



Introduction to C Programming

Assoc. Prof. Özgür ZEYDAN

<https://ozgurzeydan.com.tr/>





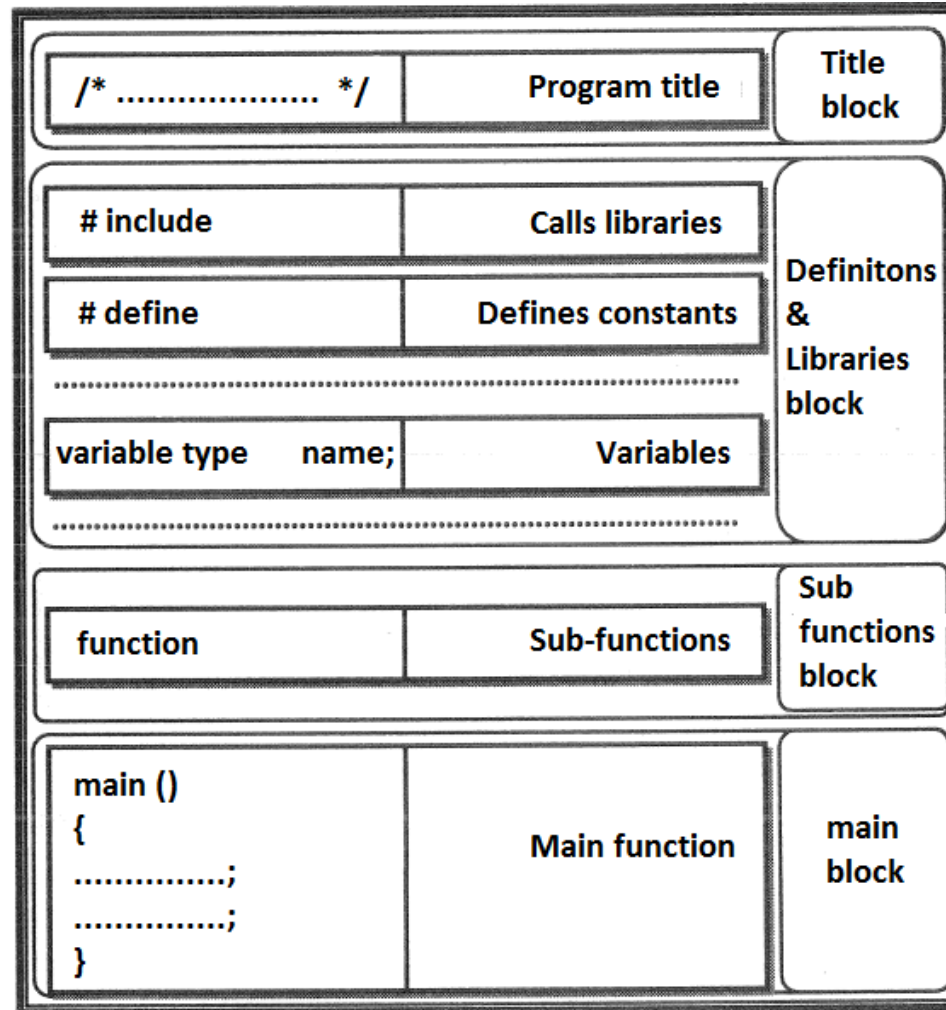
Structure of C program - input/output functions - variables

Assoc. Prof. Özgür ZEYDAN

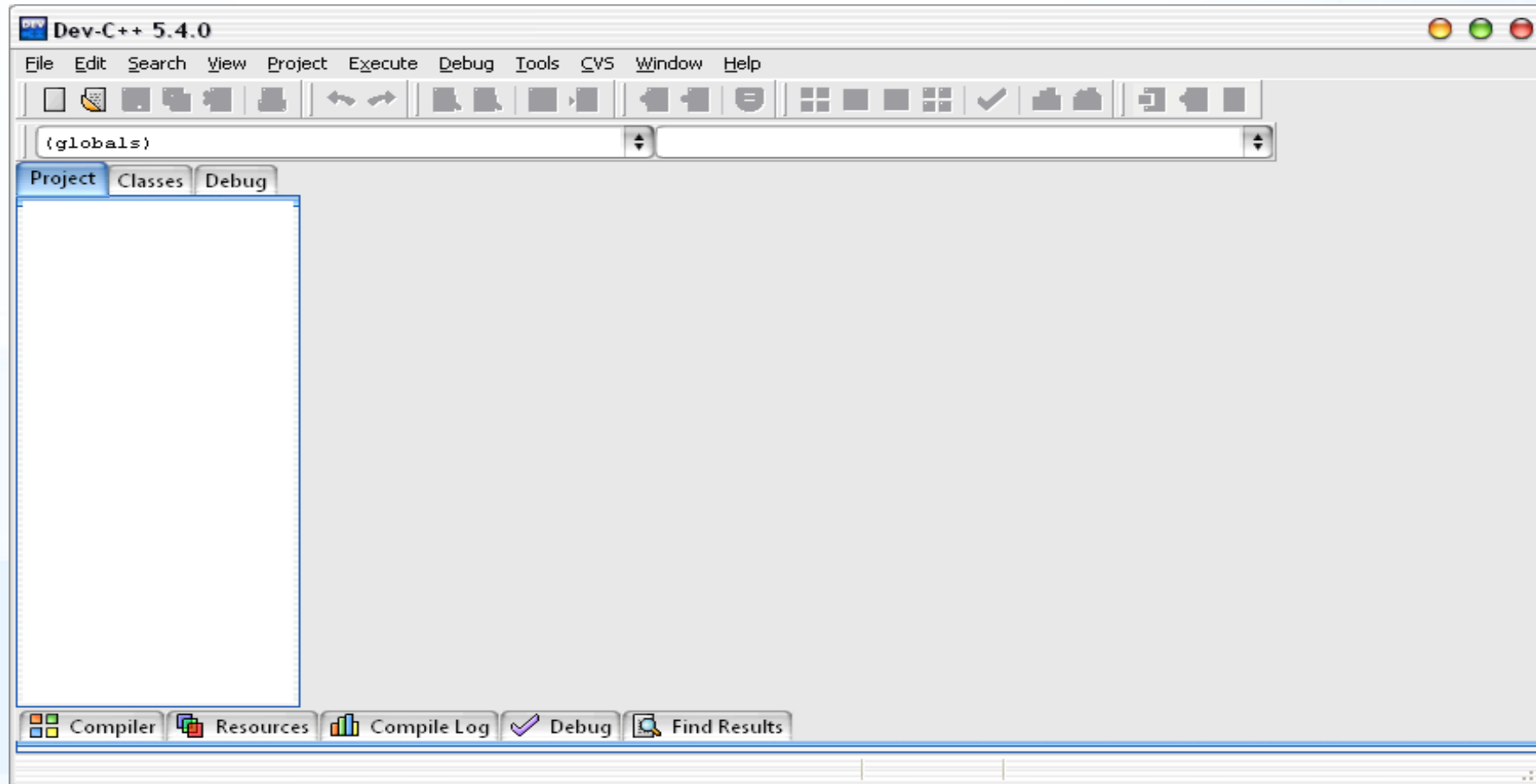
<https://ozgurzeydan.com.tr/>



Structure of a C program



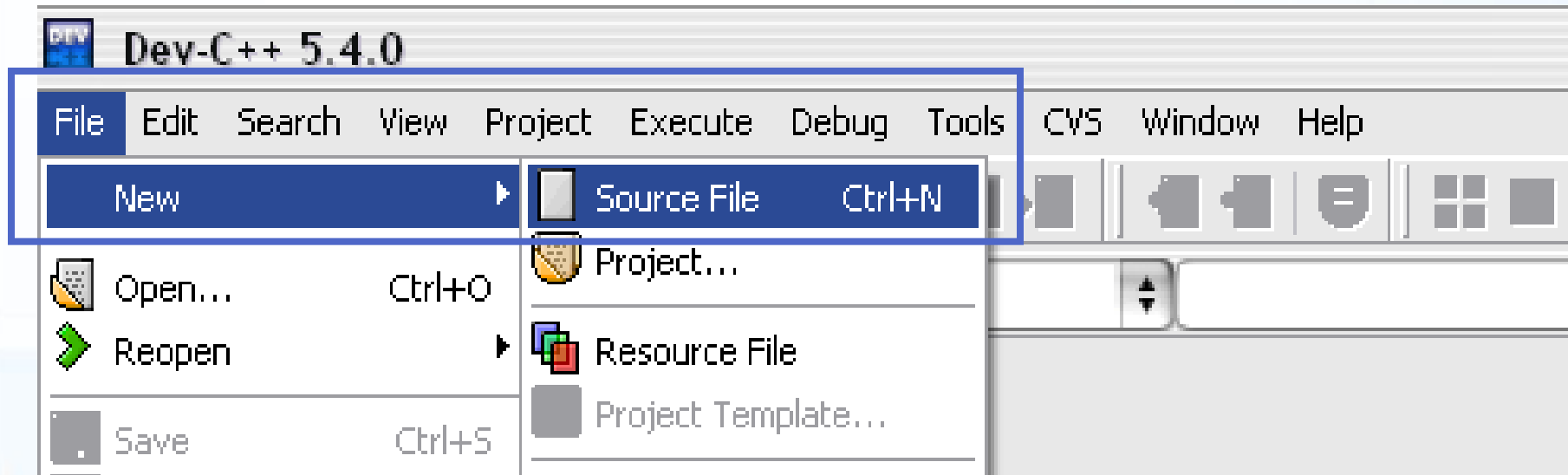
C Compiler: Dev C++



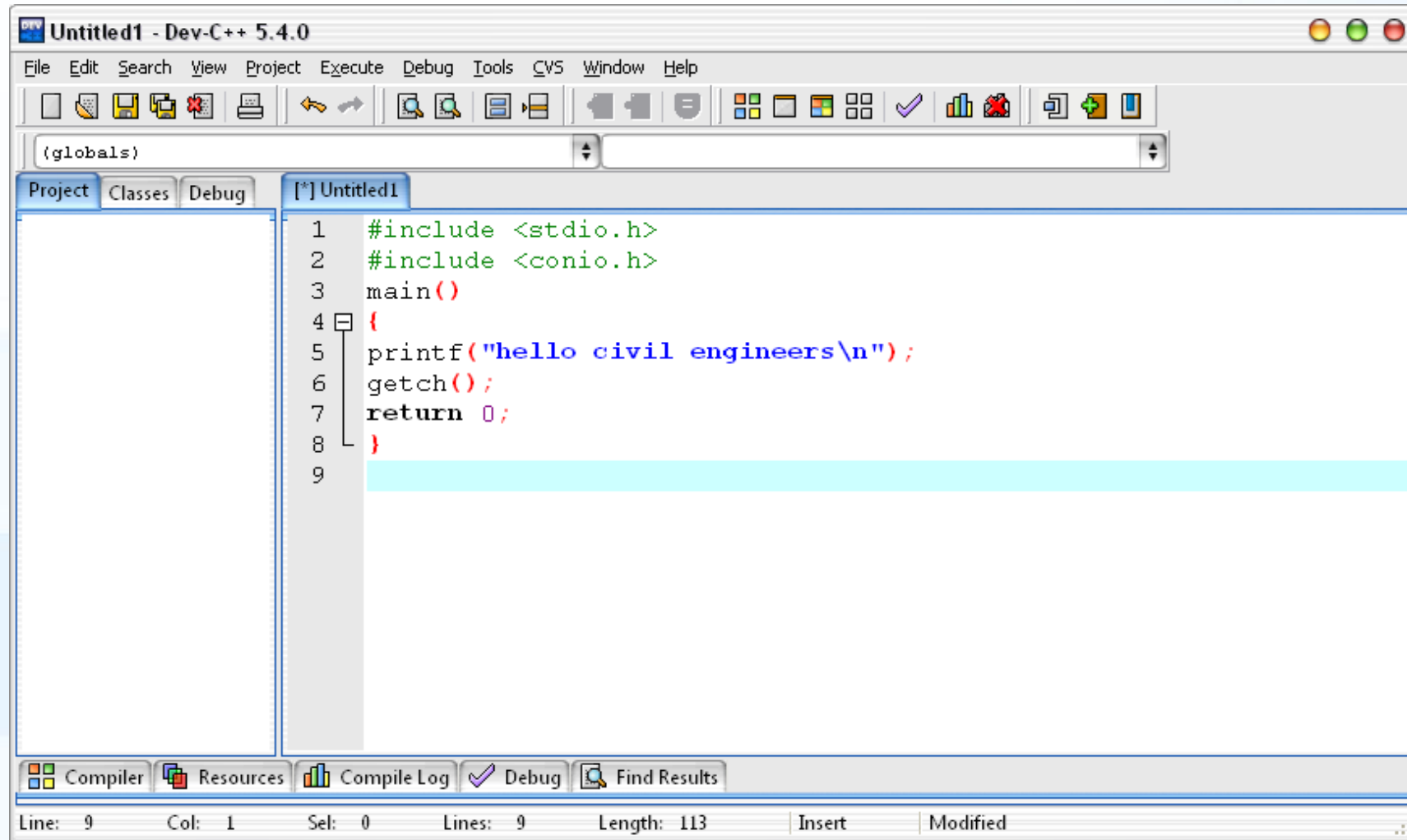
Download latest version from following link:

<http://dev-cpp.com/>

Creating a new file



Your First C Program



The screenshot shows the Dev-C++ 5.4.0 IDE with a new file named 'Untitled1'. The code is a simple C program that includes `<stdio.h>` and `<conio.h>`, defines a `main()` function, and prints the message "hello civil engineers" followed by a newline character. The program then calls `getch()` and returns 0.

```
1  #include <stdio.h>
2  #include <conio.h>
3  main()
4  {
5      printf("hello civil engineers\n");
6      getch();
7      return 0;
8  }
9
```

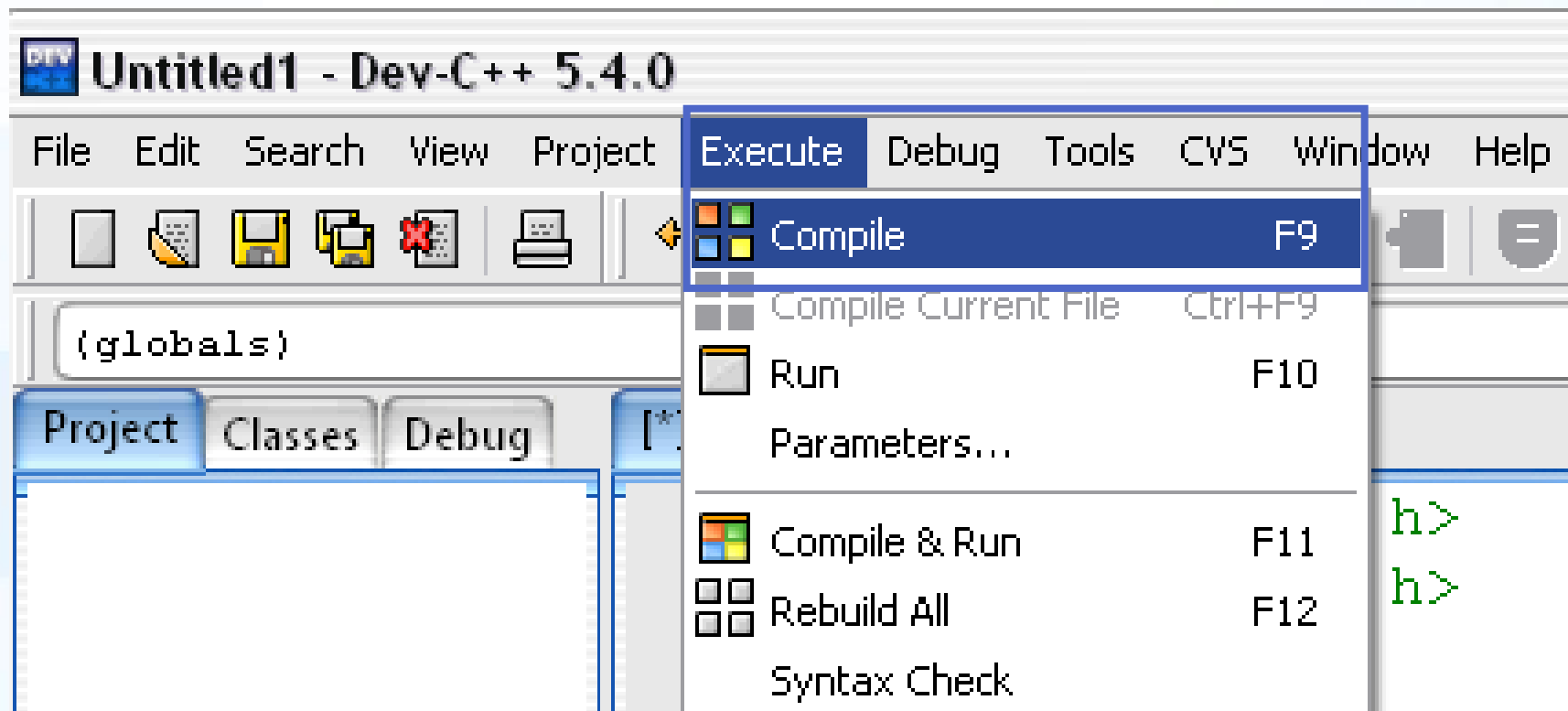
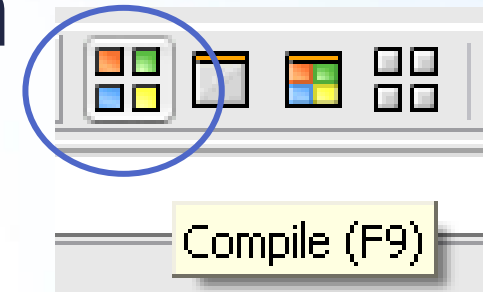
The status bar at the bottom indicates the current position is Line: 9, Col: 1, Sel: 0, with a total of 9 lines and 113 characters. The 'Insert' mode is active, and the 'Modified' flag is set.

