C Programming

Course info

- Grading – 80% exam, 20% tutorials
- Contact – assafh@ariel.ac.il
- Course supervisor – Dr. Gil Ben Artzi
- Reception hours – for all 3 groups together
  - Sunday - 10:00 → 11:00
  - Monday – 13:30 → 14:30
- 4 exercises
- TA –
  - יבגני
  - חיה / חרות
  - עקיבא

Working environment

- Linux environment
- GCC compiler
- Locally - Ubuntu - open source operating system (download)
- Virtually - c9.io website - Online IDE (integrated development environment)

Algorithm

- Series of command that lead to the challenge solution
- Algorithm characteristics
  - Clear definition
  - Finiteness
  - Input & Output
  - Effectiveness
- Great algorithm is defined by –
  - Thorough Q&A
  - Effectiveness – running time and memory consumption

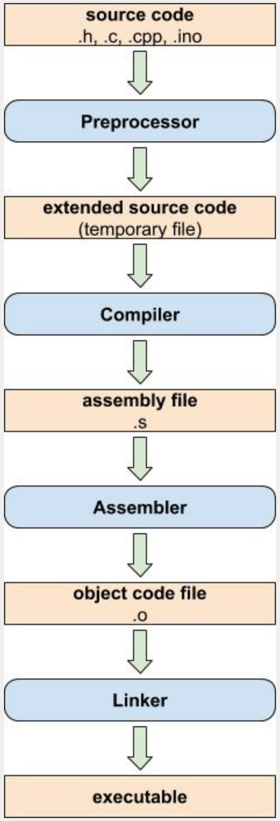
Why C ?

- The most fundamental language that supplies Building block for many other programming languages
- Powerful and efficient – can work on different data types
- Portable – machine independent
- Many built-in functions
- Dynamic memory allocation – during run time and not in advance
- System programming – considers the hardware limitations
- Running speed

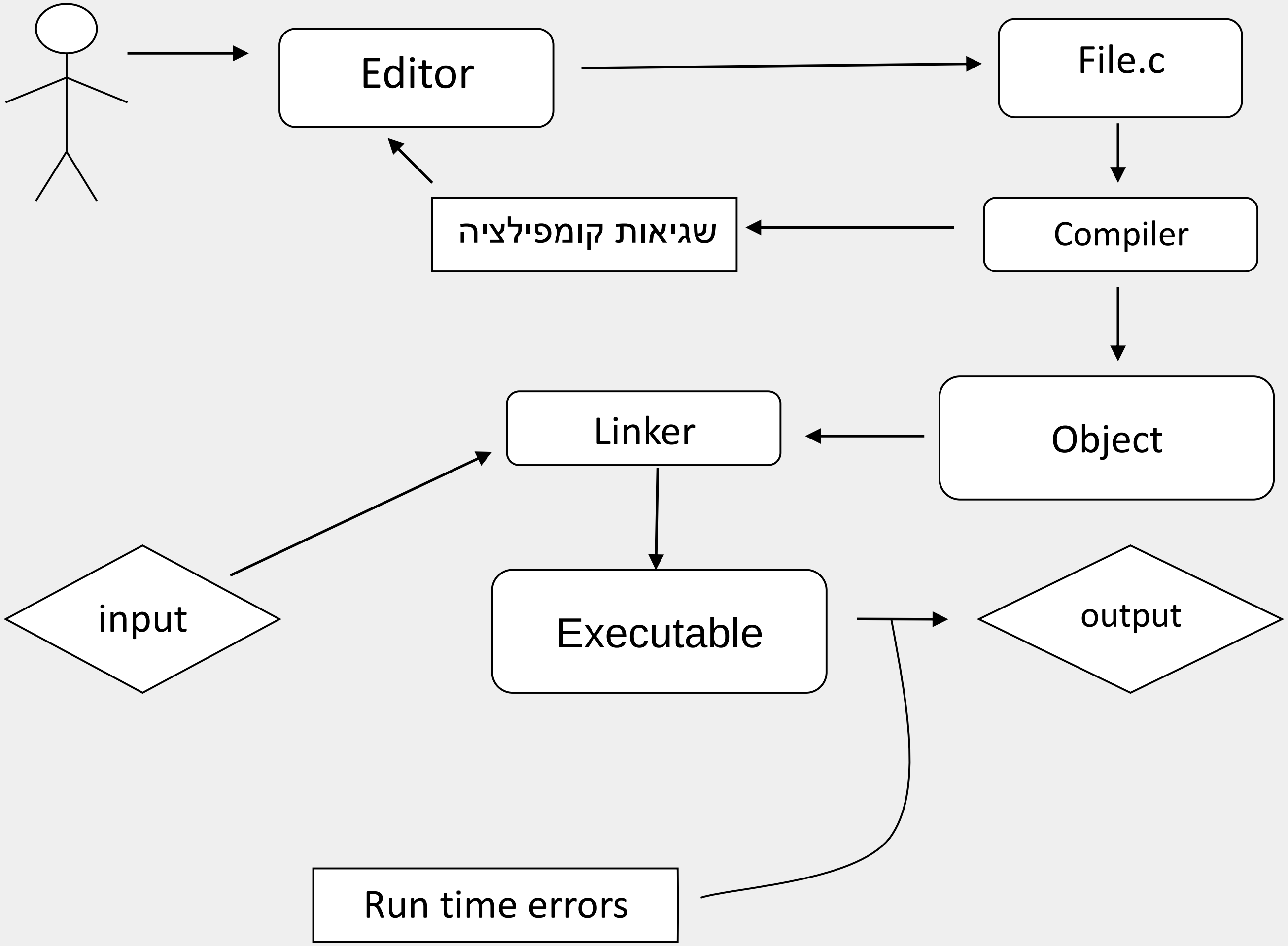
Building a program in C: Preprocessor, Compilation and Linkage

