Introduction to C

Atiya Jokhio

We'll Learn Today:

- History of C
- Characteristics of C
- Writing C Programs
- Compilation
- Anatomy of C Program
- Constants and Variables
- printf() Function
- Format Specifiers
- Escape Sequences
- scanf() Function
- Operators

History of C

The *milestones* in C's development as a language are listed below:

- C evolved from two previous languages, BCPL and B.
- BCPL was developed as a language for writing operating-systems software and compilers.
- Ken Thompson modeled many features in his B language after their counterparts in BCPL, and in 1970 he used B to create early versions of the UNIX operating system
- The C language was evolved from B by Dennis Ritchie at Bell Laboratories and was originally implemented in 1972.

C has now become a widely used professional language for various reasons.

- It has high-level constructs.
- It can handle low-level activities.
- It produces efficient programs.
- It can be compiled on a variety of computers.

Writing C Program

- A programmer uses a text editor to create or modify files containing C code.
- Code is also known as source code.
- A file containing source code is called a source file.
- After a C source file has been created, the programmer must invoke the C compiler before the program can be executed (run).

3 Stages of Compilation

Stage 1: Preprocessing

- Performed by a program called the preprocessor
- Modifies the source code (in RAM) according to preprocessor directives (preprocessor commands) embedded in the source code
- skips comments and white space from the code

3 Stages of Compilation (con't)

Stage 2: Compilation

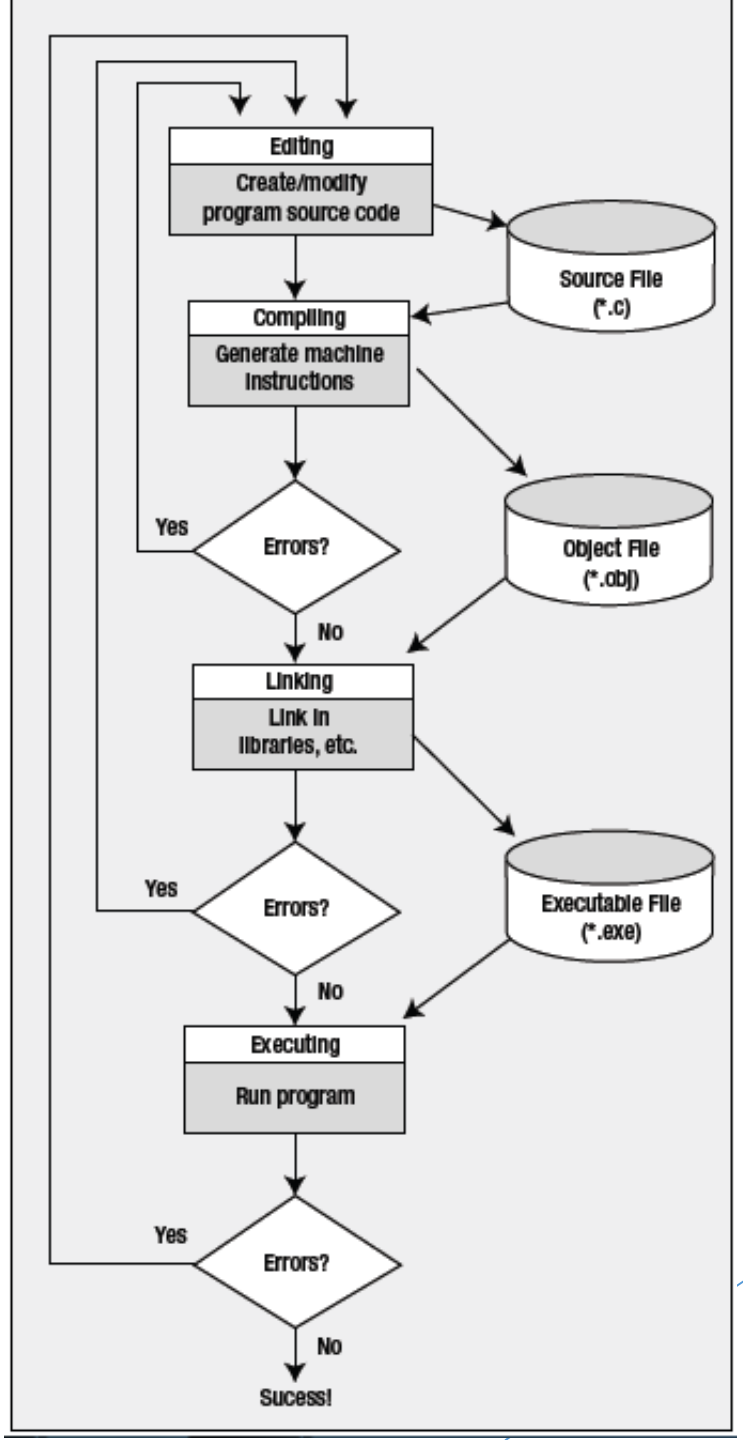
- Performed by a program called the compiler
- Translates the preprocessor-modified source code into object code (machine code)
- Checks for syntax errors and warnings
- Saves the object code to a disk file, if instructed to do so (we will not do this).
  - If any compiler errors are received, no object code file will be generated.
  - An object code file will be generated if only warnings, not errors, are received.