Your First C Program

```
#include <stdio.h>
#include <conio.h>
main()
{
printf("hello civil engineers\n");
getch();
return 0;
}
```

Compiling your first program

Execute > Compile (F9)

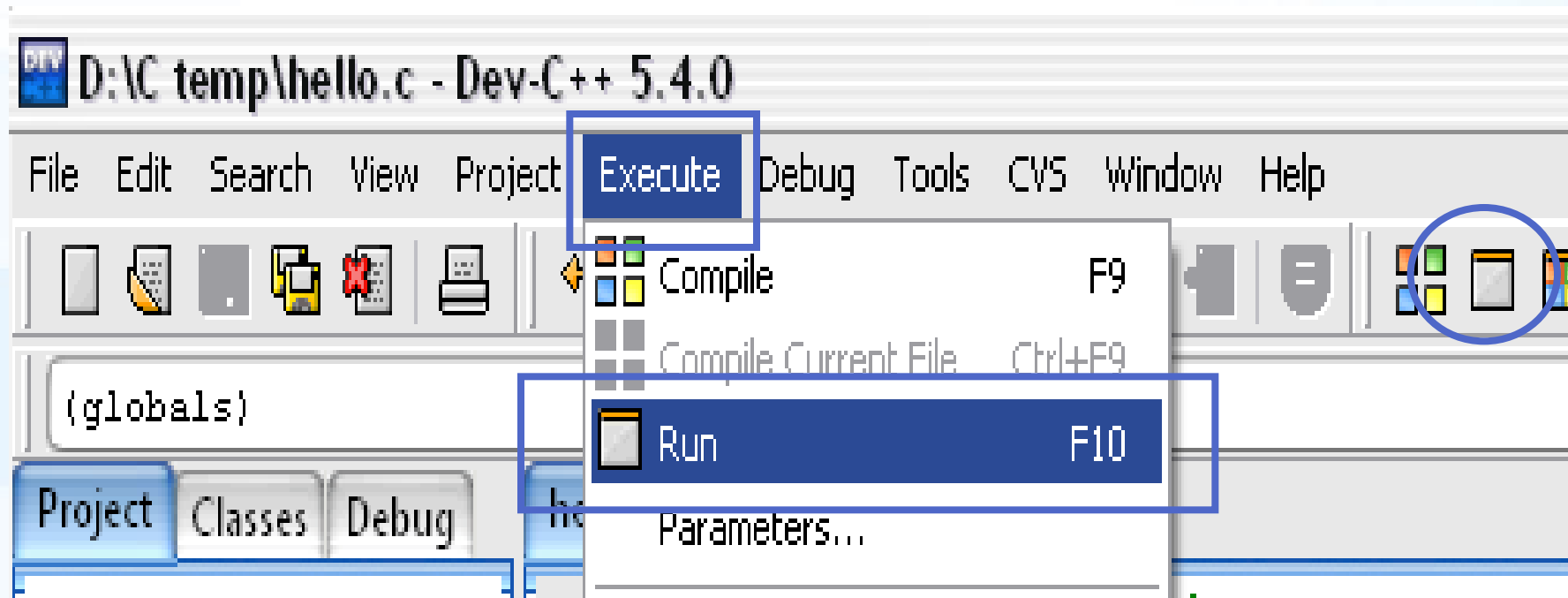


Compile Screen

Compile Progress			
Compiler:	MinGW GCC 4.7.2 32-bit		Compiler: MinGW GCC 4.7.2 32-bit Done
Status:	Done in 1.70 seconds.		
File:			
Errors:	0	Warnings: 0	
<div></div>			
<div>Close</div>			

Running your first program

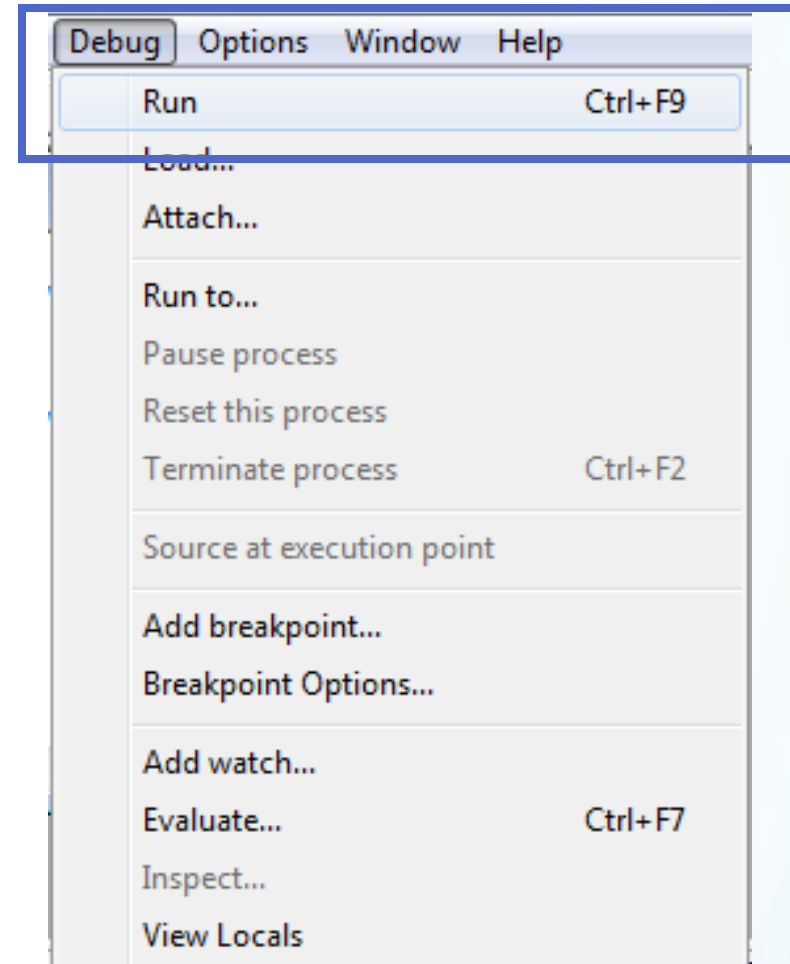
Execute > Run (F10)



Running your first program



Debug > Run (CTRL + F9)



Output Screen



A screenshot of a Windows console window. The title bar at the top reads "C:\Program Files\Dev-Cpp\ConsolePauser.exe" and includes standard minimize, maximize, and close buttons. The main area of the window is black, with the text "hello civil engineers" displayed in a white, monospaced font at the top left. A vertical scrollbar is visible on the right side of the window.

```
C:\Program Files\Dev-Cpp\ConsolePauser.exe  
hello civil engineers
```

#include<stdio.h>

- This C program starts with `#include <stdio.h>`
- This line includes the "standard I/O library" into your program.
- The standard I/O library lets you:
 - read input from the keyboard,
 - write output to the screen,
 - process text files stored on the disk, and so on.
- It is an extremely useful library.
- C has a large number of standard libraries like stdio, including string, time and math libraries.

Headers

- .h files are called header files
- `#include<stdio.h>`
 - `printf()` → output command
 - `scanf()` → input command
- `#include<conio.h>`
 - `getch()` → reads key (program waits until key pressed)

Clearing Screen

➤ Dev C++ IDE

```
#include<stdlib.h>
main()
{
    system("cls");
}
```


main() function

- The line `int main()` declares the main function.
- Every C program must have a function named main somewhere in the code.
- At run time, program execution starts at the first line of the main function.
- In C, the `{` and `}` symbols (**block delimiters**) mark the beginning and end of a block of code.

printf()

- The `printf` statement in C allows you to send output to standard out (for us, the screen).
- The portion in quotes is called the format string and describes how the data is to be formatted when printed.
- The format string can contain:
 - string literals such as "Hello civil engineers"
 - symbols for carriage returns (`\n`)
 - operators as placeholders for variables.

return 0;

- The `return 0;` line causes the function to return an error code of 0 (no error) to the shell that started execution.
- `;` symbol is statement terminator.

Modifying first program

```
/* Programmer : Özgür ZEYDAN */  
#include <stdio.h>  
#include <conio.h>  
main()  
{  
printf("hello civil engineers\n");  
getch();  
printf("\nhello civil engineers");  
getch();  
return 0;  
}
```

Comments

- `/* Programmer : Özgür ZEYDAN */`
- Comments can be inserted into C programs by bracketing text with the `/*` and `*/` delimiters.
- Comments are useful for a variety of reasons.
- Primarily they serve as internal documentation for program structure and functionality.
- Best programmers comment as they write the code, not after the fact.

Variables

- As a programmer, you will frequently want your program to "remember" a value. For example, if your program requests a value from the user, or if it calculates a value, you will want to remember it somewhere so you can use it later. The way your program remembers things is by using variables. For example:

```
int b;
```

- This line says, "I want to create a space called b that is able to hold one integer value." A variable has a name (in this case, b) and a type (in this case, int, an integer). You can store a value in b by saying something like:

```
b = 5;
```

- You can use the value in b by saying something like:

```
printf("%d", b);
```

Variable Names

- Variable names starts with a letter or underscore “_” and followed by either letters or digits or underscore “_”.
- Variable names can not start with digits or special characters.
- C is a case sensitive language. It means variables SUM is different than Sum or sum.
- You can not use **Reserved Words** as variable names.

Reserved Words

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

Example - 1

```
#include <stdio.h>
#include <conio.h>
int main() {
    int a, b, c;
    a = 5;
    b = 7;
    c = a + b;
    printf("%d + %d = %d\n", a, b, c);
    getch();
    return 0; }
```

Explanation of Example - 1

- The line `int a, b, c;` declares three integer variables named `a`, `b` and `c`. Integer variables hold whole numbers.
- The next line initializes the variable named `a` to the value 5.
- The next line sets `b` to 7.
- The next line adds `a` and `b` and "assigns" the result to `c`.
- The `printf` statement then prints the line "5 + 7 = 12."

Placeholders

- The `%d` placeholders in the `printf` statement act as placeholders for values.
- There are three `%d` placeholders, and at the end of the `printf` line there are the three variable names: **a**, **b** and **c**.
- C matches up the first `%d` with **a** and substitutes 5 there. It matches the second `%d` with **b** and substitutes 7. It matches the third `%d` with **c** and substitutes 12.
- Then it prints the completed line to the screen: $5 + 7 = 12$.

Example - 2

- In previous example, user can not change the values of a and b.
- The program always use $a = 5$ and $b = 7$.
- Let us write a better program:
 - Programs asks user to write first number and then initialize the value of a to that number.
 - Programs asks user to write first number and then initialize the value of b to that number.
 - Then program calculates c as $c = a + b$.
 - Finally, program displays the output.

Example – 2 (Pseudocode)

Start

Use Variables: a, b and c

Display "write a number"

Read a

Display "write a number"

Read b

Calculate $c = a + b$

Display " $c = a + b$ "

Stop

Example – 2 (C code)

```
#include <stdio.h>
#include <conio.h>
int main() {
    int a, b, c;
    printf("Enter the first value:");
    scanf("%d", &a);
    printf("Enter the second value:");
    scanf("%d", &b);
    c = a + b;
    printf("%d + %d = %d\n", a, b, c);
    getch();
    return 0; }
```


Example – 2

understanding
the execution
of a C Program

©2004 HowStuffWorks

<http://www.howstuffworks.com/c.htm>

Scanf()

- The **scanf** function allows you to accept input from standard **in**, which for us is generally the keyboard.
- Note that scanf uses the same sort of format string as printf . Also note the **&** in front of a and b.
- This is the **address operator** in C: It returns the address of the variable.
- You must use the & operator in scanf on any variable of type char, int, or float, as well as structure types.
- If you leave out the & operator, you will get an error when you run the program.

Placeholders

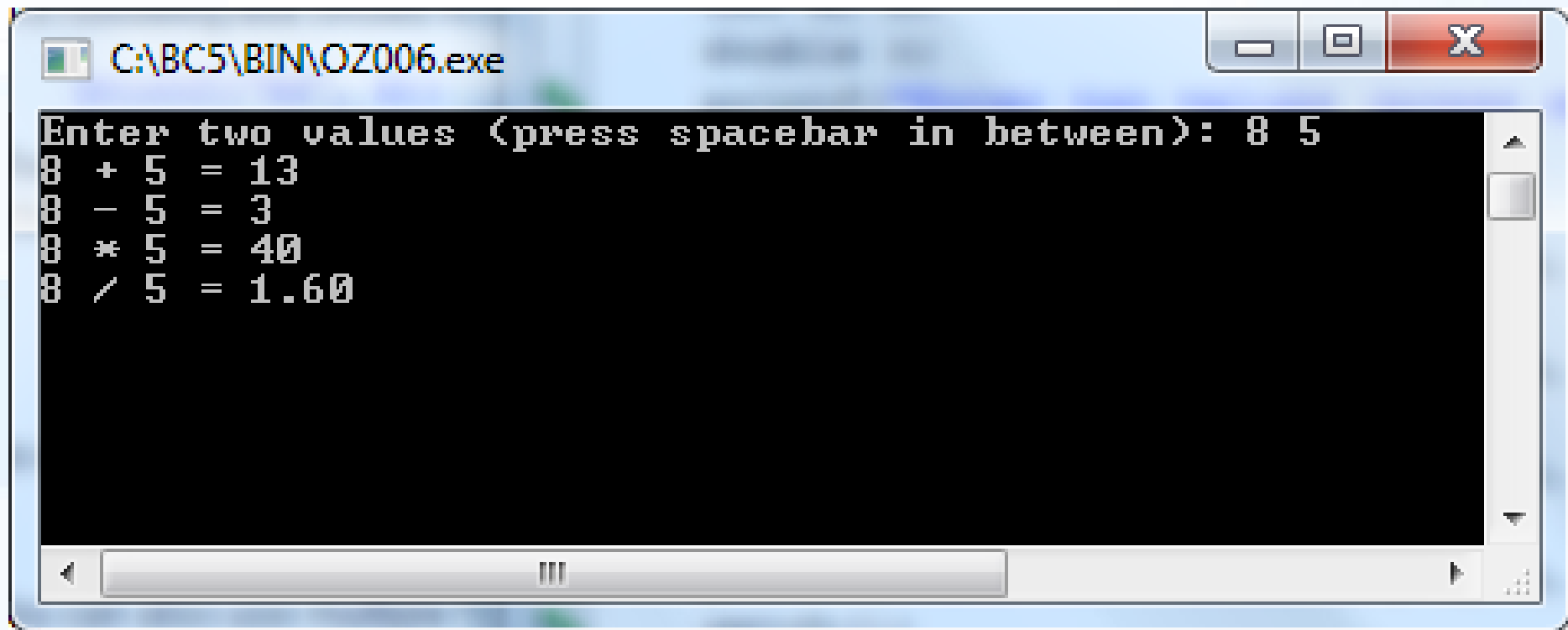
- You can **print all the normal C types with `printf`** by using different placeholders:
 - **int** (integer values) uses `%d`
 - **float** (floating point values) uses `%f`
 - **double** (decimal values) uses `%f`
 - **char** (single character values) uses `%c`
 - **character strings** (arrays of characters) use `%s`
 - **double** (decimal values) uses `%lf` in `scanf`

Example – 2 (Better C code)

```
#include <stdio.h>
#include <conio.h>
int main() {
    int a, b, c;
    printf("Enter two values (press spacebar in between): ");
    scanf("%d %d", &a, &b);
    c = a + b;
    printf("%d + %d = %d\n", a, b, c);
    getch();
    return 0;
}
```

Example - 3

Write a C program that gives following output screen.
Make all necessary changes.



The screenshot shows a Windows command prompt window titled "C:\BC5\BIN\OZ006.exe". The window has a black background with white text. The prompt "Enter two values (press spacebar in between):" is followed by the input "8 5". Below this, four arithmetic operations are displayed: addition, subtraction, multiplication, and division, each with its result. The window includes standard Windows window controls (minimize, maximize, close) and a scrollbar on the right side.

```
C:\BC5\BIN\OZ006.exe
Enter two values (press spacebar in between): 8 5
8 + 5 = 13
8 - 5 = 3
8 * 5 = 40
8 / 5 = 1.60
```

Example - 3

```
#include <stdio.h>
#include <conio.h>
int main() {
    int a, b;    double c;
    printf("Enter two values (press spacebar in between): ");
    scanf("%d %d", &a, &b);
    c = a + b;    printf("%d + %d = %0.0f \n", a, b, c);
    c = a - b;    printf("%d - %d = %0.0f \n", a, b, c);
    c = a * b;    printf("%d * %d = %0.0f \n", a, b, c);
    c = (double) a / b;    printf("%d / %d = %4.2f \n", a, b, c);
    getch();
    return 0;
}
```

Formatting Outputs

- Integers: **%nd**
- n: minimum width on the output where value is printed on the right.

a=15;

printf("%3d",a); → _ 1 5

printf("%5d",a); → _ _ _ 1 5

b=5682;

printf("%2d",b); → 5 6 8 2

printf("%6d",b); → _ _ 5 6 8 2

Formatting Outputs

- Doubles: `%n.mf`
- n: minimum total width of the output
- m: exact width for the output of part after decimal point

```
c=15.648;
```

```
printf("%9.4f",c);    →    _ _ 1 5 . 6 4 8 0
```

```
printf("%4.2f",c);    →    1 5 . 65
```

```
- _ _ _ _
```

Example - 4

```
#include<stdio.h>
#include<conio.h>
main()
{
char me[20];
printf("Please write your name: ");
scanf("%s",&me);
printf("Nice to meet you, %s",me);
getch();
return(0);
}
```

Arithmetic Operators and Precedence

Operation	Arithmetic operator	C programming
Addition	+	$a + 5$
Subtraction	-	$b - 14$
Multiplication	*	$c * d$
Division	/	j / i
Modulus	%	$x \% y$

Operator	Precedence (evaluation order)
()	First
* / %	Second
+ -	Third

Example - 5

- Remember from previous week. We have written an algorithm and drawn a flowchart for **celcius to fahrenheit conversion program**.
- Now, write a C code of that program that converts F value, which is given by user, to C value.