C programming

- Writing the program in the editor
- Generating code source file (*.c)
- Compiling the source file and generating the object code file (*.o)
- Linking the generating file with all other files that are required to activate the program (e.g. aux libraries)
- Executing the file (*.exe)



Toolchain



Editor

File.c

Compiler

שגיאות קומפילציה

Object

Linker

input

Executable

output

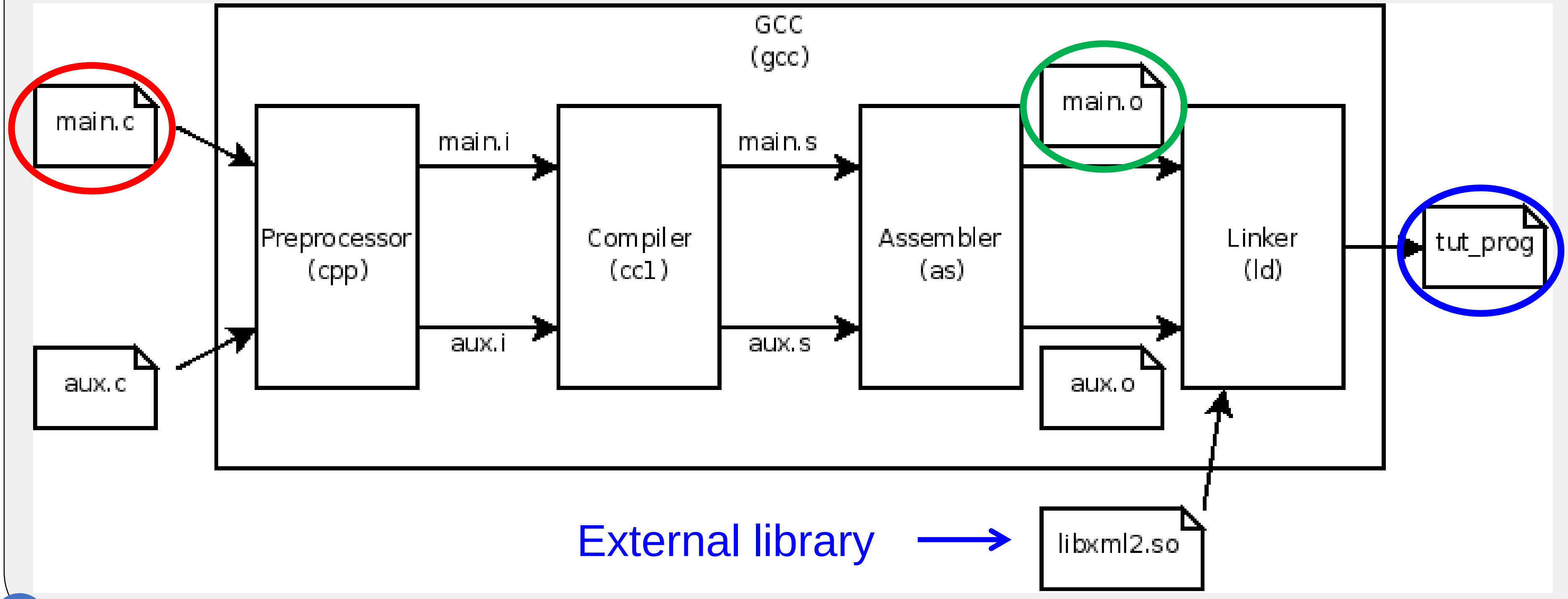
Run time errors

C Compiler

- A compiler is a tool that translates our code from high-level language to a low-level language or machine language (0 and 1), which can be understood by the computer (object file).
- GCC is an example to such a compiler. You can download and install it by going into this link - **http://www.codeblocks.org/downloads/26**

Preprocessing + Compiling

- Creates an object file for each code file (.c -> .o)
- Each .o file contains code of the functions and and global variables
- Linker - **combines object** file with **external references** into an fully executable file



The basic syntax

Basic Syntax

- Comment - // This line is a comment (can be also written as /* …*/
- General program –

```
#include <stdio.h>
void main()
{
        int first;
        printf(“please enter the first number");
}
```

- Details –
  - Almost each program begins with **#include** (preprocessor directive)
  - Many programs include aux libraries as **<stdio.h>** with built-in functions
  - The main block begins with **void main**
  - The program block is defined inside curly brackets - **{….}**
  - **Variables definition** is done at the beginning of the program
  - Each line must be ended with “ **;** ”
  - **White spaces** are not an issue in C

<stdio.h library>

- The library contains standard functions for dealing with Inputs / Outputs
  - putchar (character or variable that contains character) – prints single character on the screen
  - getchar () – get a single input from the keyboard
  - printf () - prints to the standard output (for example screen).
  - scanf () -gets values from the standard input (for example keyboard), and assigns them to variables.
  - flushall () – clean the buffer (temp memory cell) – for example to enable getchar () to get a new character

```
# include <stdio.h>
void main ()
{
		putchar(‘A’);
		printf (“Enter a character\n”);
		ch=getchar();