3 Stages of Compilation (con't)

Stage 3: **Linking**

- Combines the program object code with other object code to produce the executable file.
- The other object code can come from the **Run-Time Library**, other libraries, or object files that you have created.
- Saves the executable code to a disk file.
  - If any linker errors are received, no executable file will be generated.

Architectural Diagram

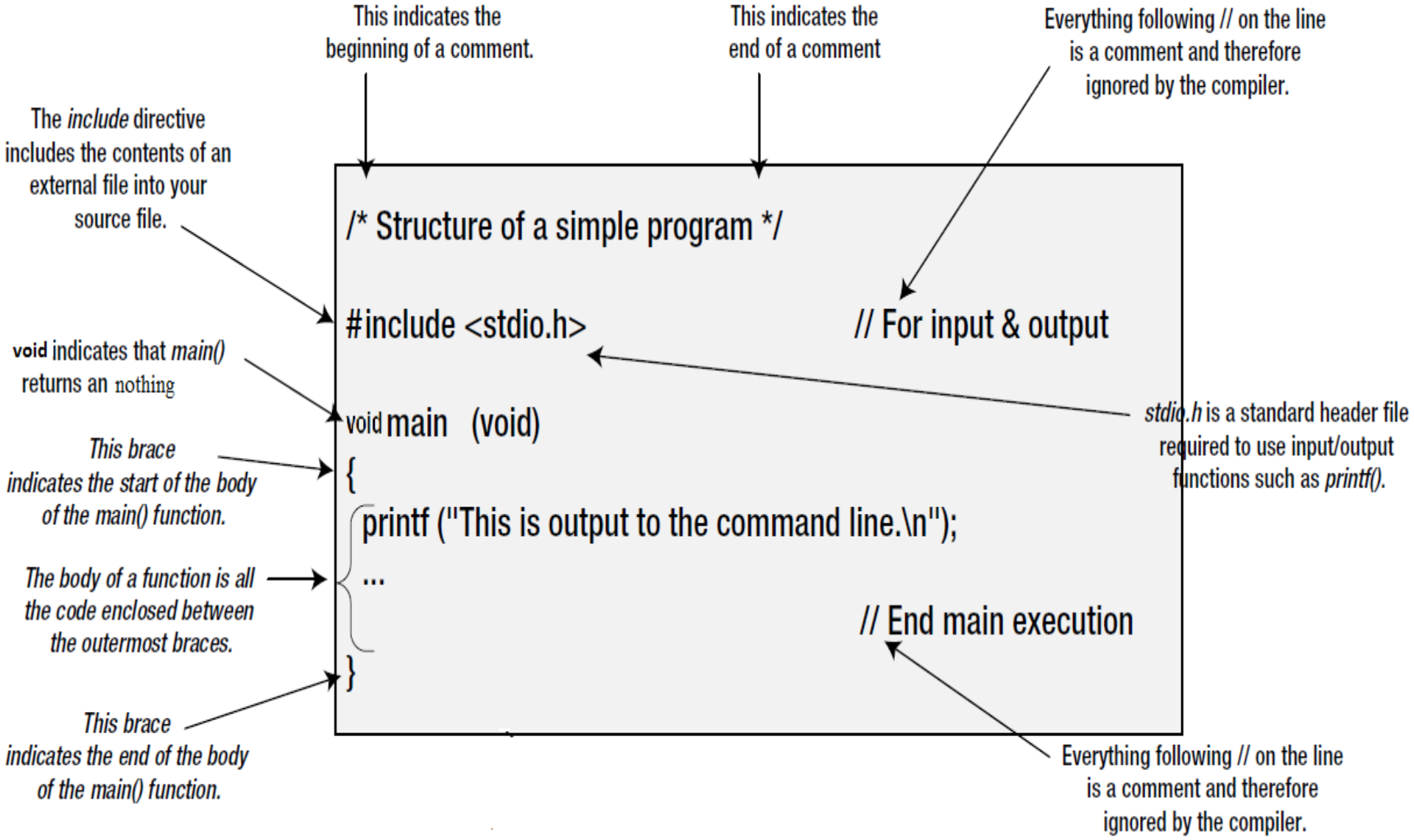


A Simple C Program

```
/* Filename:        First.c
   Author:          FAST
   Description:     This program prints the greeting
                    “Hello, World!”
*/

#include  <stdio.h>
void main ( void )
{
    printf ( “This is  class CS118!\n” ) ;
    getch( ) ;
}
```