Example - 5

```
#include <stdio.h>
#include <conio.h>
main() {
float c,f;           /* c: celcius, f: fahrenheit */
printf("Fahrenheit to celcius conversion program.\n");
printf("Write F value : ");
scanf("%f",&f);
c=(f-32)*5/9;
printf("\nC value is %4.1f",c);
getch();
return(0); }
```

Homework

1. Write a C program that converts centimeters to inches and feet.
2. Write a C program that converts gallons to liters and cubic meters.

Example - 6

- Write a C program that calculates area and circumference of a circle whose radius is given by the user.
- You can define pi by writing this line immediately after header lines:

```
#define PI 3.1415962
```

- $A = \pi * r^2$
- $C = 2 * \pi * r$

Example - 6


```
#include<stdio.h> #include<conio.h>
#define PI 3.1415962
main()      {
int r;  double a,c;
printf("This program calculates area and circumference of a circle.");
printf("Radius : ");  scanf("%d",&r);
a=PI*r*r;    c=2*PI*r;
printf("\nArea: %f, Circumference: %f",a,c);
getch();
return(0); }
```


Example - 7

- Write a program that asks user to write radius (r) and height (h) of a cylinder and then calculates:
 - Volume ($V = \pi r^2 h$)
 - Side area ($2 \pi r h$)
 - Total area ($2 \pi r^2 + 2 \pi r h$)

Example – 7

```
#include<stdio.h> #include<conio.h>
#define PI 3.1415962
main()      {
int r,h;      double V,SA,TA;
printf("Clynder calculator.\n");
printf("Write radius and height: "); scanf("%d %d",&r,&h);
V=PI*r*r*h; SA=2*PI*r*h;      TA=2*PI*r*(r+h);
printf("Volume: %4.2f, Side area: %4.2f, Total area: %4.2f",V,SA,TA);
getch();
return(0); }
```



Selection structures - If, If/Else, Switch/Case

Assoc. Prof. Özgür ZEYDAN

<https://ozgurzeydan.com.tr/>



C Data Types

Data Type	Meaning	Storage Space	Format	Range of Values
char	A character	1 byte	%c	ASCII character set
int	An integer	2 bytes	%d	-32768 to +32767
float	A single precision floating point number	4 bytes	%f	-3.4×10^{38} to $+3.4 \times 10^{38}$
double	A double precision floating point number	8 bytes	%lf	-1.7×10^{308} to $+1.7 \times 10^{308}$
void	valueless or empty	0 byte	-	-

float: 7 significant digits

double: 15-16 significant digits

The `if/else` Selection Structure

- **if** : Only performs an action if the condition is **true**.
- **if/else** : A different action when condition is **true** than when condition is **false**
- Relational and logical operators: `<`, `>`, `<=`, `>=`, `==`, `!=`, `&&`, `||`
- Psuedocode:
 - If student's average grade is smaller than 60
 Print "Failed"
 - else
 Print "Success"
- C code:

```
if ( avg < 60 )  
    printf("Failed");  
else  
    printf("Success");
```

Relational Operators

Operator	Meaning
==	Equal to
!=	Not equal to
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to

The `if/else` Selection Structure

Compound statement:

- Set of statements within a pair of braces
- Example:

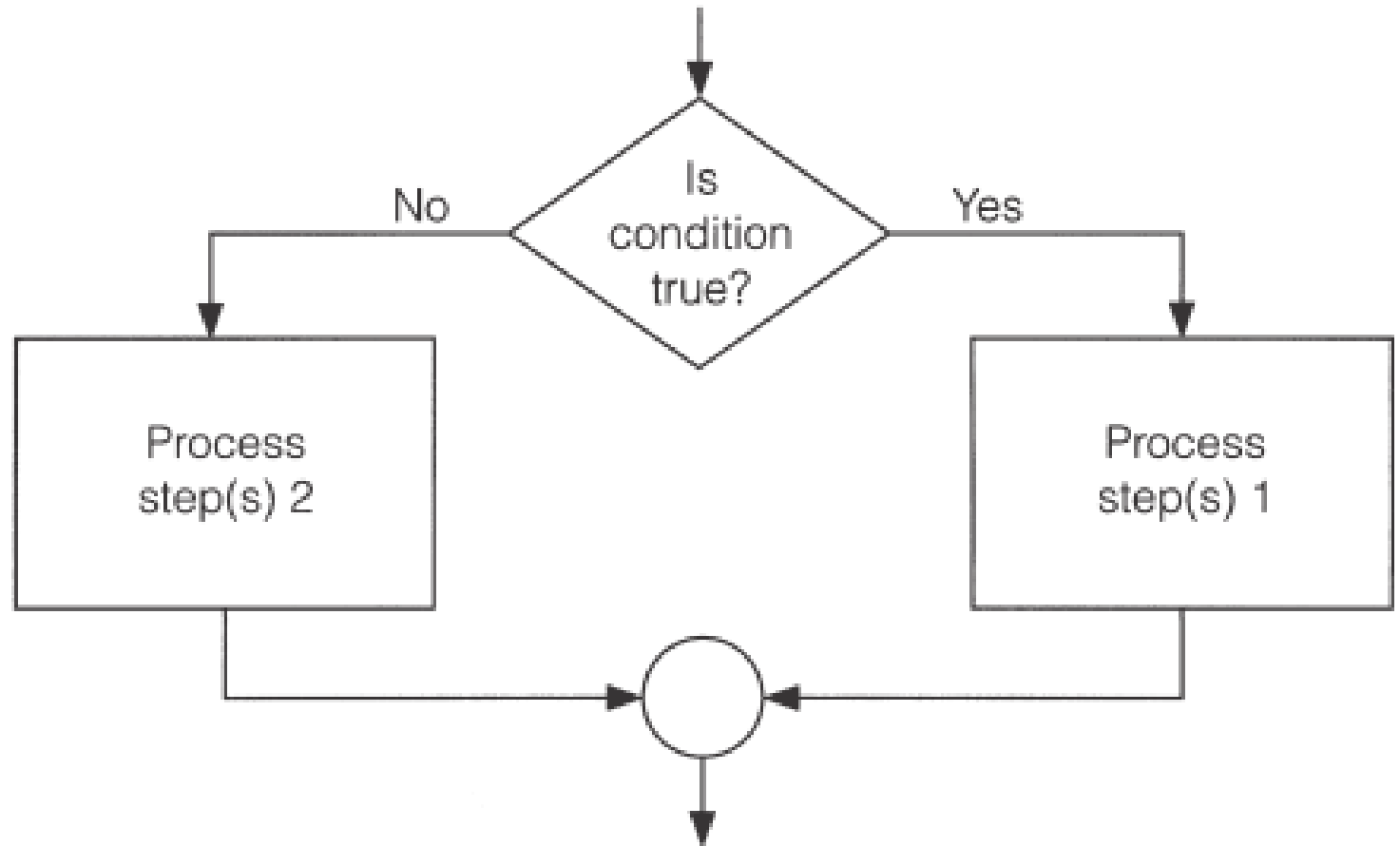
```
if ( avg < 60 )  
    printf("Failed");  
else {  
    printf("Success \n");  
    printf("You must take this course  
again.\n"); }
```

- Without the braces,

```
printf( "You must take this course again.\n" );  
would be automatically executed
```

Pseudocode and Flowchart for a Decision Structure

```
If condition is true Then
    Process step(s) 1
Else
    Process step(s) 2
End If
```



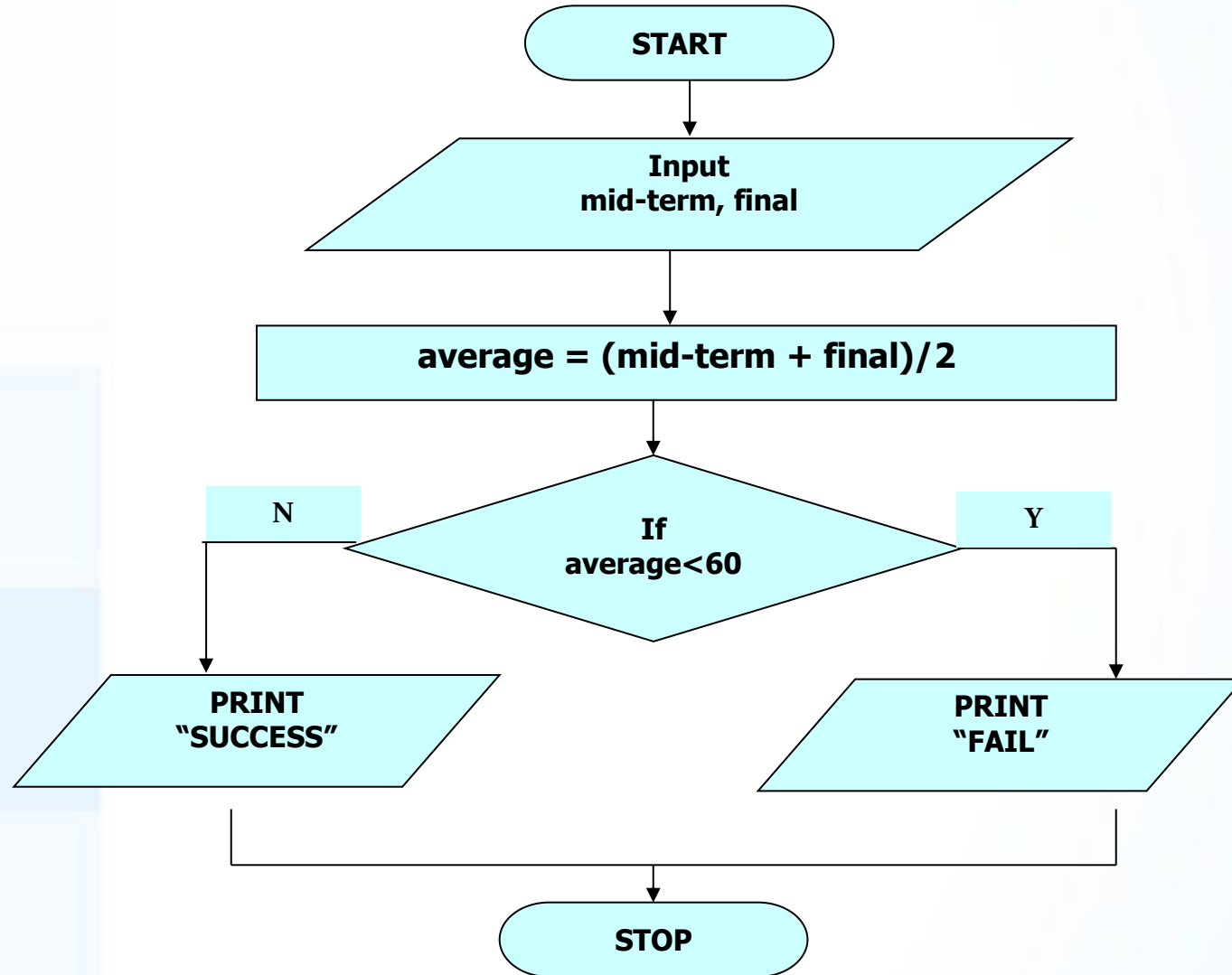
Example - 1

- Write an algorithm to determine a student's average grade and indicate whether he is successful or not.
- The **average** grade is calculated as the average of **mid-term** and **final** grades.
- Student will be successful if his average grade is grater or equals to 60.

Detailed Algorithm

- 1. Step: Input **mid-term** and **final**
- 2. Step: **average** = (**mid-term** + **final**)/2
- 3. Step: if (**average** < 60) then
 Print "FAIL"
 else
 Print "SUCCESS"
 endif

Flowchart



Example – 1: C Codes

```
#include<stdio.h>
#include<conio.h>
main() {
int m,f;      /* m: mid-term grade , f: final grade*/
double avg;   /* avg: average grade*/
printf("Write mid-term and final grades: ");
scanf("%d %d",&m,&f);
avg=(m+f)/2;
if (avg<60) printf("Fail");
else printf("Success");
getch();     return(0); }
```

Equality (==) and Assignment (=) Operators

- Do not confuse Equality (==) with Assignment (=) Operators in **if** statements.

```
if ( avg == 100 )  
    printf( "You get max average" );
```

- Correct usage

```
if ( avg = 100 )  
    printf( "You get max average" );
```

- This *sets* **avg** to 100
- Since 100 is nonzero, so expression is **true**, and this line is executed no matter what the value of **avg** was
- Logic error!!!

Example - 2

- Write a program which asks user to write an integer value and then computer decides whether this number is
 - odd number
- or
 - even number.

Example - 2

```
#include <stdio.h>
#include <conio.h>
main()
{
int i;
printf("Write an integer: ");
scanf("%d",&i);
    if (i%2==1) printf("Odd number");
    else printf("Even number");
getch();
return 0;    }
```

Example - 3

- Write a program that asks user to write his/her weight and height.
- Then calculates mass body index (mbi).
- After that program displays a message with respect to the value of mbi.
 - if $mbi < 18.5$ → Underweight
 - if $mbi < 25.0$ → Normal weight
 - if $mbi < 30.0$ → Slightly overweight
 - if $mbi < 35.0$ → Overweight
 - otherwise → Obesite warning

Example - 3

```
/* Body Mass Index Calculator written by Özgür ZEYDAN */
#include <stdio.h>      #include <conio.h>
main()                  {
double w,h, mbi; /* w: weight  h: height  mbi: body mass index
*/
printf("*** Body Mass Index Calculator by Özgür ZEYDAN ***
\n");
printf("Write your weight in kilograms :");
scanf("%lf",&w);
printf("Write your height in meters :");
scanf("%lf",&h);
```

Example - 3

```
mbi=w/(h*h);  
    if (mbi<18.5) printf("Underweight");  
    else if (mbi<25.0) printf("Normal weight");  
    else if (mbi<30.0) printf("Slightly overweight");  
    else if (mbi<35.0) printf("Overweight");  
    else printf("Obesite warning");  
getch();  
return 0;  
}
```

Math Library Functions

- Math library functions
 - perform common mathematical calculations
 - `#include <math.h>`
- Format for calling functions

`FunctionName (argument) ;`

 - `printf ("%4.2f", sqrt(81.0)) ;`
 - Calls function `sqrt`, which returns the square root of its argument
 - All math functions return data type `double`
 - Arguments may be constants, variables, or expressions

Example - 4

- Write a C program in order to solve quadratic equation.
 - Program ask values of "a" , "b" and "c"
 - Calculates delta
 - If $\text{delta} > 0$ then calculates x_1 , x_2 display results
 - If $\text{delta} = 0$ then calculates x_1 display results
 - If $\text{delta} < 0$ then display "no real roots" message

Example - 4

```
/* Quadratic Equation Solver written by Özgür ZEYDAN */
#include <stdio.h>
#include <conio.h>
#include <math.h>
main()
{
    int a,b,c;                /*  $a*x^2 + b*x + c = 0$  */
    float d,x1,x2;           /* d: delta */
    printf(" *** Quadratic Equation Solver by Özgür ZEYDAN ***\n");
    printf("Write a b c values: ");
    scanf("%d %d %d",&a,&b,&c);
```

Example - 4

```
d=b*b-4*a*c;
if (d>0)    {
    x1=(-b-sqrt(d))/(2*a);
    x2=(-b+sqrt(d))/(2*a);
    printf("X1 = %4.2f and X2 = %4.2f",x1,x2);    }
else if (d==0)    {
    x1=(-b)/(2*a);
    printf("X1 = X2 = %4.2f",x1);    }
else printf("No real root exists");
getch();
return 0;    }
```

Example – 4 with less coding

```
/* Quadratic Equation Solver written by Özgür ZEYDAN */  
#include <stdio.h>  
#include <conio.h>  
#include <math.h>  
main()      {  
int a,b,c;      /*  $a*x^2 + b*x + c = 0$  */  
float d;        /* d: delta */  
printf(" *** Quadratic Equation Solver by Özgür ZEYDAN ***  
\n");  
printf("Write a b c values: ");  
scanf("%d %d %d",&a,&b,&c);
```

Example – 4 with less coding

```
d=b*b-4*a*c;  
if (d>=0)  
printf("X1 = %4.2f and X2 = %4.2f", (-b-  
sqrt(d))/(2*a), (-b+sqrt(d))/(2*a));  
else printf("No real root exists");  
getch();  
return 0;  
}
```


Defining Macros

- `#define cube(x) x*x*x`
- If you use,
- `y = cube(a) + b;`
- This statement will be replaced by,
- `y = a*a*a + b;`

Some Useful Macros

➤ Mean:

➤ `#define mean(x,y) ((x)+(y))/2)`

➤ Maximum number:

➤ `#define max(x,y) ((x)>(y) ? (x) : (y))`

➤ Minimum number:

➤ `#define min(x,y) ((x)<(y) ? (x) : (y))`

➤ Logical expression ? True action : False Action

Example – 5

- Write a C program that asks user to write two integers.
- Then, program displays bigger number and smaller number.