		putchar(ch);
}
```

printf()

- A built-in function that is part of the <stdio.h> library
- printf() – for standard output
    - printf(output structure, var1, var2 ….)
    - printf(“the number of adults is %d and their total income is %f \n”, adults, income);
    - printf ("z=%d\n", z) –
        - The sequence %d is a special sequence and is not printed!
        - It indicates to printf to print the value of a variable written after the printed string.
        - With many variables: printf ("zeros=%d- ones=%d-letters=%d\n“, a , b, d);

printf()

- printf (“%4d%7d\n”,a,b)

רוחב שדה של
הדפסת
המשתנים

- printf (“%8.2f%10.4g\n”,c,d)

רוחב שדה של
הדפסת
המשתנים כולל
הנקודה
העשרונית

מספר הספרות
המודפסות אחרי
הנקודה
העשרונית

scanf()

- scanf() – another command in <stdio.h> library - for standard input
  - scanf(input structure, var1, var2 ….)
  - scanf(“%f”, &salary)
  - & - must be added just before the variable – only in scanf
  - Forgetting the & can lead to the program shutdown

```
/* Get a length in cm and convert to inches */

#include <stdio.h>

void main()
{
            double cm, inches;

            printf("Please enter length in centimeters: ");
            scanf("%lf",&cm);

            inches = cm / 2.54;
            printf(“%f centimeters are equal to %f inches\n", cm, inches);

}
```

Exercise

Write a program that

- accepts as input -
  - The Dollar-Shekel exchange rate
  - An integer amount of dollars
- and outputs -

  The equivalent amount in Shekels

Solution

```
#include <stdio.h>
void main()  {
        double shekels, xchange;
        int dollars;

        printf("Enter the US$-NIS exchange rate: ");
        scanf("%lf", &xchange);
        printf("Enter the amount of dollars: ");
        scanf("%d", &dollars);
        shekels = dollars * xchange;
        printf("%d dollars = %f shekels\n", dollars, shekels);
}
```

Identifiers in C

- Can be one of the following –
  - Variables
  - Reserved keywords
  - Functions
  - Constants
  - Structures

Reserved keywords

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

Variables

- Variable is an allocated spot in the memory
- The size of the spot depends on the variable type
- Before using a variable – we should first define it
- The definition should be (Var_type)(Var_name)
- Few Var_names can be defined by using comma (“,”) between them

