

Structured Programming

Dr. H. İrem Türkmen

Course Outline

- **Review of “Introduction to Computer Engineering II” course**
 - data types, control flow, and so on...
- **Arrays**
 - Strings, multi-dimensional arrays, pointers
- **Dynamic memory allocation**
 - multi-dimensional
- **Functions**
 - parameter passing, return values
 - recursion
 - function pointers

Course Outline Cont'd

- **Structures**
 - structures, unions
 - linked lists
- **Storage classes**
 - scope & duration
- **C preprocessors**
- **FILE I/O**
 - high level operations
 - file descriptors, low level functions
- **Multithreaded Programming**

Course Material

- **Lecture notes**
 - slides available at www.ce.yildiz.edu.tr
- **Book :** Darnell P. A. and Margolis P. E., C: A Software Engineering Approach, 1996 (3rd) edition



What is C?

- Dennis Ritchie
AT&T Bell Laboratories – 1972
- Widely used today
 - extends to newer system architectures
 - efficiency/performance
 - low-level access



Features of C

- Few keywords
- Structures, unions – compound data types
- Pointers – memory management
- External standard library – I/O, other facilities
- Compiles to native code
- Macro preprocessor



Versions of C

Evolved over the years:

- 1972 – C invented
- 1978 – *The C Programming Language* published; *first specification of language*
- 1989 – C89 standard (known as ANSI C or Standard C)
- 1990 – ANSI C adopted by ISO, known as C90
- 1999 – C99 standard
 - mostly backward-compatible
 - not completely implemented in many compilers
- 2007 – work on new C standard C1X announced



What is C used for?

- Systems programming
 - Operating Systems like Linux
 - Microcontrollers: automobiles and airplanes
 - Embedded processors : phones, portable electronics, etc.
 - DSP processors: digital audio and TV systems
 - ...