Example – 5

```
#include <stdio.h>
#include <conio.h>
#define max(x,y) ((x)>(y) ? (x) : (y))
#define min(x,y) ((x)<(y) ? (x) : (y))
main()    {
    int i,j;
    printf("Write two integers (space in between): ");
    scanf("%d %d",&i,&j);
    printf("Bigger number is %d\n",max(i,j));
    printf("Smaller number is %d",min(i,j));
    getch();
    return 0;    }
```

The `switch` Multiple-Selection Structure

➤ `switch`

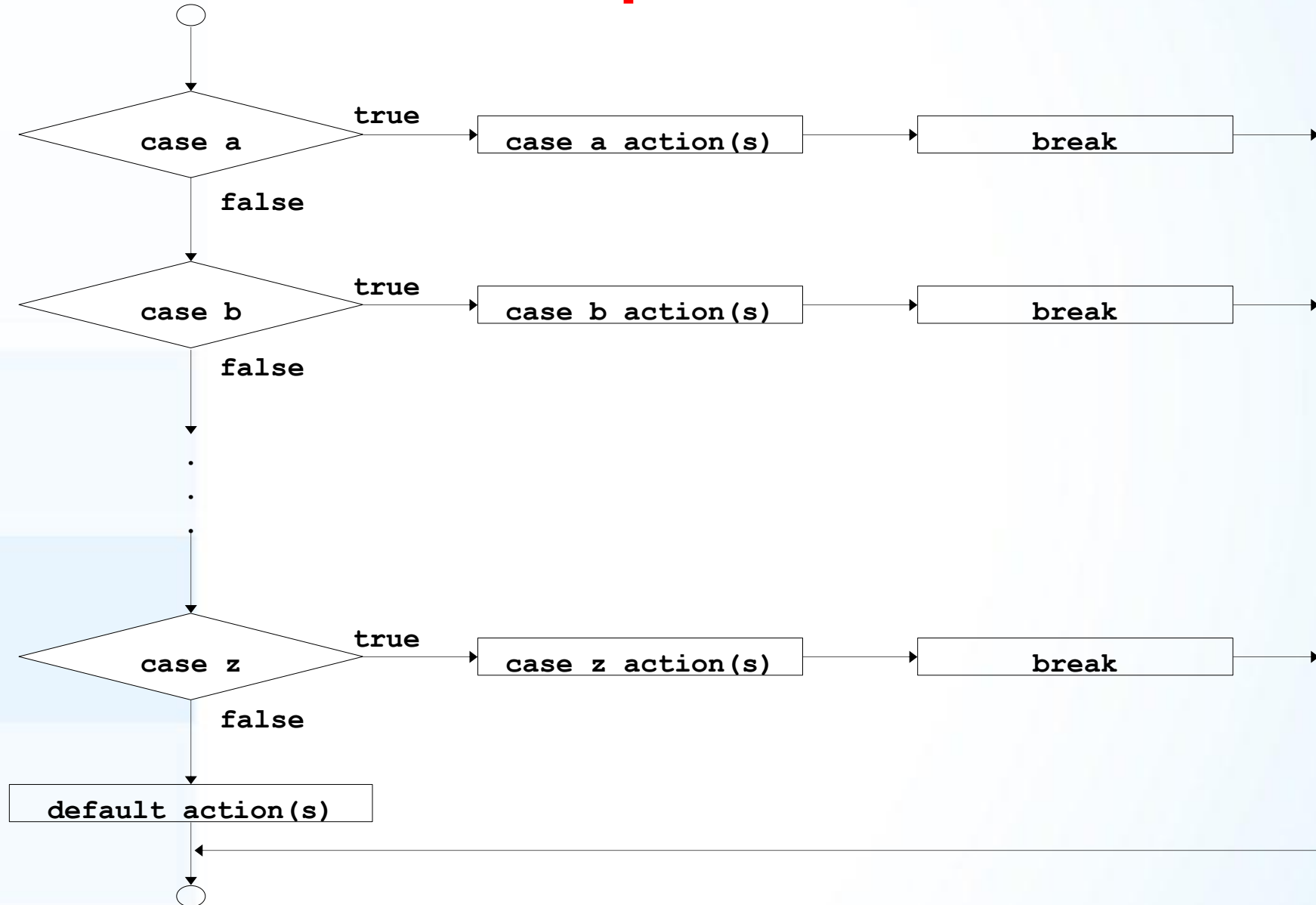
- Useful when a variable or expression is tested for all the values it can assume and different actions are taken.

➤ Format : Series of `case` labels and an optional `default case`

```
switch ( value ){  
    case '1':  
        actions; break;  
    case '2':  
        actions;          break;  
    default:  
        actions; break;  
}
```

- `break;` causes exit from structure

The switch Multiple-Selection Structure



Example – 6

- Rewrite odd – even number program by using **switch** function.

Example – 6

```
#include <stdio.h>
#include <conio.h>
main()    {
int i,j;
printf("Write an integer: ");
scanf("%d",&i);
j=i%2;
switch (j) {
    case 0: printf("Even number"); break;
    case 1: printf("Odd number");      }
getch();    return 0;    }
```


Example – 7

- Write a Calculator program with switch function.
- Program will ask operator (+ - * /) and make calculations.
- You may use ascii values of operators like:

```
char c1,c2;  /* c1: choice  
              c2: ascii code of c1 */  
  
scanf ("%s",&c1) ;  
c2=(int) c1;
```

42	2A	052	*	*
43	2B	053	+	+
44	2C	054	,	,
45	2D	055	-	-
46	2E	056	.	.
47	2F	057	/	/

ASCII Table


Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040	 	Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	\$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	((72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051))	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[[123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174	|	
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135]]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177		DEL

Example – 7

```
/* Simple Calculator by Özgür ZEYDAN */  
#include <stdio.h>      #include <conio.h>  
main()    {  
float n1,n2;      /* n1: number 1 , n2: number 2 */  
char c1,c2;      /* c1: choice , c2: ascii code of c1 */  
printf("Simple Calculator by Özgür ZEYDAN");  
printf("Write two numbers: ");  
scanf("%f %f",&n1,&n2);  
printf("Select choice (+ - * /) ");  
scanf("%s",&c1);  
c2=(int)c1;
```

Example – 7

```
switch (c2) {  
    case 43: printf("%0.0f + %0.0f = %0.0f",n1,n2,n1+n2); break;  
    case 45: printf("%0.0f - %0.0f = %0.0f",n1,n2,n1-n2); break;  
    case 42: printf("%0.0f * %0.0f = %0.0f",n1,n2,n1*n2); break;  
    case 47: if (n2==0) printf("Division by zero error!");  
             else printf("%4.2f / %4.2f = %5.3f",n1,n2,n1/n2); break;  
    default: printf("Wrong operator!.."); break;    }  
getch();    return 0;    }
```



Logical Operators - math.h library - for loop

Assoc. Prof. Özgür ZEYDAN

<https://ozgurzeydan.com.tr/>



Logical Opeartors

- You use C logical operators to connect expressions and / or variables to form compound conditions.
- The C logical expression return an integer (int).
- The result has value 1 if the expression is evaluated to true otherwise it return 0.
- C uses the following symbols for the boolean operations AND, OR ,and NOT.

Operator	Meaning
&&	logical AND
	logical OR
!	logical NOT

Logical Operators

Operator	Symbol	Usage	Operation
Logical AND	&&	exp1 && exp2	If both exps true, gives true else false
Logical OR		exp1 exp2	If both exps false give false, else true
Logical NOT	!	! exp3	Opposite of exp (true gives false or vice versa)

Logical Opeartors

➤ Examples:

```
if ((final>=50) && (average>=60))  
    printf("Success");
```

```
if ((final<35) || (average<35))  
    printf("Fail");
```


<math.h> functions

Function	Description
sqrt (x)	Square root of x
cbrt (x)	Cubic root of x
exp (x)	Exponential function e^x
log (x)	Natural logarithm of x (base: e)
log10 (x)	logarithm of x (base: 10)
pow (x,y)	Power function x^y
fabs (x)	Absolute value of x
ceil (x)	returns the nearest integer not less than x
floor (x)	returns the nearest integer not greater than x

x, y : double

<math.h> functions

Function	Description
$\sin(x)$	sine of x
$\cos(x)$	cosine of x
$\tan(x)$	tangent of x
$\operatorname{asin}(x)$	arc sine of x
$\operatorname{acos}(x)$	arc cosine of x
$\operatorname{atan}(x)$	arc tangent of x
$\sinh(x)$	hyperbolic sine of x
$\cosh(x)$	hyperbolic cosine of x
$\tanh(x)$	hyperbolic tangent of x
$\operatorname{asinh}(x)$	hyperbolic arc sine of x
$\operatorname{acosh}(x)$	hyperbolic arc cosine of x
$\operatorname{atanh}(x)$	hyperbolic arc tangent of x

x : double

The Essentials of Repetition

➤ Loop

- Group of instructions computer executes repeatedly while some condition remains true

➤ Counter-controlled repetition

- Definite repetition - know how many times loop will execute
- Control variable used to count repetitions

➤ Sentinel-controlled repetition

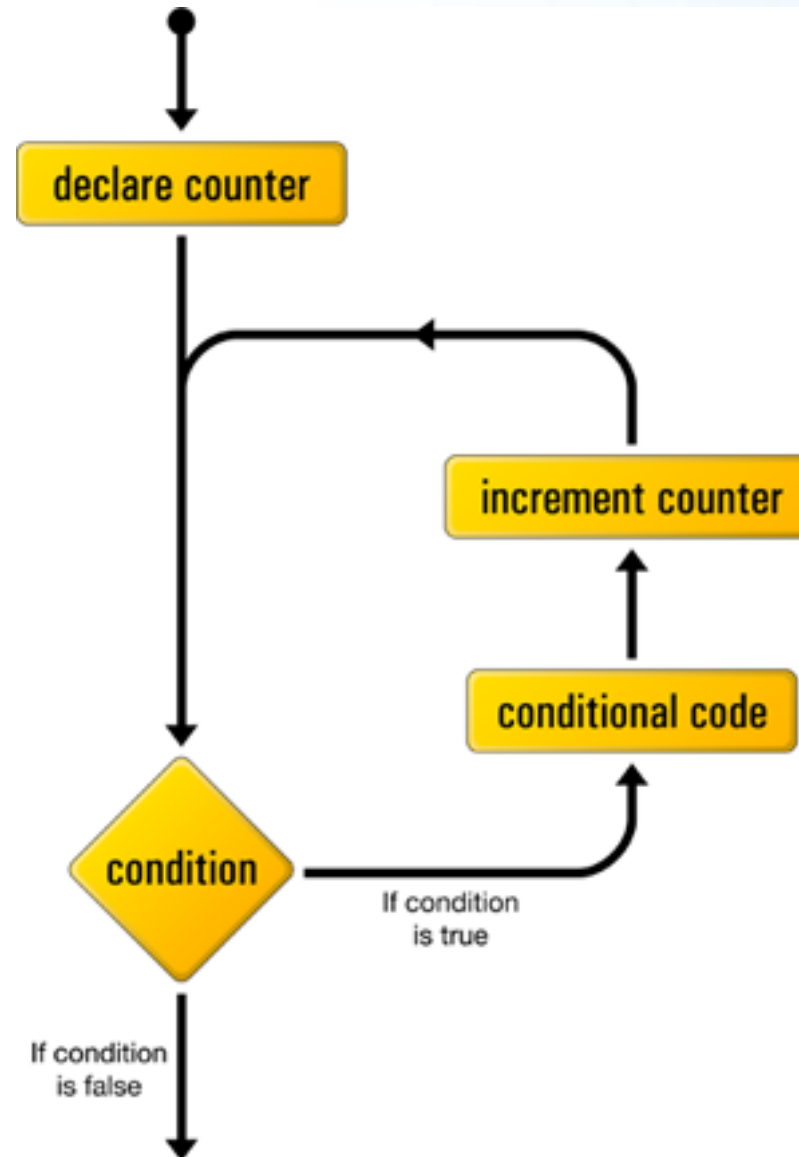
- Indefinite repetition
- Used when number of repetitions not known
- Sentinel value indicates "end of data"

The Essentials of Repetition

- Counter-controlled repetition requires
 - *name* of a control variable (or loop counter).
 - *initial value* of the control variable.
 - condition that tests for the *final value* of the control variable (i.e., whether looping should continue).
 - *increment* (or *decrement*) by which the control variable is modified each time through the loop.

The **for** Loop

- **For** loop statement to execute a block of code repeatedly with various options.



The **for** Loop

➤ **for** (*initialization* ; *loopContinuationTest* ; *increment*)
 statement;

Or

➤ **for** (*initialization* ; *loopContinuationTest* ; *increment*)
 {
 statement 1;
 statement 2;
 ...
 statement n;
 }

The `for` Loop

- There are three parts that are separated by semi-colons in control block of the C for loop.
 - `initialization expression` is executed before execution of the loop starts. The initialization expression is typically used to initialize a counter for the number of loop iterations. You can initialize a counter for the loop in this part.
 - The `execution of the loop` continues until the loop condition is false. This expression is checked at the beginning of each loop iteration.
 - The `increment expression`, is usually used to increase (or decrease) the loop counter. This part is executed at the end of each loop iteration.

Increment and Decrement Operators

- C provides two operators for incrementing and decrementing the value of variables.
- The increment operator `++` add 1 to its operand.
- The decrement operator `--` subtracts 1 from its operand
- The C increment and decrement operators can be used either as prefix operator or postfix operators as follows:
 - `variable++;`
 - `variable--;`
 - `++variable;`
 - `--variable;`

Increment and Decrement Operators

- The C increment operator in both prefix or postfix contexts is to add 1 to a variable.
- But,
- the expression *`++variable`* increments *variable* before its value is used,
- whereas *`variable++`* increments *variable* after its value has been used.

Example:

- If **x = 10**, then
 - `printf("%d", ++x);`
 - Prints 11
 - `printf("%d", x++);`
 - Prints 10

Assignment Operators

- Assignment operators abbreviate assignment expressions
`i = i + 3;` can be abbreviated as `i += 3;` using the addition assignment operator

- Statements of the form

variable = variable operator expression ;

can be rewritten as

variable operator= expression ;

- Example:

<code>a -= 1</code>	<code>(a = a - 1)</code>
<code>a *= 2</code>	<code>(a = a * 2)</code>
<code>a /= 3</code>	<code>(a = a / 3)</code>
<code>a %= 4</code>	<code>(a = a % 4)</code>

The `for` Loop – Example 1

- Write a C program that calculates the sum of numbers from 1 to 50.
- Then displays the sum as an output.
- Use for loop instead of Gauss formula.
- Pseudocode:

Start

Initialize `sum = 0`

For (`counter = 1; counter <= 50; counter++`)

`sum=sum+counter`

Display sum

Stop

Example 1

```
#include <stdio.h>
#include <conio.h>
main()
{
    int sum=0,counter;
    for (counter=1;counter<=50;counter++)
        sum+=counter;
    printf("Sum = %d",sum);
    getch();
    return(0);
}
```

Example 1 – same solution

```
#include <stdio.h>
#include <conio.h>
main()
{
    int sum,counter;
    sum=0;
    for (counter=1;counter<=50;counter=counter+1)
        sum=sum+counter;
    printf("Sum = %d",sum);
    getch();
    return(0); }
```

Example – 2

- Write an algorithm which calculates the average **exam grade** for a class of n students (n is given by user).
- What are the program **inputs**?
 - the exam grades
- **Processing**:
 - Find the sum of the grades;
 - count the number of students;
 - calculate average grade = sum of grades / number of students.
- What is the program **output**?
 - the average exam grade

Example – 2

```
#include <stdio.h>      #include <conio.h>
main()                  {
int sum=0,counter,n,g; /*n: # of students, g:grade */
float ave;
printf("Enter number of students: ");    scanf("%d",&n);
for (counter=1;counter<=n;counter++)
{   printf("Enter %d student grade: ",counter);
    scanf("%d",&g);
    sum+=g;   }   ave=sum/n;
printf("Average grade of class = %4.1f",ave);
getch();    return(0);  }
```

Example – 3

- Write a C program that calculates the geometrical mean of numbers given by user.
- Then program displays geometrical mean as an output.
- Use `pow()` function to calculate geometrical mean.