Anatomy of a C Program



Program Header Comment

- A **comment** is descriptive text used to help a reader of the program understand its content.
- All comments must begin with the characters /* and end with the characters */
- These are called **comment delimiters**
- The program header comment always comes first.

Preprocessor Directives

- Lines that begin with a # in column 1 are called **preprocessor directives** (**commands**).
- Example: the **#include <stdio.h>** directive causes the preprocessor to include a copy of the standard input/output header file **stdio.h** at this point in the code.
- This header file was included because it contains information about the printf ( ) function that is used in this program.

stdio.h

- When we write our programs, there are libraries of functions to help us so that we do not have to write the same code over and over.
- Some of the functions are very complex and long. Not having to write them ourselves make it easier and faster to write programs.
- Using the functions will also make it easier to learn to program!

void main (void)

- Every program must have a function called main. This is where program execution begins.
- main() is placed in the source code file as the first function for readability.
- The reserved word “void” indicates that main() returns nothing.
- The parentheses following the reserved word “main” indicate that it is a function.
- The reserved word “void” means nothing is there.

The Function Body

- A left brace (curly bracket) -- **{** -- begins the **body** of every function. A corresponding right brace -- **}** -- ends the function body.
- The style is to place these braces on separate lines in column 1 and to indent the entire function body 3 to 5 spaces.

printf (“Hello, World!\n”);

- This line is a C statement.
- It is a call to the function printf ( ) with a single argument (parameter), namely the string “Hello, World!\n”.
- Even though a string may contain many characters, the string itself should be thought of as a single quantity.
- Notice that this line ends with a semicolon. All statements in C end with a semicolon.

getch();

- getch() is built-in function
- By using this function at end of file, it holds your output screen until you press any character.

Constants and Variables

- **Constants:** A specific alphabetical and/or numeric value that is never changed after it is initially given a value. Storage location given a name. Referred to by the given name.

  For Ex. **PI** - 3.14159

- **Variables:** The value that can be changed. Storage locations are given names. Values of the contents for name variable locations can be changed. Referred to by variable name in the instructions.

  For Ex. ShoeCost = 56.00 and

  ShoeCost = 35.00

Variables (con't)

- **Rules for Naming and using variables**

- Name a variable according to what it represents. Create as short name as possible but one that clearly represents the variable.
- Do not use spaces in a variable name.
- Start a variable with a letter.
- Do not use dash (-) or any symbol that is used as a mathematical operator.
- Use the same variable name to represent a specific data.
- Be consistent when using upper and lower-case characters.
- Use the naming convention specified by the company where you work.

Variables (con't)

Variable Types:

Type	Memory (byte)	Range
char	1	-128 to 127
int	2	-32,768 to 32,767
long int	4	-2,147,483,648 to 2,147,483,647
float	4	10^{-38} to 10^{38} 7 digits precisions
double	8	10^{-308} to 10^{308} 7 digits precisions

Data Types

- Computers must be told the **data type** of each variable or constant. The data which computer uses are of many different types.

✓ **Numeric Data**

- Numeric data include all types of numbers. It includes subtypes:
  - **Integers are** Whole Numbers, includes positive and negative numbers.
  - **Real Numbers** are Whole Numbers and Decimal Parts.

✓ **Character Data**

- Alphanumeric data contains of al single-digit numbers, letters and special characters available to the computer.
- **String Data:** more than one character is put together.

Data Types [Cont.]

- **Logical Data**
  - Consist of two values in the data set - *True* and *False*.

- **Other Data Types**
  - **Date Data Type** is a number for the date that is the number of days from a certain date in the past.
  - **User Defined Data Types** Programmers may define their own data types.

Data Types [Cont.]