  - int first;
  - char ch;
  - float f1, f2;
  - double d1,d2,d3;

Variable types

Variable name	type	Sign	Values range
Signed char or char	Single character (not only numbers)	%c	-128 to 127
Unsigned char	Single character (not only numbers)	%c	0 to 255
Signed Int or Int (default)	Integers	%d	-32768 to 32767
Unsigned Int	Integers	%d	0 to 65535
Signed Long or Long	Integers in a larger range	%l	-2147483648 to 2147483647
Unsigned Long	Integers in a larger range	%l	0 to 4294967295
Float	Real numbers	%f	6 digits accuracy
Double	Real numbers in a larger range	%d	10 digits accuracy
Long double	Real numbers in a larger range	%ld	10 digits accuracy

Casting

- Using a specific type of variable instead of another type
- We will have to perform casting from one type to another one

```
#include <stdio.h>

void main() {
		int a=1, b=2;

		printf("%d / %d = %d\n", a, b, a/b);
		printf("%d / %d = %f\n", a, b, (float)a / b);
}
```

- What is the output of each printf line ?

Casting

- Using a specific type of variable instead of another type
- We will have to perform casting from one type to another one

```
#include <stdio.h>

void main() {
		int a=1, b=2;

		printf("%d / %d = %d\n", a, b, a/b);  // print 0
		printf("%d / %d = %f\n", a, b, (float)a / b); // print 0.50
}
```

Exercise – what is wrong ?

```
#include <stdio.h>

void main(){

        int a = 10;

        int b = 20;

        printf("The average of %d and %d is %d\n",

                                  a, b, (a + b) * (1 / 2));

}
```

Output:

```
The average of 10 and 20 is 0
```

Will this new one work ?

```
#include <stdio.h>

void main()

{

        int a = 10,  b = 20;

        printf ("The average of %d and %d is %d\n",

                        a, b, (a + b)*(1.0 / 2));

}
```

Ouput:

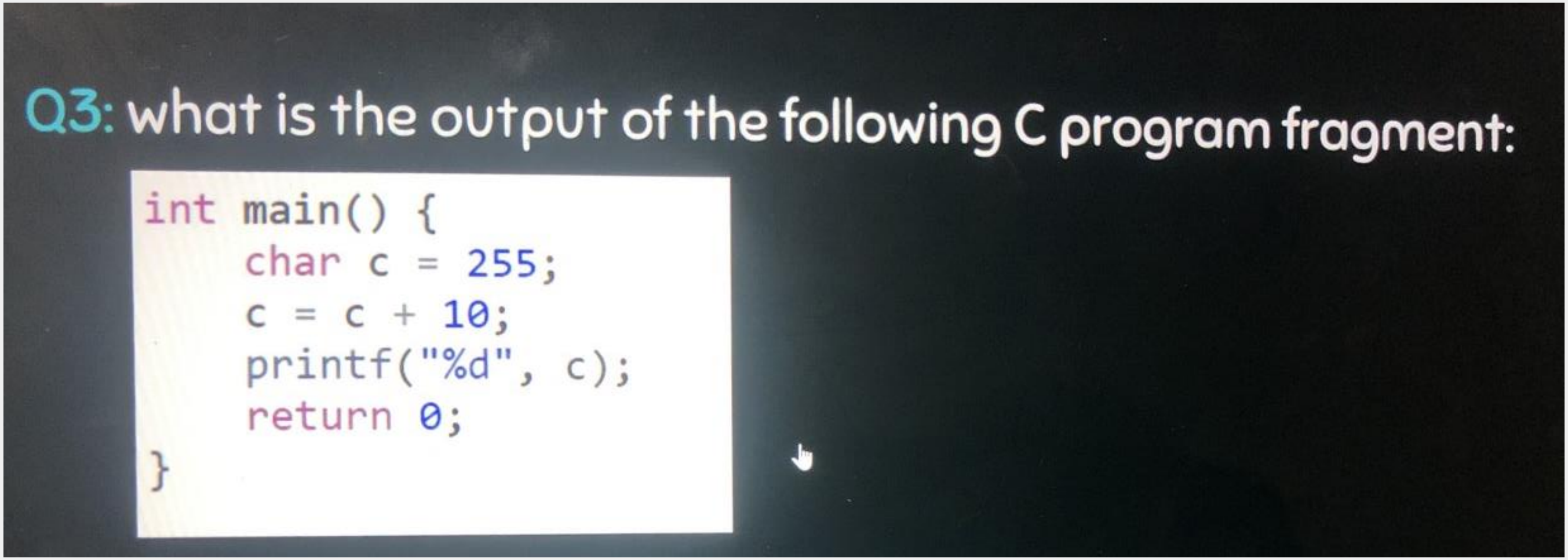
```
The average of 10 and 20 is 0
```

And this one?

```
#include <stdio.h>

void main(void) {

        int a = 10;

        int b = 20;

        printf ("The average of %d and %d is %f\n",

                                    a, b, (a + b)*(1.0 / 2));

}
```