Example – 3

```
#include <stdio.h>
#include <conio.h>
#include <math.h>
main()
{
int counter,n,x; /*n: total numbers, x: each number */
float geo,sum=1; /*geo: geometrical mean */
printf("How many numbers do you have: ");
scanf("%d",&n);
```

Example – 3

```
for (counter=1;counter<=n;counter++)  
{  
    printf("Enter %d number: ",counter);  
    scanf("%d",&x);  
    sum*=x;  
}  
geo=pow(sum,(1.0/n));  
printf("Geometrical mean = %4.2f",geo);  
getch();  
return(0);  
}
```

Example – 4

- Write a C program that calculates the factorial of a number n given by user.
- Then program displays the factorial as an output.
- Be careful that $n \geq 0$
- If $n < 0$ then use `abort()` ; function

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

```
abort() ;
```

Example – 4

```
#include <stdio.h>      #include <conio.h>
#include <stdlib.h>
main()                  {
int n,f=1,c;            /*n: number, f: factorial, c:counter */
printf("Enter positive number to calculate its factorial: ");
scanf("%d",&n);
if (n<0) abort();
else if (n==0);
else for (c=1;c<=n;c++) f*=c;
printf("Factorial of %d is %d",n,f);
getch();    return(0); }
```

Homework for next week

- Try to write programs that calculate,
 - C to F conversion for each degrees
 - $1! + 2! + 3! + \dots + n!$
 - Combination

$$\binom{m}{n} = \frac{m!}{n!(m-n)!}$$



While loop - Do while loop - Debugging - Random numbers

Assoc. Prof. Özgür ZEYDAN

<https://ozgurzeydan.com.tr/>



`break` and `continue` Statements

- `break` statement is used to terminate any type of loop such as while loop, do while loop and for loop. C break statement terminates the loop body immediately and passes control to the next statement after the loop.
- `continue` statement is used to skip over the rest of the current iteration. After continue statement, the control returns to the top of the loop.

break and continue Statements

continue;

➤ Example:

```
for (i=0 ; i<10 ; i++)  
{  
    if (i==5) continue;  
    printf ("%d\n", i) ;  
}
```

- The numbers 0 through 9 are printed except for 5.

break;

➤ Example:

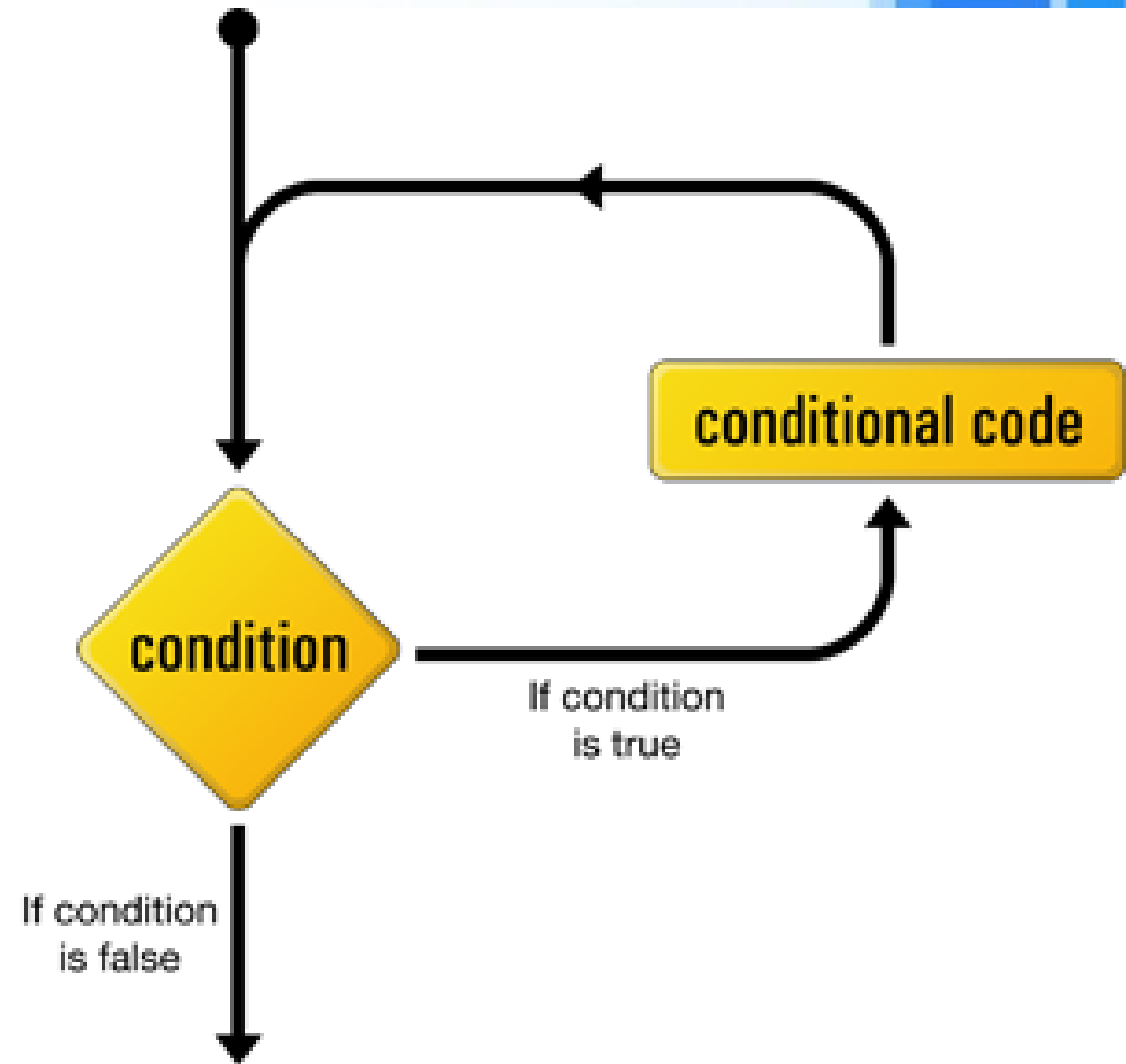
```
for (i=0 ; i<10 ; i++)  
{  
    if (i==5) break;  
    printf ("%d\n", i) ;  
}
```

- The numbers 0 through 4 are printed.

While loop

- A loop statement allows you to execute a block of code repeatedly.
- The C while loop is used when you want to execute a block of code repeatedly with checked condition before making an iteration.
- `while (expression) statement...`
- *statement* is executed repeatedly as long as *expression* is true. The test on *expression* takes place before each execution of *statement*.

While Loop

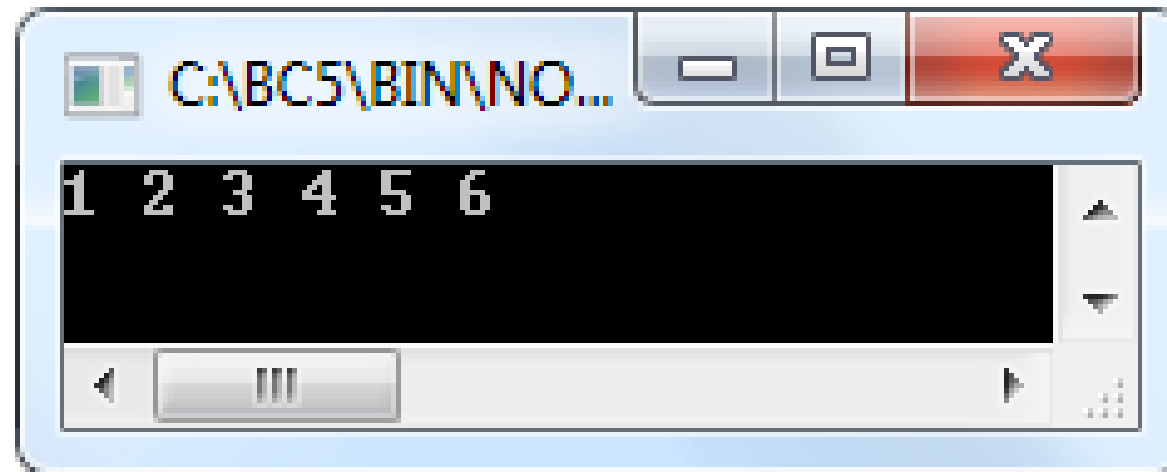


While loop

➤ Example:

```
j = 1;  
while (j <= 6 )  
{  
    printf("%d ", j);  
    ++j;  
}
```

Output:



Infinite loops

```
➤ while ( 10 )  
{    statement;    }
```

```
➤ while ( -12.6 )  
{    statement;    }
```

Creates infinite loops. Expression part is always TRUE (1)

```
➤ while ( 0 )  
{    statement;    }
```

Never executed. Expression part is FALSE (0)

Example – 1

- Write a C program that reads in n (n is given by user) numbers and displays their maximum and minimum values.
- Use **while loop**.

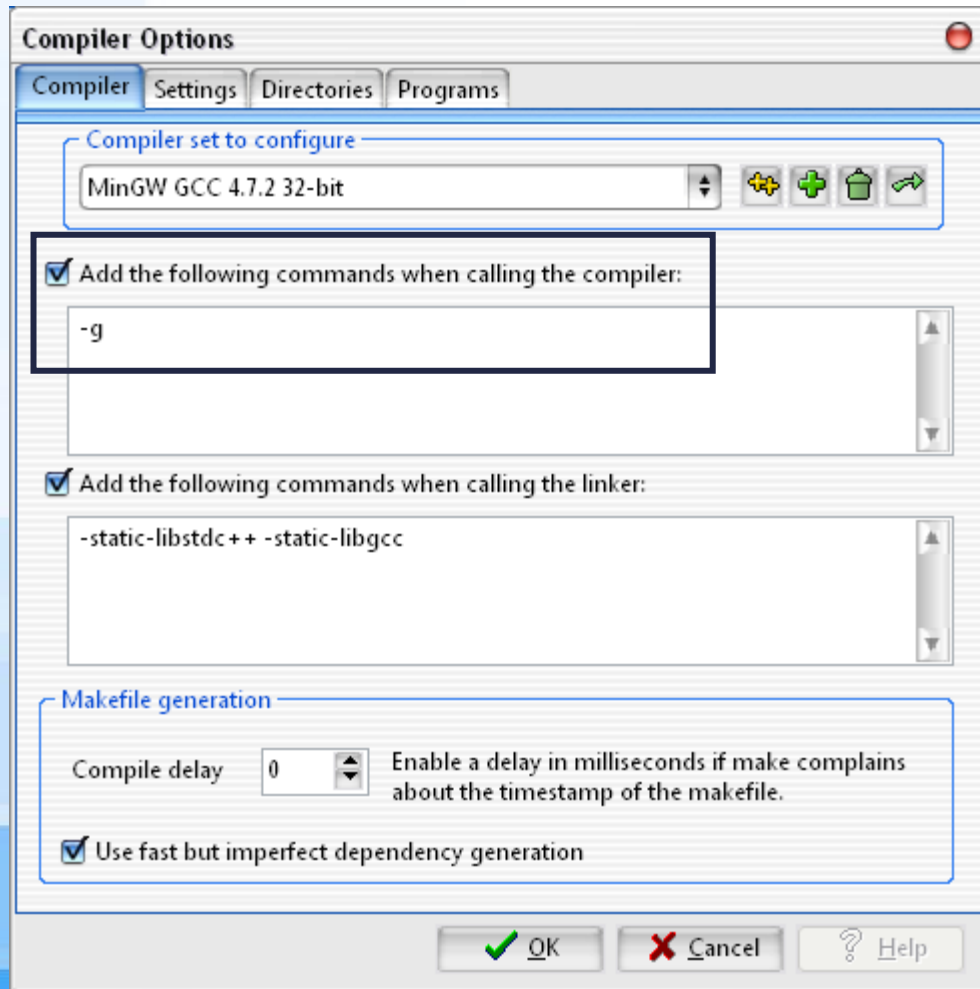
Example – 1

```
#include <stdio.h>
#include <conio.h>
main()
{
    int x, c=1, n, min, max; /* x:number, n: total numbers, c:counter
    */
    printf("How many numbers do you have:");
    scanf("%d", &n);
    printf("Enter the number:");
    scanf("%d", &x);
    max=x;
    min=x;
    c+=1;
```

Example – 1

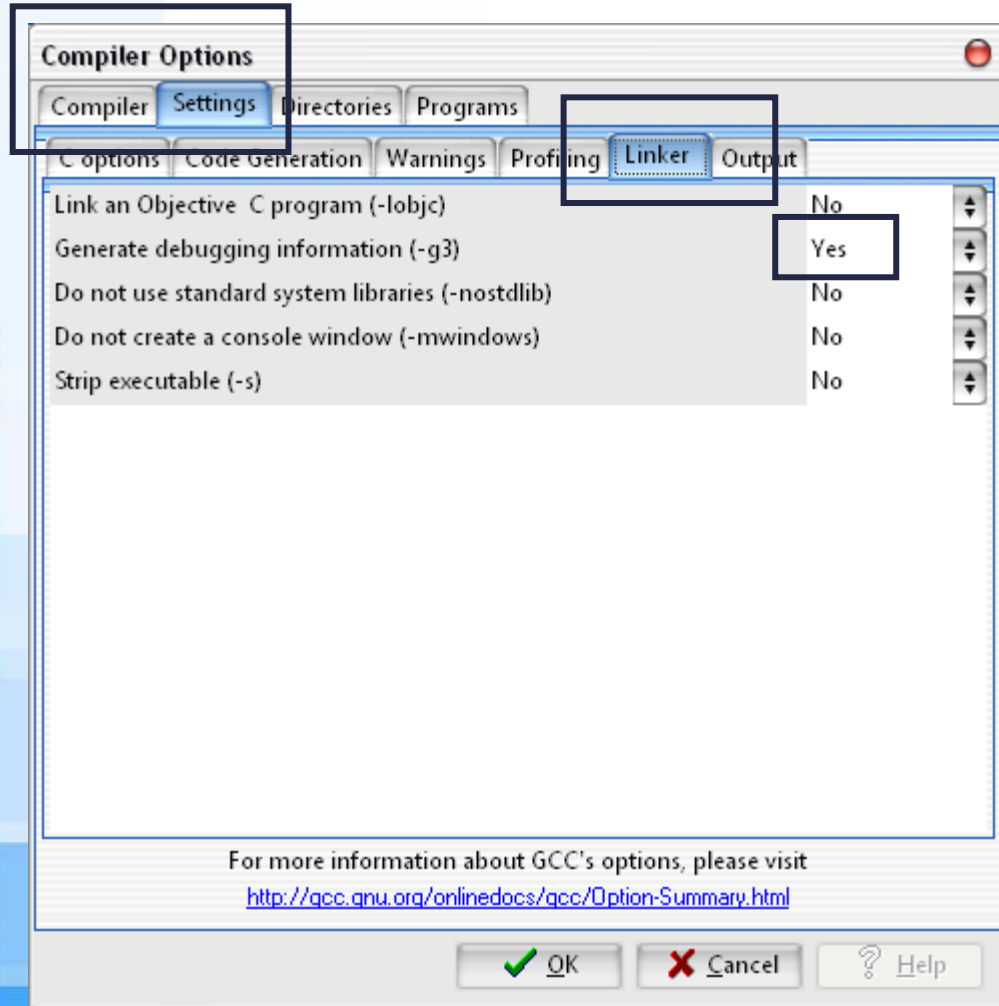
```
while (c<=n)
{
    printf("Enter the number:");
    scanf("%d", &x);
    if (x>max) max=x;
    if (x<min) min=x;
    c+=1;
}
printf("Maximum is %d and minimum is %d",max,min);
getch();
return 0;
}
```

Debugging in Dev C++



- Tools -> Compiler Options
- Check "Add the following commands when calling compiler:"
- type the following in the textbox :
-g

