatting source code

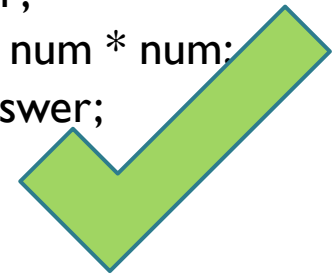
```
int square (num) {  
int answer;  
answer = num * num;  
return answer;  
}
```



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



```
int square (num) {  
    int answer;  
    answer = num * num;  
    return 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 : 3/2017  
 * Params : an integer  
 * Returns : square of its  
 parameter  
 */
```

THE main() FUNCTION

```
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 a main() function is exactly the same as executing **return** statement.



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 with a semicolon**
- It is also capable of
 - macro processing
 - conditional compilation
 - debugging with built-in macros

PRINTF() and SCANF() functions

- 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**

```
int num;
```

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


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 enables 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>
```

- include standard input output library

```
int main ( void ) {
```

- starting point of your program

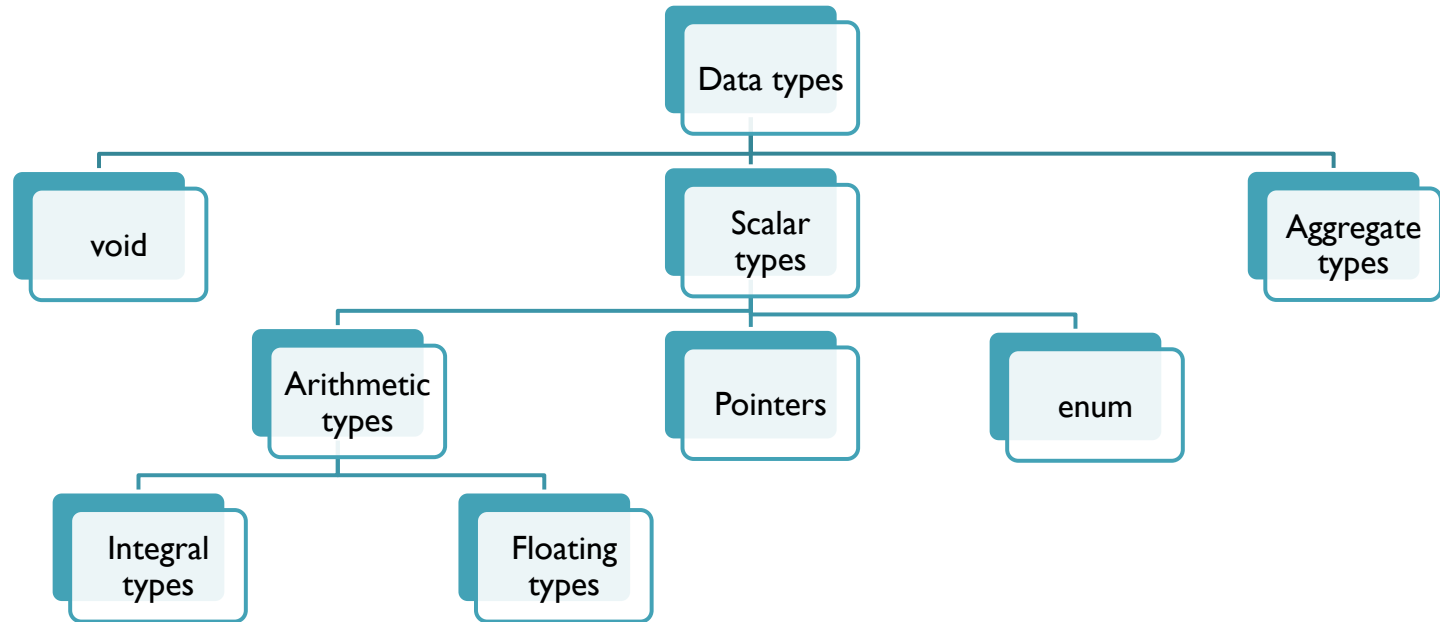
```
    printf(“Hello World...\n”);
```

```
    return 0;
```

```
}
```

- return a value to the calling program
 - in this case 0 to show success?

data types



data types

- 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;
- 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**

char	double	short	signed
int	enum	long	unsigned
float			

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 means the number of bits that the CPU usually handles in a single instruction

Type	Size (in bytes)	Value Range
int	4	-2^{31} to $2^{31} - 1$
short int	2	-2^{15} to $2^{15} - 1$
long int	4	-2^{31} to $2^{31} - 1$
unsigned short int	2	0 to $2^{16} - 1$
unsigned long int	4	0 to $2^{32} - 1$
signed char	1	-2^7 to $2^7 - 1$
unsigned char	1	0 to $2^8 - 1$

different types of integers

Integer constants

Decimal

Octal

Hexadecimal

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

l or L

Decimal	Octal	Hexadecimal
3	003	0x3
8	010	0x8
15	017	0xF
16	020	0x10
21	025	0x15
-87	-0127	-0x57
255	0377	0xFF

floating point types

- to declare a variable capable of holding floating-point values
 - **float**
 - **double**
- 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
 - 3e2
 - 5E-5