Ouput:

```
The average of 10 and 20 is 15.0000
```



Q3: what is the output of the following C program fragment:

```
int main() {
    char c = 255;
    c = c + 10;
    printf("%d", c);
    return 0;
}
```

Output = 9

Incremental operators

- Used as a short-hand for incrementing (or decrementing) variables.

i++ or ++i ←==→ i = i + 1

i-- or --i ←==→ i = i – 1

i += a ←==→ i = i + a

i -= a ←==→ i = i - a

i *= a ←==→ i = i * a

i /= a ←==→ i = i / a

Prefix ++i and Postfix i++

```
#include <stdio.h>
void main()
{
          int i=5;
          printf("i++=%d\n", i++);
          printf(“++i=%d\n",++i);
}
```

Prefix ++i and Postfix i++

```
#include <stdio.h>
void main()
{
        int i=5;
        printf("i++=%d\n", i++); // print  5
        printf(“++i=%d\n",++i); // print  7
}
```

i++ means "do something with i, and then increment it afterwards".

++i means "increment a first, then do something with the new value".

Variables definition (inside a block)

- In the beginning of the block – before any other command
- These variables will be recognized only within the block
- Variable definition – (type) (name)
  - char ch
  - int num1, num2
- Values assignments –
  - Without assignment the value of the variable is unknown and can have a random value
  - The assignment can be done in the definition line or at any other location in the program

```
void main()
{
        int first=1, second=2, third;
        float salary, price=10.5;
        salary=3000;
        third=55;
}
```

Local / Global Variables

- Local – defined inside a block - will be visible only in block.
- Global - outside all blocks - will be visible everywhere.
- Local has higher priority over global

```
int x=0; // global
void main()
{
   int x=1; //local overwrites global
   {
      int x=2; //local overwrites outer scope
      //x is 2
   }
   //x is 1 again!
}
```

2nd example

```
#include <stdio.h>
void main()
{
   int i;     // declares i as an integer
   int j = 0; // declares j as an integer and initializes it to 0
   // for( initial ; test condition ; update step)
   for( i = 0; i < 10; ++i )
   {
      j += i; // shorthand for j = j + i
      printf("%d %d %f\n", i, j, (i*(i+1))/2);
   }
}
```

Output

```
0 0 0
1 1 1
2 3 3
3 6 6
4 10 10
5 15 15
6 21 21
7 28 28
8 36 36
9 45 45
```

Logical operators

- The value of a variable is “true” if it is non-zero
- The value of a variable is “false” if it is zero

Operator	Meaning
>	Bigger than
<	Smaller than
>=	Bigger or equal to
<=	Smaller or equal to
==	Equal to
!=	Different than
!	Not (for boolean only)

```
int main()
{
  int a = 5;
  while(1)
  {
    if(!(a-3))
    {
      printf("** a=3 **\n");
      break;
    }
    printf("a=%d\n",a--);
  }
  return 0;
}
```

and (&&) ------ OR (||) ------ Not (!)