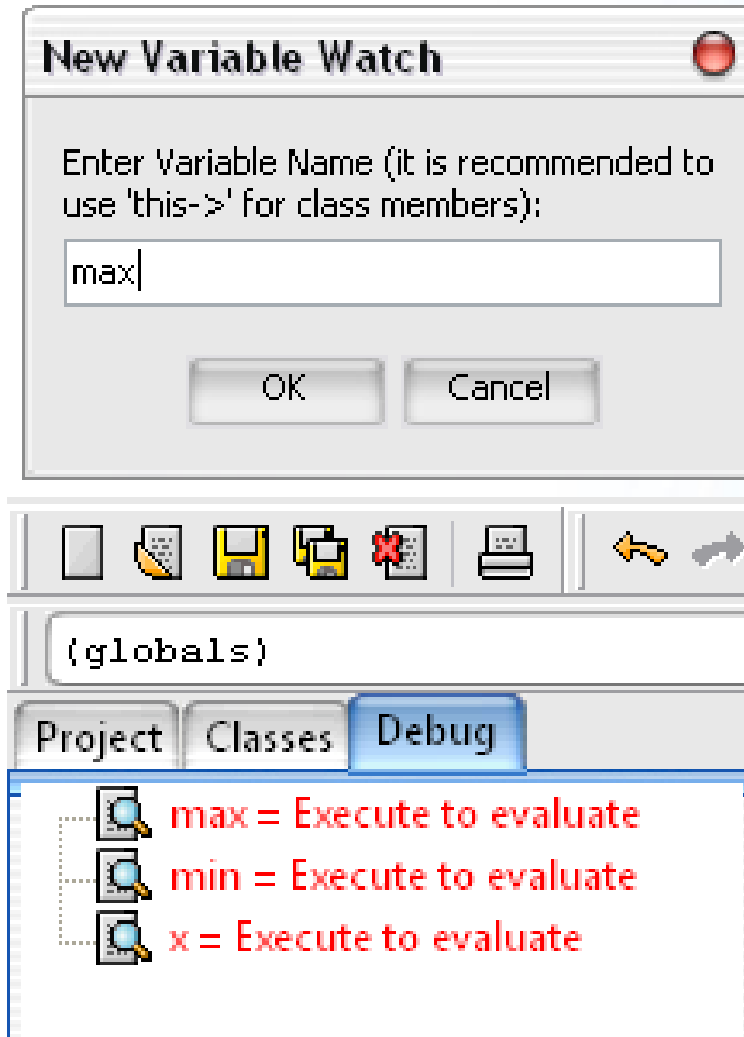
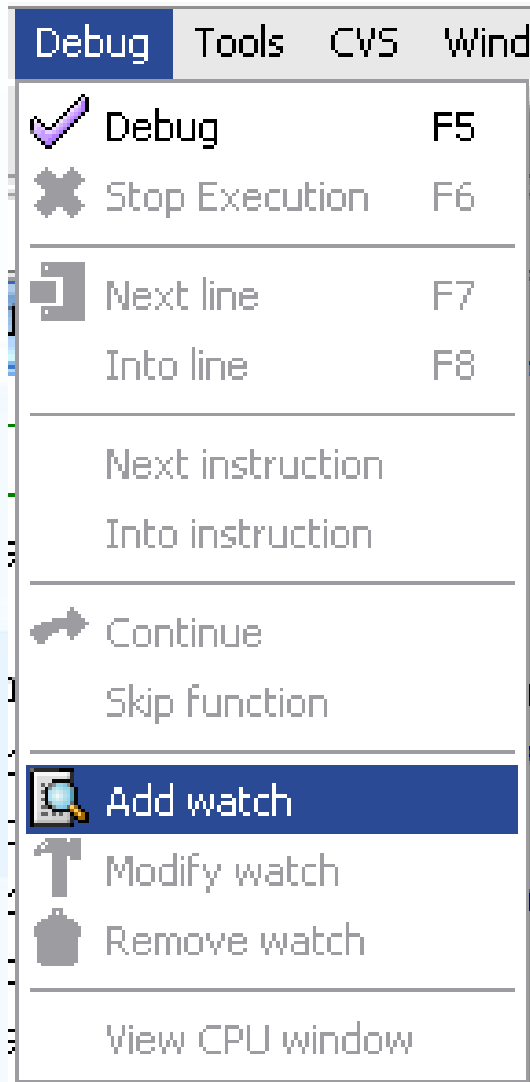
Debugging in Dev C++



- Compiler Options
- Settings tab -> Linker
- select **Yes** under "Generate debugging information"
- click OK

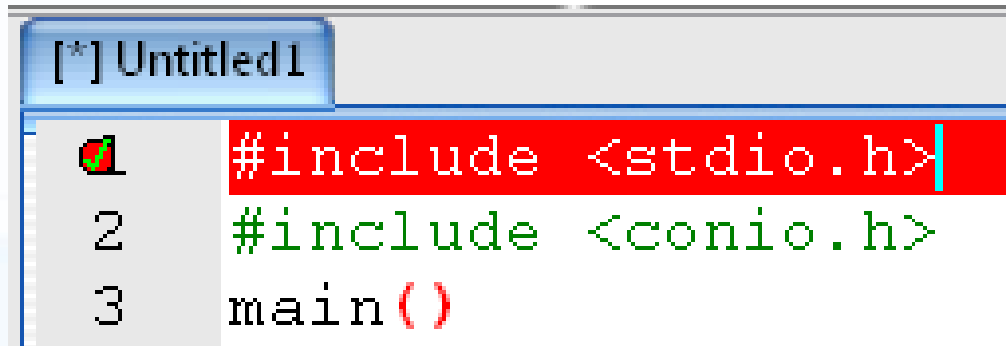
Add watch (Dev C++)

➤ Debug -> add watch



Breakpoint & Debug (Dev C++)

- Right click in first line -> Toggle Breakpoint

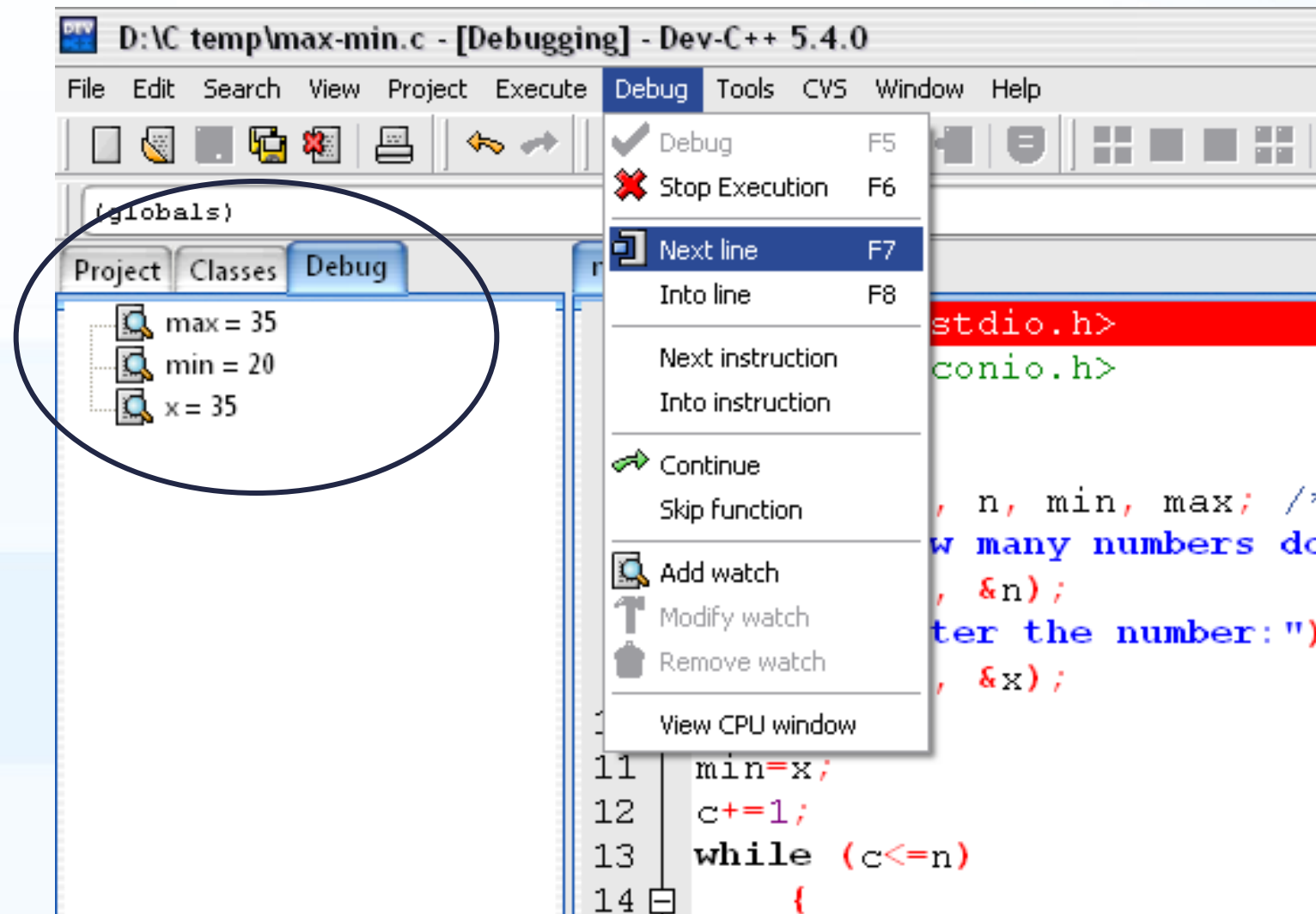


```
[*] Untitled1
1  #include <stdio.h>
2  #include <conio.h>
3  main()
```

- Debug -> Debug (F5)



Debug & Next line (Dev C++)



Example – 2

- Repeat Example – 1 without asking n (how many numbers)
- Use **sentinel controlled loop** instead of using **counter control loop**.
- For instance: use **-999** as sentinel.

Example – 2

```
#include <stdio.h>
#include <conio.h>
main()
{
    int x, min, max;
    printf("Enter the number (-999 to end): ");
    scanf("%d", &x);
    max=x;
    min=x;
```

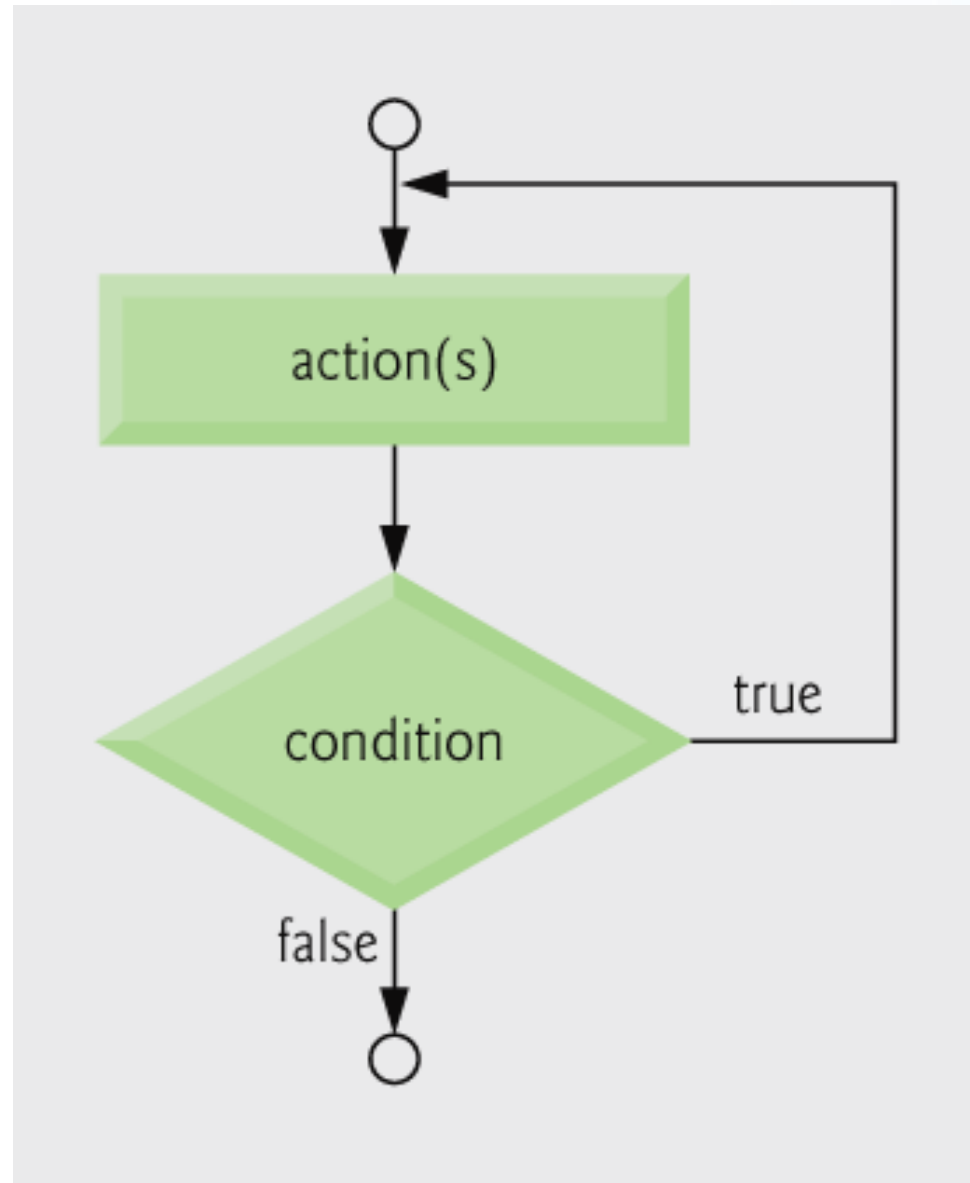
Example – 2

```
while (x!=-999)
{
    printf("Enter the number (-999 to end): ");
    scanf("%d", &x);
    if (x>max && x!=-999) max=x;
    if (x<min && x!=-999) min=x;
}
printf("Maximum is %d and minimum is %d",max,min);
getch();
return 0;
}
```

Do While Loop

- The C do while loop statement consists of a block of code and a Boolean condition.
- First the code block is executed, and then the condition is evaluated.
- If the condition is true, the code block is executed again until the condition becomes false.
- `do statement... while(expression);`
- *statement* is executed repeatedly as long as *expression* is true. The test on *expression* takes place after each execution of *statement*.

Do While Loop



Do While Loop

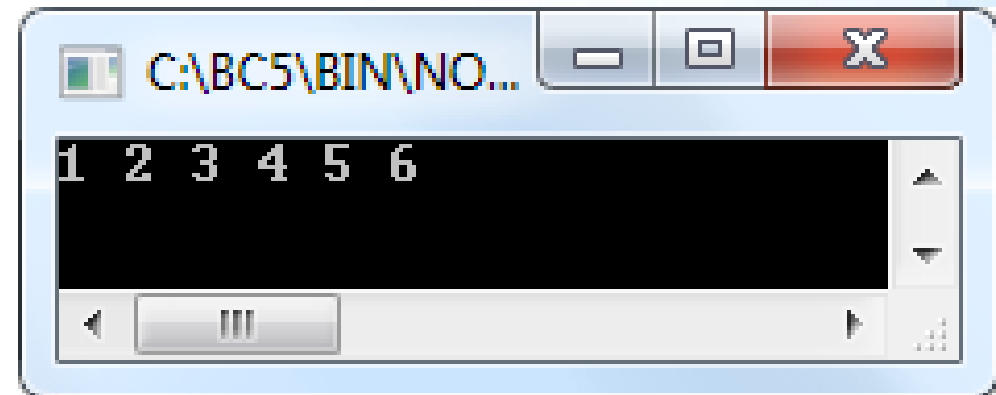
➤ Syntax:

```
do {  
    statement  
} while ( condition );
```

➤ Example:

```
j = 1;  
do {  
    printf("%d ", j);  
    ++j;  
}while (j <= 6 );
```

➤ Output:



Common Programming Errors for Loops

- Infinite loops are caused when the loop-continuation condition in a `while`, `for` or `do...while` statement never becomes false.
- To prevent this, make sure there is not a semicolon immediately after the header of a `while` or `for` statement.
- In a counter-controlled loop, make sure the control variable is incremented (or decremented) in the loop.
- In a sentinel-controlled loop, make sure the sentinel value is eventually input.

Example – 3

- Remember **factorial example** from previous week.
- Rewrite C code for factorial program by using **do-while loop** instead of **for loop**.
- Do not forget to check the number must be positive.

Factorial example (for loop)

```
#include <stdio.h>      #include <conio.h>
#include <stdlib.h>
main()                  {
int n,f=1,c;            /*n: number, f: factorial, c:counter */
printf("Enter positive number to calculate its factorial: ");
scanf("%d",&n);
if (n<0) abort();
else if (n==0);
else for (c=1;c<=n;c++) f*=c;
printf("Factorial of %d is %d",n,f);
getch();    return(0); }
```

Example – 3

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
main()          {
int n,f=1,c;      /*n: number, f: factorial, c:counter
*/
printf("Enter positive number to calculate its factorial: ");
scanf("%d",&n);
if (n<0) abort();
else if (n==0) printf("0! is 1");
```

Example – 3

else

```
{ c=1;
```

```
do { f*=c; c+=1; }
```

```
while (c<=n);
```

```
printf("Factorial of %d is %d",n,f);
```

```
}
```

```
getch();
```

```
return(0);
```

```
}
```

Generating Random Numbers

```
#include <stdlib.h> /* required for randomize() and  
random() */  
randomize(); /* to reset random numbers */  
rand() % value;
```

➤ For example;

```
y = rand() % 100;
```

➤ y will be in the range of 0 though 99.

➤ if you want to generate numbers between the range (max and min):

➤ $r = (\text{rand}() \% (\text{max} + 1 - \text{min})) + \text{min};$

Reseting Random Number

➤ Libraries:

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

```
#include <time.h>
```

➤ Functions:

```
srand (time (0) ) ;
```

or

```
srand (time (NULL) ) ;
```

Example – 4

- Write a number guess game.
- By using **random number generator** function generate a secret number between 0 and 100.
- Then computer will ask user to guess the number.
- Use **if** control structure to display whether,
 - The written number is smaller than secret number
 - The written number is greater than secret number
- To repeat asking guess use **do-while** loop.
- Count the total number of guesses.
- When then secret number is guessed correctly, program displays a message "Congratulations, you have guessed in n times."

Example – 4

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <time.h>
main()
{
    int r,g,c; /*r: random number created by computer
               g: number written by user          c: counter */
    printf(" ** Number Guess Game by Özgür ZEYDAN ** \n");
    srand(time(0));
    r=rand()%101;
    c=0;
```

Example – 4

```
do {  
    printf("Write a number between 0 and 100\n");  
    scanf("%d",&g);  
    c=c+1;  
    if (r>g) printf("My number is bigger than your guess.\n");  
    else if (r<g) printf("My number is smaller than your guess.\n");  
    else printf("Congratulations, you have guessed correctly in %d  
guess.",c);  
}  
while (r!=g);  
getch();    return 0;    }
```

Homework

- Compare structures of loops:
 - For loop
 - While loop
 - Do-while loop



Modular programming - Functions - Recursive Function

Assoc. Prof. Özgür ZEYDAN

<https://ozgurzeydan.com.tr/>



Modular Programming

- A function in C is a **small “sub-program”** that performs a particular task and supports the concept of **modular programming** design techniques.
- In modular programming the various tasks that your overall program must accomplish are assigned to individual functions and the main program basically calls these functions in a certain order.
- The **main** body of a C program, identified by the keyword **main**, and enclosed by left and right braces is a function.
- It is called by the operating system when the program is loaded, and when terminated, returns to the operating system.