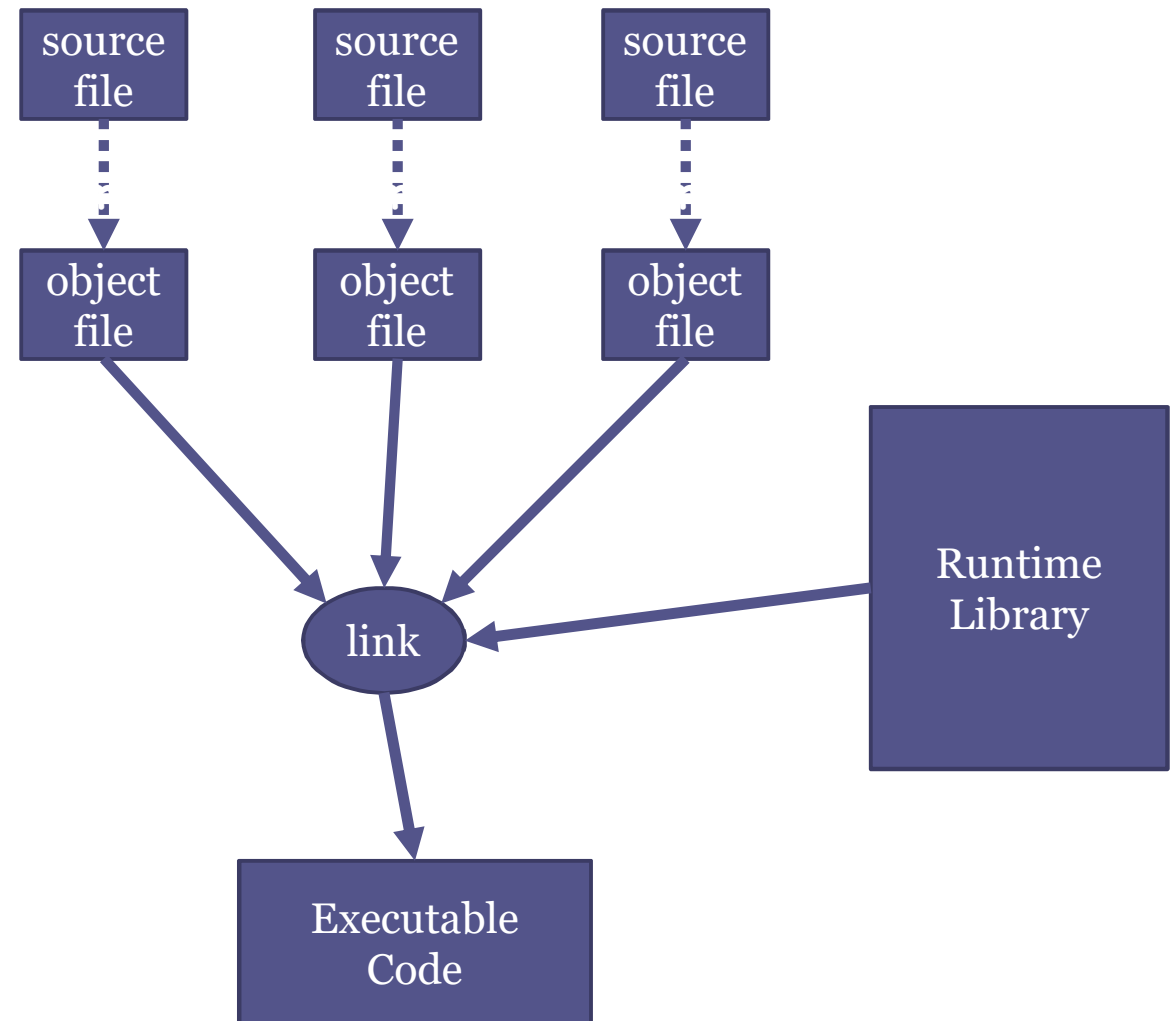


C Related Languages

- More recent derivatives: C++, Objective C, C#
- Influenced: Java, Perl, Python (quite different)
- C lacks:
 - exceptions
 - range-checking
 - garbage collection
 - object-oriented programming
 - polymorphism...
- Low-level language faster code

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.



C Runtime Library

- C 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>`

Compilers / Editors&IDEs

- IDE (Integrated Development Environments)
 - DevC++ (gcc compiler), <http://www.bloodshed.net>
 - Windows, Linux
 - Code::Blocks (gcc, Borland CC, Digital Mars, Open Watcom), <http://www.codeblocks.org>
 - Windows, Linux, Mac OS X
 - Xcode (gcc), <https://developer.apple.com/xcode/>
 - Mac OS X
 - Eclipse, <http://www.eclipse.org/cdt/>
- Editors
 - Ultrapad
 - Notepad++



Writing C Programs in Linux

- .c extension
- gcc compiler
- Terminal
 - `gcc bubbleSort.c -o bubbleSort`
 - `./bubbleSort`

Anatomy of a C Program

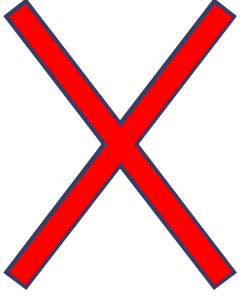
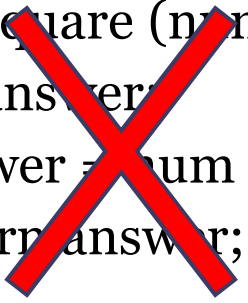
```
/*Sum of two numbers*/ // Comments
# include<stdio.h>      // Preprocessor Commands
int topl (int, int);    // Function prototypes

int main() // Main function
{
    int a,b,c; // Variables
    scanf ("%d %d",&a,&b); //Commands & Expressions
    c=topl(a,b);
    printf("c:%d",c);
    exit (0);
}

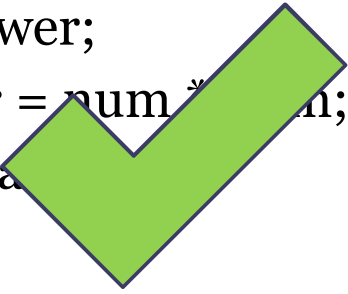
int topl (int sayi1, int sayi2) // Functions
{
    return sayi1+sayi2;
}
```

Formatting source code

```
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
 * Params : an integer
 * Returns : square of its
parameter
 */
```