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 use
```

- **Enumeration types** enable you to declare variables and the set of named constants that can be legally stored in the variable.
- The default values start at zero and go up by one with each new name.
- You can override default values by specifying other values

- 
- `#include<stdio.h>`
 -
 - `enum week{Mon,Tue,Wed,
Thur, Fri, Sat, Sun};`
 - `int main()`
 - `{`
 - `enum week day;`
 - `day = Wed;`
 - `printf("%d",day);`
 - `return 0;`
 - `}`



VOID DATATYPE

- 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 !**

TYPedefs

- **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.

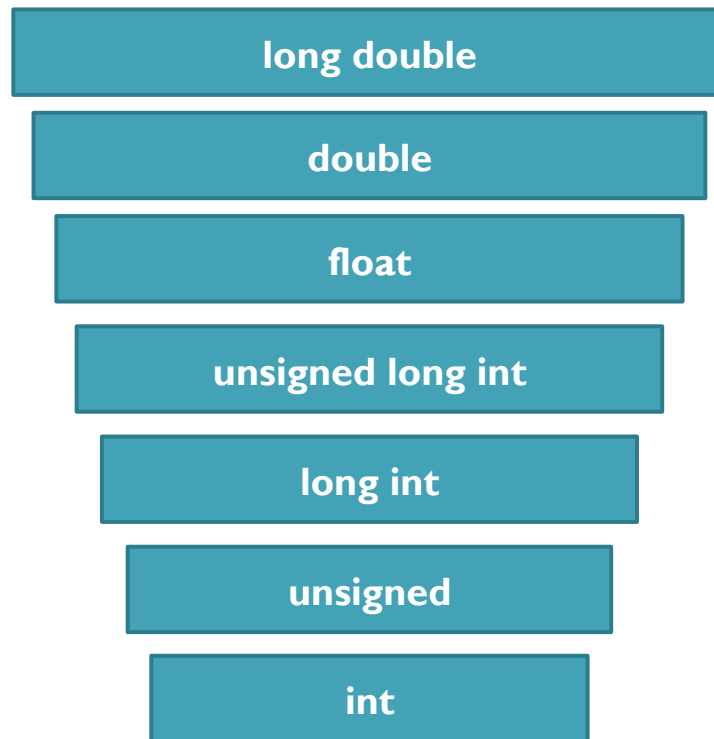
```
typedef long int int_32;  
long int j;  
int_32 j;
```



Mixing types

- Implicit conversion
 - Mixing signed and unsigned types
 - Mixing integers with floating point types
 - Explicit conversion
-

Mixing types



* Many computers perform arithmetic operations with floats faster than with doubles. You should only use larger types (doubles or long doubles) if you need the greater precision.

IMPLICIT CONVERSIONS

- When the compiler encounters an **expression**, it divides it into **subexpressions**, where each expression consists of one operator and one or more objects, called **operands**, that are **bound to the operator**.
- **Ex :** $- 3 / 4 + 2.5$ # The expression contains three operators $-, /, +$
- Each operator has its own rules for operand type agreement, but most binary operators require both operands to have the same type.
 - If the types differ, the compiler converts one of the operands to agree with the other one.
 - For this conversion, compiler resorts to the hierarchy of data types. **(Please check previous slide)**
- **Ex :** $1 + 2.5$ # involves two types, an int and a double



MIXING SIGNED AND UNSIGNED TYPES

- The only difference between signed and unsigned integer types is the way they are interpreted.
 - They occupy same amount of storage
- 11101010
 - has a decimal value of -22 (in two's complement notation)
 - An unsigned char with the same binary representation has a decimal value of 234
- 10u – 15 = ?
 - – 5
 - 4,294,967,291

MIXING INTEGERS WITH FLOATING TYPES

- Invisible conversions

```
int j;
```

```
float f;
```

```
j + f;           // j is converted to float
```

```
j + f + 2.5;    // j & f both converted to double
```

- **Loss of precision**

```
j = 2.5;         // j's value is 2
```

```
j = 9999999999999.888888 // overflow
```

EXPLICIT CONVERSIONS - CASTS

```
int j=2, k=3;  
float f;
```

```
f = k / j ;
```

```
f = (float) k / j;
```

- Explicit conversion is called casting and is performed with a construct called a cast
- To cast an expression, enter the target data type enclosed in parenthesis directly before expression