Reasons for Modular Programming

- Construct a program from smaller pieces or components
- **Avoids code repetition.** Don't have to repeat the same block of code many times in your code. Make that code block a function and call it when needed.
- **Easy to debug.** Each piece more manageable than the original program
- **Easy to modify and expand.** Just add more functions to extend program capability.
- **Software reusability.** Use existing functions as building blocks for new programs

Function Definitions

- Function definition format:

```
return-value-type  function-name (parameter-list)
{
    declarations and statements
}
```

- **Function-name**: any valid identifier
- **Return-value-type**: data type of the result (default int)
 - void - function returns nothing
- **Parameter-list**: comma separated list, declares parameters (default int)

Function Definitions

```
return-value-type  function-name (parameter-list)
{
    declarations and statements
}
```

- **Declarations and statements: function body**
 - Variables can be declared inside function body
 - Function can not be defined inside another function
- **Returning control**
 - If nothing returned: `return;`
 - If something returned: `return expression;`

Function Prototypes

- Used to validate functions
- Prototype only needed if function definition comes after use in program

➤ Examples:

```
➤ int factor(int);
```

```
/* function prototype */
```

```
➤ void intro (void) ;
```

```
/* function prototype */
```

Functions

- Functions that do not receive or send data.
- You can use:

```
void function-name (void)
{
    Statements;
}
```

Example – 1

- Write a C code to display an introduction message (given below) on the beginning of the program.
- Use function (function will neither receive nor return any value)
- Introduction message:

```
=====
*  Programmer : Özgür ZEYDAN  *
=====
```

Example – 1

```
#include<stdio.h>
#include<conio.h>
void intro(void);    /* function prototype */
int main(void)       /* main function */
{
    intro();          /* function call */
    statements;
}
void intro(void)      /* intro function */
{ printf("=====\n");
  printf("*  Programmer : Özgür ZEYDAN  *\n");
  printf("=====\n\n"); }
```

Example – 2

- Write a C program that asks user to write two integer values.
- Then computer these two integers and returns result.
- Use local variables.

Example – 2

```
#include <stdio.h>
#include <conio.h>
int sum ();      /* decleration */
main() {
    int x,y,result;      /* local variables */
    printf("Enter two integers (space in between): ");
    scanf("%d %d",&x,&y);
    result=sum(x,y);  /* function call */
    printf("Sum of two numbers: %d",result);
    getch();
    return 0; }
int sum(int a, int b)      /* function */
{    return a+b; }
```


Example – 3

- Write a C program that asks user to write two integer values.
- Then computer these two integers and returns result.
- Use global variables.

Example – 3

```
#include <stdio.h>
#include <conio.h>
int x,y;                /* global variables */
int sum ();             /* decleration */
main() {
    int result;         /* local variable */
    printf("Enter two integers (space in between): ");
    scanf("%d %d",&x,&y);
    result=sum();       /* function call */
    printf("Sum of two numbers: %d",result);
    getch();
    return 0; }
int sum()               /* function */
{    return x+y; }
```

Example – 4

- Write a C program that asks user to write an integer values.
- Then computer calculates factorial of a given integer.
- At this time, use modular programming!
- Define a function that
 - receives an integer from main function,
 - calculates and
 - returns value of its factorial.

Example – 4 (using local variables)

```
#include<stdio.h>
#include<conio.h>
int factor(int);          /* function prototype */
int main(void)            /* main function */
{
    int x;                /* local variable */
    printf("Enter integer value to calculate its factorial: ");
    scanf("%d",&x);
    printf("Factorial of %d is %d",x,factor(x)); /* function call */
    getch();
    return(0);  }
```

Example – 4 (using local variables)

```
int factor( int a)
{
int fac=1,i; /* local variables */
for (i=1;i<=a;i++)
fac*=i;
return fac;
}
```

Example – 4 (using global variables)

```
#include<stdio.h>
#include<conio.h>
int x;                /* global variable */
int factor(int);      /* function prototype */
int main(void)        /* main function */
{
    printf("Enter integer value to calculate its factorial: ");
    scanf("%d",&x);
    printf("Factorial of %d is %d",x,factor(x)); /* function call */
    getch();
    return(0);  }
```

Example – 4 (using global variables)

```
int factor( int x)
{
int fac=1,i; /* local variables */
for (i=1;i<=x;i++)
fac*=i;
return fac;
}
```

Example – 5

- Write a C program that calculates Combination of a given two numbers `m` and `n`.
- Use a function to make factorial calculations.

$$\binom{m}{n} = \frac{m!}{n!(m-n)!}$$

Example – 5

```
#include <stdio.h>
#include <conio.h>
int factor(int);      /* function prototype */
int main(void)        /* main function */
{
    int m,n;
    printf("Combination (m/n) calculator.\n");
    printf("Enter m and n values: ");
    scanf("%d %d",&m,&n);
    printf("Combination of (%d/%d) is
           %d",m,n,factor(m)/(factor(n)*factor(m-n)));
    call */           /* function
```

Example – 5

```
getch();  
return(0); }
```

```
int factor( int x)  
{  
    int fac=1,i;      /* local variables */  
    for (i=1;i<=x;i++)  
        fac*=i;  
    return fac;  
}
```

Example – 6

- Write a C program that calculates fifthroot of a given floating point number.
- Use function to calculate `fifthroot`.

Example – 6

```
#include <stdio.h>
#include <conio.h>
#include <math.h>
float x,y;
float fifthroot ();
main() {
    printf("Fifthroot calculator\n");
    printf("Enter a number: ");
    scanf("%f",&x);
    y=fifthroot(x);
    printf("Fifthroot of %f is %f",x,y);
```

Example – 6

```
    getch();  
    return 0; }  
float fifthroot()  
{  
    float z;  
    z=pow(x,(1.0/5.0));  
    return z;  
}
```

C Recursive Function

- Recursive function is a function that contains a call to itself. C supports creating recursive function with ease and efficient.
- Recursive function allows you to divide your complex problem into identical single simple cases which can handle easily.
- Recursive function must have at least one exit condition that can be satisfied. Otherwise, the recursive function will call itself repeatedly until the runtime stack overflows.

Example of Using Recursive Function

- Recursive function is closely related to definitions of functions in mathematics so we can solving factorial problems using recursive function.
- All you know in mathematics the factorial of a positive integer N is defined as follows:
 - $N! = N*(N-1)*(N-2)...2*1;$
 - Or in a recursive way:
 - $N! = 1$ if $N \leq 1$ and $N*(N-1)!$ if $N > 1$

Example – 7

```
# include<stdio.h>
int factor(int a)
{
    if(a <= 1) return 1;
    return a * factor (a - 1);
}
void main()
{
    ...
    printf("Factorial of %d is %d",x,factor(x));
    ... }
```




Arrays - Bubble Sorting Algorithm

Assoc. Prof. Özgür ZEYDAN

<https://ozgurzeydan.com.tr/>



Arrays

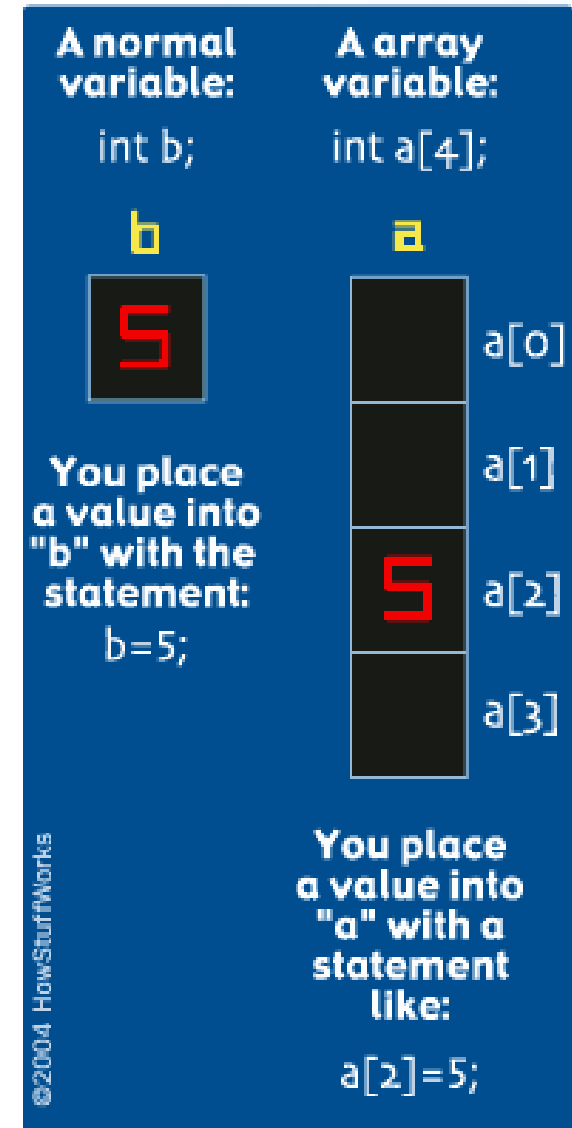
- Array is a collection of same type elements under the same variable identifier referenced by index number.
- Arrays are widely used within programming for different purposes such as sorting, searching and etc.
- Arrays allow you to store a group of data of a single type.
- Arrays can be created from any of the C data-types int, float, double or char.
- `<variable type>`
`array_name[size_of_array];`

Arrays

- An array lets you declare and work with a collection of values of the same type. For example, you might want to create a collection of four integers. One way to do it would be to declare four integers directly:
- `int a, b, c, d;`
- This is okay, but what if you needed a thousand integers? An easier way is to declare an array of four integers:
- `int a[4];`
- The four separate integers inside this array are accessed by an **index**.
- All arrays start at index **zero** and go to **n-1** in C.

Arrays

- Thus, `int a[4];` contains four elements.
- `int a[4];`
 - `a[0] = -18;`
 - `a[1] = 90;`
 - `a[2] = 5;`
 - `a[3] = 43;`



Arrays

- Using a loop to manipulate the index:

```
int a[4];  
int i;  
for (i=0; i<4; i++)  
a[i] = 0;
```

Initializers

```
int x[ 5 ] = { 1, 2, 3, 4, 5 };
```

- If not enough initializers, rightmost elements become 0

```
int x[ 5 ] = { 0 } (All elements 0)
```

- If too many initializers, a syntax error occurs
- C arrays have no bounds checking
- If size omitted, initializers determine it

```
int x[ ] = { 1, 2, 3, 4, 5 };
```

(5 initializers, therefore 5 element array)

Two Dimensional Arrays

- Tables with rows and columns (m by n array)
- Like matrices: **specify row, then column**
- Initialization

```
int b[ 2 ][ 2 ] = { { 1, 2 }, { 3, 4 } };
```

- Initializers grouped by row in braces
- If not enough, unspecified elements set to zero

```
int b[ 2 ][ 2 ] = { { 1 }, { 3, 4 } };
```

- Referencing elements

- Specify row, then column

```
printf( "%d", b[ 0 ][ 1 ] );
```

Example – 1

- Write a C code to create one dimensional array with size 10.
- Computer will ask user to write integer values to that array.
- Finally, values of an array will be displayed on the screen.

Example – 1

```
#include <stdio.h>
#include <conio.h>
int main(void)
{ int i, a[10];
  for (i=0; i<10; i++)
    { printf("Write %d. integer: ",i+1);
      scanf("%d",&a[i]); }
  for (i=0; i<10; i++)
    printf("%d. value: %d\n",i+1,a[i]);
  getch();
  return(0); }
```

Example – 2

- Modify your first program to find the **maximum** and **minimum** of the integers given by user.

Example – 2

```
#include <stdio.h>
#include <conio.h>
int main(void)
{ int i, a[10], max, min;
  max=0;
  min=0;
  for (i=0; i<10; i++)
  { printf("Write %d. integer: ",i+1);
    scanf("%d",&a[i]);
    if (a[i]>a[max]) max=i;
    if (a[i]<a[min]) min=i;  }
```

Example – 2

```
for (i=0; i<10; i++)  
    printf("%d. value: %d\n",i+1,a[i]);  
printf("\nMax value: %d\n",a[max]);  
printf("\nMin value: %d\n",a[min]);  
getch();  
return(0); }
```

Bubble Sort

- The simplest sorting algorithm is bubble sort.
- The bubble sort works by iterating down an array to be sorted from the first element to the last, comparing each pair of elements and switching their positions if necessary.
- This process is repeated as many times as necessary, until the array is sorted.

Bubble Sort - Animation

6 5 3 1 8 7 2 4

<http://en.wikipedia.org/wiki/File:Bubble-sort-example-300px.gif>

Bubble Sort Algorithm

```
for(int x=0; x<n; x++)  
{   for(int y=0; y<n-1; y++)  
    { if(a[y]>a[y+1])  
        { temp = a[y+1];  
          a[y+1] = a[y];  
          a[y] = temp;  
        }  
    }  
}
```

Example – 3

- Write a C code that asks user to give values of two 3x3 matrices (Matrix A and matrix B).
- Then computer calculates matrix C which is equals to sum of two matrices.
- Finally, computer displays C matrix.
- The sum $A+B$ of two m -by- n matrices A and B is calculated entrywise:
- $C_{i,j} = A_{i,j} + B_{i,j}$, where $1 \leq i \leq m$ and $1 \leq j \leq n$.
- Example:
$$\begin{bmatrix} 1 & 3 & 1 \\ 1 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 5 \\ 7 & 5 & 0 \end{bmatrix} = \begin{bmatrix} 1+0 & 3+0 & 1+5 \\ 1+7 & 0+5 & 0+0 \end{bmatrix} = \begin{bmatrix} 1 & 3 & 6 \\ 8 & 5 & 0 \end{bmatrix}$$

Example – 3

```
#include<stdio.h>
#include<conio.h>
void main()
{ int a[3][3],b[3][3],c[3][3],i,j;
printf("Write integer values of A matrix\n");
for(i=0;i<3;i++)
{for(j=0;j<3;j++)
{printf("Value of %d. row and %d. column: ",i+1,j+1);
scanf("%d",&a[i][j]);    }  }
printf("Write integer values of B matrix\n");
```

Example – 3

```
for(i=0;i<3;i++)
{for(j=0;j<3;j++)
{printf("Value of %d. row and %d. column: ",i+1,j+1);
scanf("%d",&b[i][j]); } }
printf("C matrix\n");
for(i=0;i<3;i++)
{for(j=0;j<3;j++)
{c[i][j]=a[i][j]+b[i][j];
printf("%4d",c[i][j]); }
printf("\n"); }
getch(); }
```

Example – 4

- Modify your previous C code that first asks dimensions (m and n) of $A_{m \times n}$ and $B_{m \times n}$ matrices.
 - Force user to write m and n values smaller than max size of matrices (for instance 20) and bigger than 0 by using **do-while loop**.
 - Be careful that m and n must be the same for all matrices to make addition.
- Next programs asks values of two matrices.
- Then computer calculates matrix C which is equals to sum of two matrices.
- Finally computer displays C matrix.