Data Type	Data Set	Examples
Numeric: INTEGER	All whole numbers	3580 −46
Numeric: REAL	All real numbers (whole + decimal)	3792.91 4739416.0 0.00246
CHARACTER (surrounded by quotation marks)	All letters, numbers, and special symbols	“A” “a” “M” “z” “k” “1” “5” “7” “8” “0” “+” “=” “(” “%” “$”
STRING (surrounded by quotation marks)	Combinations of more than one character	“Arcata” “95521” “707-444-5555”
LOGICAL	TRUE FALSE	TRUE FALSE

printf() Function

C uses formatted output. The printf function has a special formatting character (%) -- a character following this defines a certain format for a variable:

%c – characters

%d – integers

%f – floats

e.g.

printf("%c %d %f",ch,i,x);

NOTE: Make sure order of format and variable data types match up.

Format Specifiers

%c Single Character
%s String
%d Signed decimal integer
%f Floating point (decimal notation)
%e Floating point (exponential notation)
%u Unsigned decimal integer
%x Unsigned hexadecimal integer
%o Unsigned octal integer
l prefix used with %d, %u, %x, %o
to specify long integer (e.g. %ld)

Format Specifiers (con't)

Example:

```
void main (void)

{
    int event = 5;
    char heat = 'C';
    float time = 27.2589;
    printf(" The winning time in heat %c", heat);
    printf(" of event %d was %.2f", event, time);

}
```

Escape Sequences

The following list shows the common escape sequences:

- \n New line
- \t Tab
- \b Backspace
- \r Carriage return
- \’ Single quote
- \” Double quote
- \\ Backslash

scanf() Funtion

- Syntax:

  scanf (“ format specifier ”, & var_name);
- Obtains a value from the user
- `scanf()` uses standard input (usually keyboard)
- This `scanf` statement has two arguments
  - Format specifier -indicates format of the data
  - & `var_name` - location in memory to store variable
  - & is confusing in beginning – for now, just remember to include it with the variable name in `scanf` statement
- When executing the program the user responds to the scanf statement by typing in a number, then pressing the enter (return) key

scanf() Funtion (con't)

Example:

```
#include<stdio.h>
    main()
{
    int num1,num2,res;
     printf("Enter First Number: ");
     scanf("%d",&num1);

     printf("Enter Second Number: ");
     scanf("%d",&num2);

          res=num1+num2;

     printf("The sum of two numbers is: %d", res);
}
```

Operators

"Operators are words or symbols that cause a program to do something to variables."

There are may different kinds of operators, but most common of them are as follows:

- Arithmetic Operators
- Arithmetic Assignment Operators
- Unary Operators/ Increment & Decrement Operators
- Relational Operators

Operators (con't)

Arithmetic:

C operation	Arithmetic operator	Algebraic expression	C expression
Addition	+	$f + 7$	`f + 7`
Subtraction	-	$p - c$	`p - c`
Multiplication	*	bm	`b * m`
Division	/	x / y	`x / y`
Modulus	%	$r \bmod s$	`r % s`

Rules of Operator Precendence:

Operator(s)	Operation(s)	Order of evaluation (precedence)
`()`	Parentheses	Evaluated first. If the parentheses are nested, the expression in the innermost pair is evaluated first. If there are several pairs of parentheses "on the same level" (i.e., not nested), they are evaluated left to right.
`*`, `/`, or `%`	Multiplication,Division, Modulus	Evaluated second. If there are several, they are evaluated left to right.
`+` or `-`	Addition Subtraction	Evaluated last. If there are several, they are evaluated left to right.

Operators (con't)

Arithmetic Assignment:

C operation	Arithmetic operator	C expression
Equal	=	a=b
Addition Assignment	+=	a+=b (same as a=a+b)
Subtraction Assingment	-=	a-=b (same as a=a-b)
Multiplication Assignment	*=	a*=b (same as a=a*b)
Division Assignment	/=	a/=b (same as a=a/b)
Remainder Assignment	%=	a%=b (same as a=a%b)

Increment / Decrement:

C operation	Arithmetic operator	C expression
Increment	++	a++ or ++a
Decrement	--	a-- or --a

Operators (con't)

Equality and Relational:

Standard algebraic equality operator or relational operator	C equality or relational operator	Example of C condition	Meaning of C condition
Equality Operators			
=	==	x == y	x is equal to y
≠	!=	x != y	x is not equal to y
Relational Operators			
>	>	x > y	x is greater than y
<	<	x < y	x is less than y
>=	>=	x >= y	x is greater than or equal to y
<=	<=	x <= y	x is less than or equal to y

TASKS

1. Write a program that asks the user to enter two numbers, obtains them from the user and prints their sum, product, difference, quotient and remainder.
2. Write a program that reads in the radius of a circle and prints the circle's diameter, circumference and area. Use the constant value 3.14159 for π. Perform each of these calculations inside the printf statement(s) and use the conversion specifier %f.

3. Write a program that prints the following shapes with asterisks.

End of Lecture

Any Question ????