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

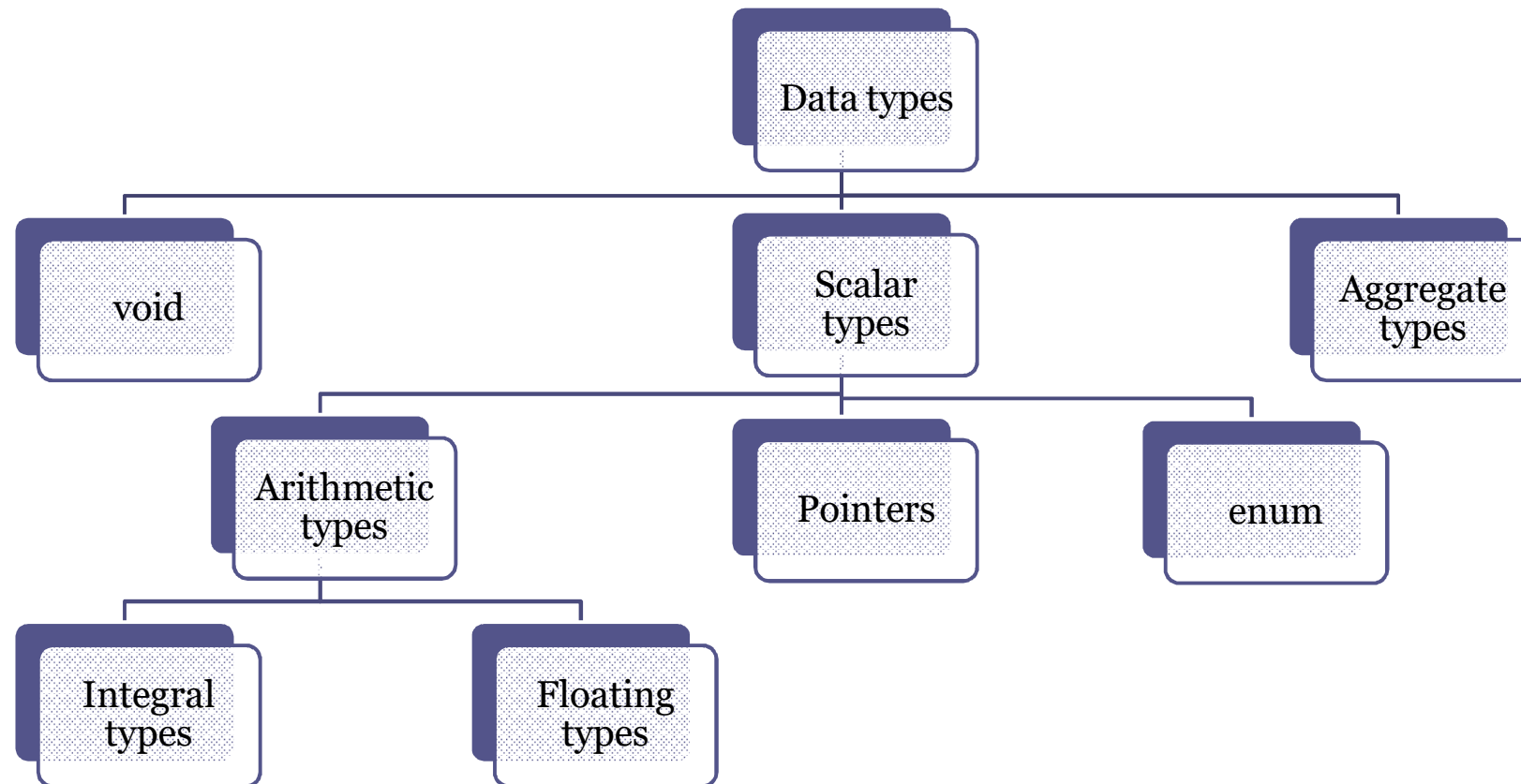
- The ***include*** facility
 - `#include` directive causes the compiler to read source text from another file as well as the file it is currently compiling
 - the `#include` command has two forms
 - `#include <filename>`
 - **the preprocessor looks in a special place designated by the operating system. 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!!!**

THE main() FUNCTION

```
int main ( ) {  
    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.

Data Types



Data Types

- Basic types
 - char, int, float, double, enum
- Qualifiers
 - long, short, signed, unsigned (can be used with char and int)
- To declare j as an integer
 - int j;
- You can declare variables that have the same type in a single declaration
 - int j,k;
- Initialization
 - char letter='z';

Types	Qualifiers
char	short
int	long
float	signed
Double	unsigned
enum	



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

Scalar Data Types

Data Type	Description	Size (byte)	Bottom Limit	Upper Limit
char	Only one character or very small integer	1	-128	127
unsigned char		1	0	255
short int	Small integer	2	-32768	32767
unsigned short int		2	0	65536
int	integer	4	-2,147,483,648	2,147,483,647
unsigned int		4	0	4,294,967,295
long int	large integer	8	-2^{63}	$2^{63}-1$
unsigned long int		8	0	$2^{64}-1$
float	Floating number	4	E-38	E+38
double		8	E-308	E+308

Different Types of Integers

- Literals
 - Prefix
 - 0x or 0X
 - 0
 - nothing
 - Suffixes
 - u or U
 - L or l
 - F or f
- 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**.

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



Reserved Keywords

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



Mixing Types

- Implicit Conversion
 - Assignment Conversions
 - Integral Widening Conversions
- Mixing signed and unsigned types
- Explicit conversion



Hierarchical Data Representation

long double

double

float

unsigned long int

long int

unsigned int

int

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
```

- **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



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;
```

printf() and scanf() functions

Symbol	Data Type
%c	char
%d	integer
%s	string (character array)
%ld	long integer
%u	unsigned integer
%lu	unsigned
%f	float
%lf	double
%x	hexadecimal

Precedence Level of Operators

1	() [] -> .	Grouping, scope, array/member access
2	! ~ - + * & sizeof type cast ++ --	(most) unary operations, sizeof and type casts
3	* / %	Multiplication, division, modulo
4	+ -	Addition and subtraction
5	<< >>	Bitwise shift left and right
6	< <= > >=	Comparisons: less-than, ...
7	== !=	Comparisons: equal and not equal
8	&	Bitwise AND
9	^	Bitwise exclusive OR
10		Bitwise inclusive (normal) OR
11	&&	Logical AND
12		Logical OR
13	?:	Conditional expression (ternary operator)
14	= += -= *= /= %= &= = ^= <<= >>=	Assignment operators
15	,	Comma operator