Example – 4

```
#include<stdio.h>
#include<conio.h>
void main()
{ int a[20][20],b[20][20],c[20][20],i,j,m,n;
do{
printf("Write dimensions of matrices (m and n)(20 MAX): ");
scanf("%d %d",&m,&n);
} while(m<=0 || m>20 || n<=0 || n>20);
printf("Write integer values of A matrix\n");
for(i=0;i<m;i++) {for(j=0;j<n;j++)
{printf("Value of %d. row and %d. column: ",i+1,j+1);
scanf("%d",&a[i][j]);    }    }
```

Example – 4

```
printf("Write integer values of B matrix\n");
for(i=0;i<m;i++)
{for(j=0;j<n;j++)
{printf("Value of %d. row and %d. column: ",i+1,j+1);
scanf("%d",&b[i][j]);    }    }
printf("\nC matrix\n");
for(i=0;i<m;i++)
{for(j=0;j<n;j++)
{c[i][j]=a[i][j]+b[i][j];
printf("%4d",c[i][j]);    }
printf("\n");    }
getch();    }
```

Matrix Multiplication

- Multiplication of two matrices is defined only if the number of columns of the left matrix is the same as the number of rows of the right matrix.
- If A is an *m-by-n* matrix and B is an *n-by-p* matrix, then their matrix product c is the *m-by-p* matrix whose entries are given by dot product of the corresponding row of A and the corresponding column of B:

$$c_{ij} = a_{i1}b_{1j} + a_{i2}b_{2j} + \cdots + a_{in}b_{nj} = \sum_{k=1}^n a_{ik}b_{kj},$$

- for $i = 1, \dots, m$ and $j = 1, \dots, p$.
- *Source:* https://en.wikipedia.org/wiki/Matrix_multiplication

Matrix Multiplication

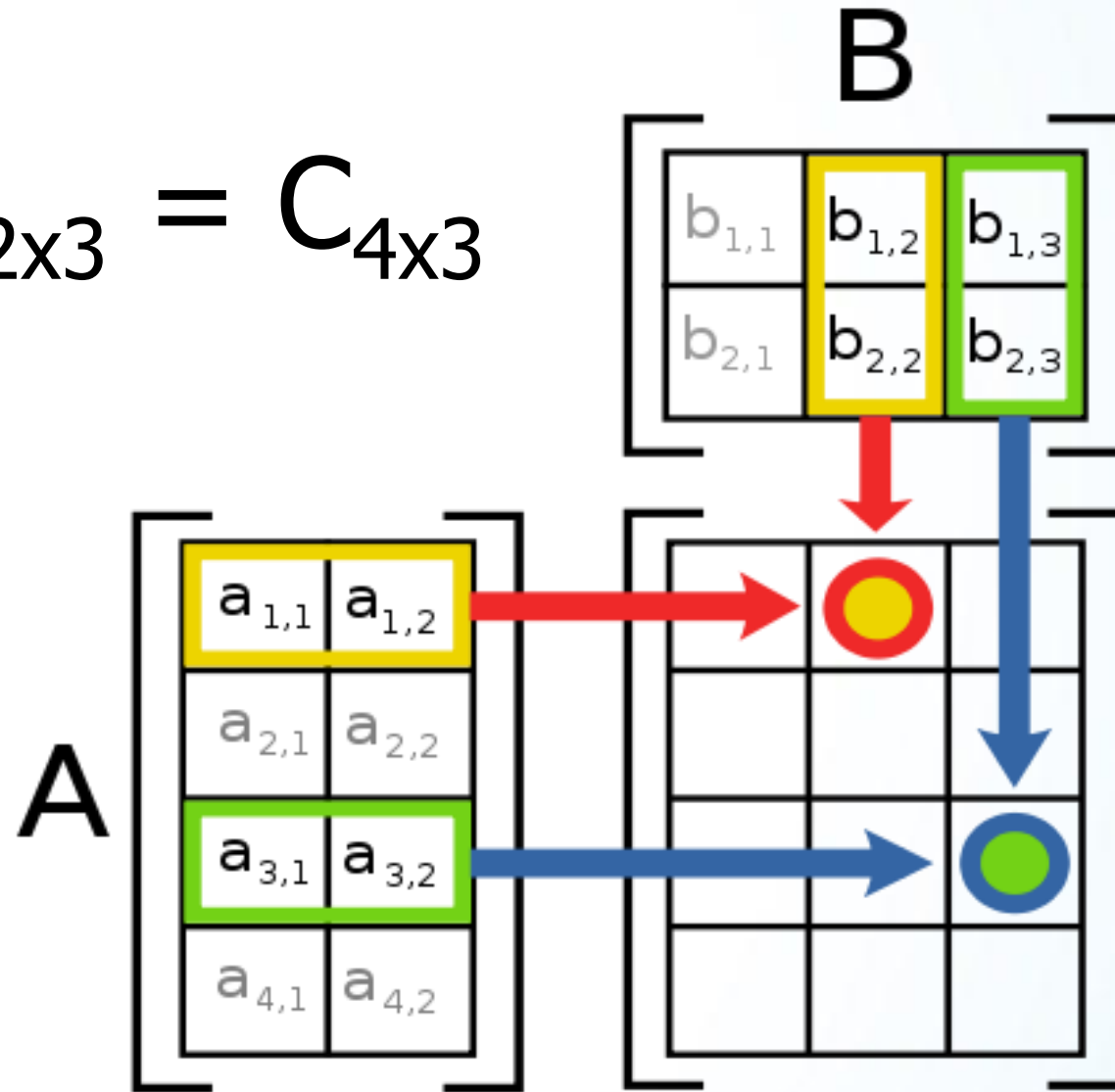
- For example, the underlined entry 2340 in the product is calculated as:

- $(2 \times 1000) + (3 \times 100) + (4 \times 10) = 2340$

$$\begin{bmatrix} \underline{2} & \underline{3} & \underline{4} \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & \underline{1000} \\ 1 & \underline{100} \\ 0 & \underline{10} \end{bmatrix} = \begin{bmatrix} 3 & \underline{2340} \\ 0 & 1000 \end{bmatrix}.$$

Matrix Multiplication

$$A_{4 \times 2} \times B_{2 \times 3} = C_{4 \times 3}$$



Matrix Multiplication Algorithm in C

```
for (i=0 ; i<row1 ; i++)  
    {   for (j=0 ; j<column2 ; j++)  
        {   c[i][j]=0 ;  
            for (k=0 ; k<column1 ; k++)  
            {  
                c[i][j]=c[i][j]+a[i][k]*b[k][j] ;  
            }  
        }  
    }
```

Example – 5

- Write a C code that asks user to give dimensions (m, n and p) of $A_{m \times n}$ and $B_{n \times p}$ matrices.
 - Force user to write m and n values smaller than max size of matrices (for instance 20) and bigger than 0 by using **do-while loop**.
- Then computer calculates matrix C which is equals to multiplication of two matrices.
- Finally computer displays C matrix.

Example – 5

```
#include<stdio.h>
#include<conio.h>
void main()
{ int a[20][20],b[20][20],c[20][20],i,j,k,m,n,p;
do{
printf("How many rows (m) in matrix A (20 MAX): ");
scanf("%d",&m);
printf("How many columns (n) in matrix A (20 MAX): ");
scanf("%d",&n);
printf("How many columns (p) in matrix B (20 MAX): ");
scanf("%d",&p);
} while(m<=0 || m>20 || n<=0 || n>20 || p<=0 || p>20);
```

Example – 5

```
printf("Write integer values of A matrix\n");
for(i=0;i<m;i++)
{for(j=0;j<n;j++)
{printf("Value of %d. row and %d. column: ",i+1,j+1);
scanf("%d",&a[i][j]);    }  }
printf("Write integer values of B matrix\n");
for(i=0;i<n;i++)
{for(j=0;j<p;j++)
{printf("Value of %d. row and %d. column: ",i+1,j+1);
scanf("%d",&b[i][j]);    }  }
```

Example – 5

```
printf("\nC matrix\n");
for(i=0;i<m;i++)
{ for(j=0;j<p;j++)
  { c[i][j] = 0;
    for(k=0;k<n;k++)
    { c[i][j] = c[i][j] + a[i][k] * b[k][j];
    }
    printf("%6d",c[i][j]);
  }
  printf("\n");
}
getch();    }
```

Homeworks

- Write a C code that sorts the values of one dimensional array by using "Bubble Sort Algorithm".
- Write a C code that calculates and displays determinant of a square matrix.
- Write a C code that calculates and displays transpose of a matrix.



Strings - File Operations

Assoc. Prof. Özgür ZEYDAN

<https://ozgurzeydan.com.tr/>



Two Dimensional Array Example

- Write a C program that calculates standard deviation of given numbers.
- Formula of standard deviation:

$$s = \sqrt{\frac{1}{N-1} \sum_{i=1}^N (x_i - \bar{x})^2},$$

- You may use two dimensional array:

std[N][3]

i	x_i	$x_i - x_{ave}$	$(x_i - x_{ave})^2$
0			
1			
2			
...			
...			
N-2			
N-1			

Characters in C

- **char** is a one-byte data type capable of holding a character.
- Characters in C consist of any printable or nonprintable character in the computer's character set including lowercase letters, uppercase letters, decimal digits, special characters and escape sequences.
 - 'a', 'b', 'c', ... 'z', '0', '1', ... '9', '+', '-', '=', '!', '~', etc, '\n', '\0', etc.
- May be used in arithmetic expressions
 - Add, subtract, multiply, divide, etc.

Strings in C

- A string is a character array terminated by '**\0**'.
- '**\0**' is a **NULL** character which indicates the end of the string.
- `char a[];` OR `char *a;`
- `char b[10] = "Hi there!";`
- in memory :

H	i		t	h	e	r	e	!	\0
---	---	--	---	---	---	---	---	---	----

- `char c[12] = "Hi there!";`
- in memory :

H	i		t	h	e	r	e	!	\0	\0	\0
---	---	--	---	---	---	---	---	---	----	----	----

Placeholders in Strings

- You can **print string types with** `printf` by using different placeholders:
 - **char** (single character values) uses `%c`
 - **character strings** (arrays of characters) use `%s`

Example – 1

```
#include<stdio.h>
#include<conio.h>
void main()
{
char b[20]="C Programming";
int i;
printf("%s\n\n",b);
for(i=0;i<=13;i++)
    printf("%c\n",b[i]);
getch();
}
```

String Functions

- Requires `#include <string.h>`
- `strcpy (str1, str2);`
- Copies a string from a source address (str2) to a destination address (str1)
- `char name[15];`
- `strcpy(name, "Ozgur hoca");`
- `printf("%s\n", name);`

String Functions

- Requires `#include <string.h>`
- `strcat (str1, str2) ;`
- Takes two C-strings as input. Adds the contents of the second string (str2) to the end of the first string (str1).
- `char str1[15] = "Hello ";`
- `char str2[30] = "Civil Engineers!";`
- `strcat(str1, str2);`
- No automatic bounds checking: programmer must ensure that str1 has enough room for result.

String Functions

- Requires `#include <string.h>`
- `strlen(str1) ;`
- Returns length of a string.
- `printf("%d",strlen("Ozgur hoca"));`
- Screen output: 10

Example – 2

```
#include <string.h>
#include <stdio.h>
#include <conio.h>
void main()
{
    char my_string[20];
    char *blank = " ", *str1 = "Ozgur", *str2 = "Zeydan";
    strcpy(my_string, str1);
    strcat(my_string, blank);
    strcat(my_string, str2);
    printf("%s\n", my_string);
    printf("%d",strlen(my_string));
    getch();    }
```


File Commands in C

- C supports several functions that can perform basic file operations, which include:
 - Naming a file
 - Opening a file
 - Reading from a file
 - Writing data into a file
 - Closing a file

File Operation Functions in C

Function Name	Operation
<code>fopen()</code>	Creates a new file for use Opens a new existing file for use
<code>fclose</code>	Closes a file which has been opened for use
<code>getc()</code>	Reads a character from a file
<code>putc()</code>	Writes a character to a file
<code>fprintf()</code>	Writes a set of data values to a file
<code>fscanf()</code>	Reads a set of data values from a file
<code>getw()</code>	Reads a integer from a file
<code>putw()</code>	Writes an integer to the file
<code>fseek()</code>	Sets the position to a desired point in the file
<code>ftell()</code>	Gives the current position in the file
<code>rewind()</code>	Sets the position to the begining of the file

Opening Text Files

- Use `fopen` to open a file.
- It opens a file for a specified mode (the three most common are:
 - `r`: read
 - `w`: write
 - `a`: append
- It then returns a file pointer that you use to access the file.
- `FILE *myfile;`
- `myfile=fopen("output.txt","w");`

Opening Text Files

- `myfile=fopen("output.txt","w");`
- The `fopen` statement here opens a file named `output.txt` with the `w` mode.
- This is a destructive write mode, which means that
 - if out does not exist it is created,
 - but if it does exist it is destroyed and a new file is created in its place.
- The `fopen` command returns a pointer to the file, which is stored in the variable `f`. This variable is used to refer to the file.

Opening Text Files with File Name

- `char filename[10];`
- `printf("Write filename: ");`
- `scanf("%s", filename);`
- `fp=fopen(filename, "w");`

File Opening Modes

- **r** : (reading from the file) If the file exists, loads it into memory and sets up a pointer which points to the first character in it. If the file doesn't exist it returns NULL.
- **w** : (writing to the file) If the file exists, its contents are overwritten. If the file doesn't exist, a new file is created. Returns NULL, if unable to open file.
- **a** : (appending new contents at the end of file) If the file exists, loads it into memory and sets up a pointer which points to the first character in it. If the file doesn't exist, a new file is created. Returns NULL, if unable to open file.

File Opening Modes

- **r+** : (reading existing contents, writing new contents, modifying existing contents of the file) If it exists, loads it into memory and sets up a pointer which points to the first character in it. If file doesn't exist it returns NULL.
- **w+** : (writing new contents, reading them back and modifying existing contents of the file) If the file exists, its contents are destroyed. If the file doesn't exist a new file is created. Returns NULL, if unable to open file.
- **a+** : (reading existing contents, appending new contents to end of file. Cannot modify existing contents) If the file exists, loads it in to memory and sets up a pointer which points to the first character in it. If the file doesn't exist, a new file is created. Returns NULL, if unable to open file.

Writing & Reading Data

- Use `fprintf()` function to write data to a text file.
- Use `fscanf()` function to read data from a text file.

- `int x,a;`

- `fprintf(myfile,"%d",x);`

- `fscanf(myfile2,"%d",&a);`

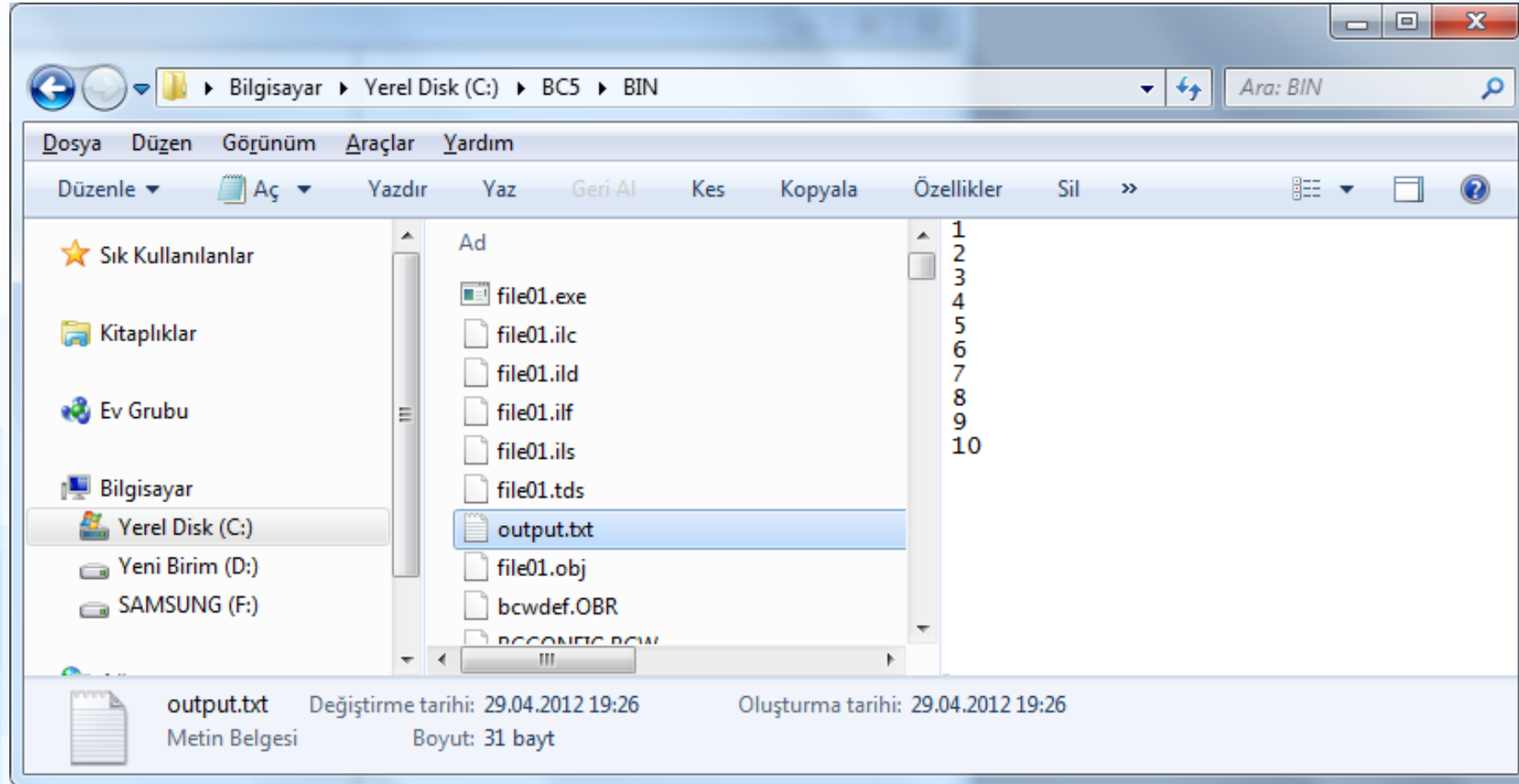
Closing Text Files

- When we have finished reading from the file, we need to close it.
- This is done using the function `fclose ()` through the statement,
- `fclose (myfile) ;`

Example – 3

```
#include <stdio.h>      #include <conio.h>
#define MAX 10
main()      {
FILE *myfile;
int x;
myfile=fopen("output.txt","w");
if (!myfile) return 1;
for(x=1; x<=MAX; x++)      fprintf(myfile,"%d\n",x);
fclose(myfile);
printf("Writing file complete. Press anykey to exit.");
getch();      return 0;      }
```

"output.txt" file



Example – 4

- Modify your previous code so that:
 - Your program will open `output.txt` file as source.
 - Read data from each line.
 - Calculate square of each value and write these values into `output2.txt` file.
 - Then program closes two files and terminates itself.

Example – 4

```
#include <stdio.h>      #include <conio.h>      #define MAX 10
int main()  {
FILE *f1, *f2;
int x,a;
f1=fopen("output.txt","r");      f2=fopen("output2.txt","w");
if (!f1 || !f2) return 1;
for(x=1; x<=MAX; x++)      {
    fscanf(f1,"%d",&a);      fprintf(f2,"%d\n",a*a);      }
fclose(f1);      fclose(f2);
printf("Writing file complete. Press anykey to exit.");
getch();      return 0; }
```

"output2.txt" file