A	B	A && B
F	F	F
T	F	F
F	T	F
T	T	T

A	B	A || B
F	F	F
T	F	T
F	T	T
T	T	T

A	! A
F	T
T	F

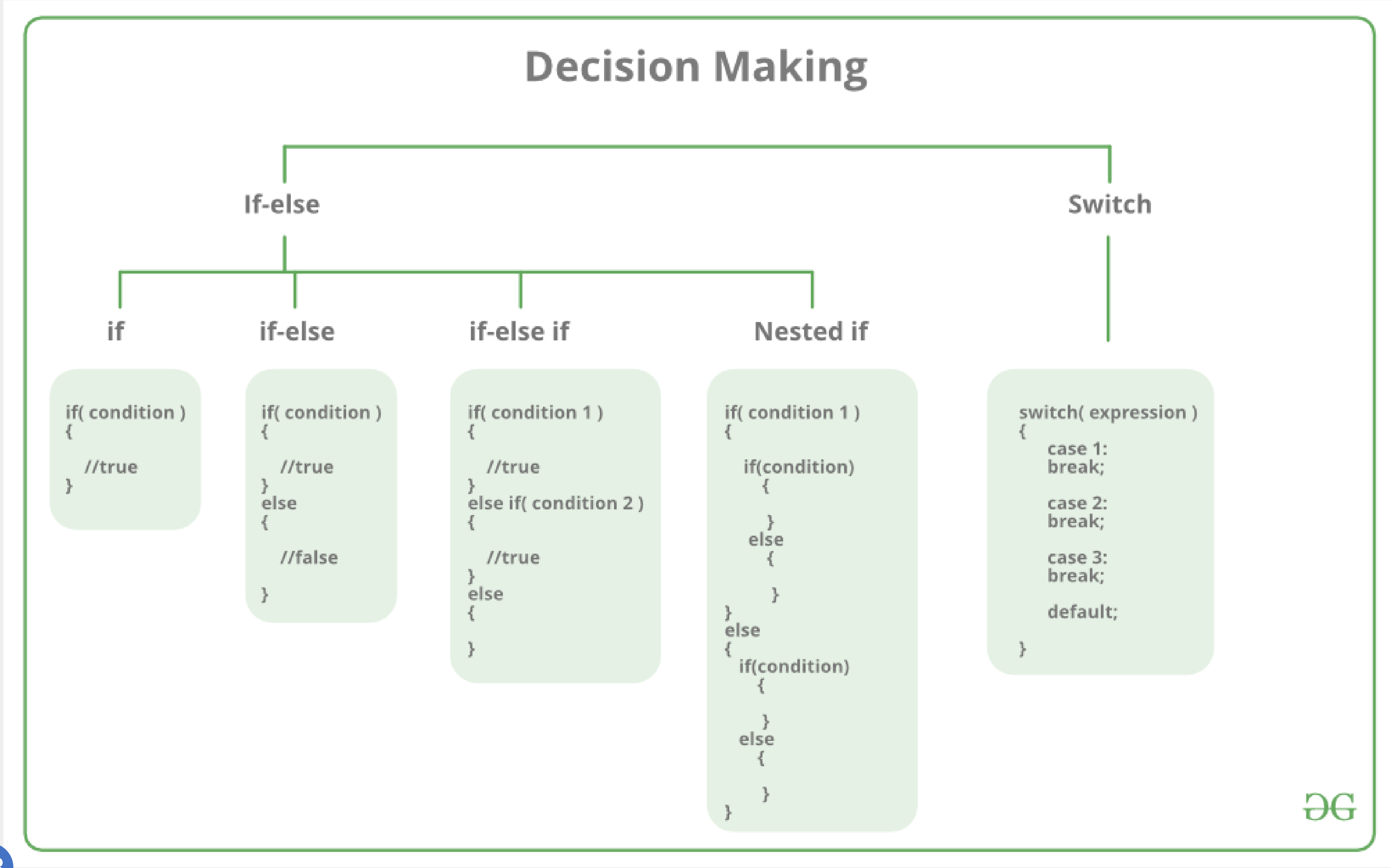
Conditions

-
  ```
  if (condition)
                  //statement or block ;
      else if (condition)
                  //statement or block ;
      else (condition)
                  //statement or block ;
  ```

-
  ```
  Switch (variable or expression) {
      case  option1:  ……
      case  option2:  ……
                  break;
      case  option3:  ……
      }
  ```

“break” – to terminate the loop. Without “break” the loop will continue

Conditions



Decision Making

If-else

Switch

if

```
if( condition )
{
   //true
}
```

if-else

```
if( condition )
{
   //true
}
else
{
   //false

}
```

if-else if

```
if( condition 1 )
{
   //true
}
else if( condition 2 )
{
   //true
}
else
{

}
```

Nested if

```
if( condition 1 )
{
   if(condition)
     {

     }
    else
     {

     }
}
else
{
  if(condition)
    {

    }
  else
    {

    }
}
```

switch

```
switch( expression )
{
    case 1:
    break;

    case 2:
    break;

    case 3:
    break;

    default;

}
```

If Condition - example

- Please write a program that gets a number and display its absolute value (5 min)

If Condition - solution

```
/* This program displays the absolute value of a number given by
the user */
#include <stdio.h>
void main() {
        double num;
        printf("Please enter a real number: ");
        scanf("%lf", &num);
        if (num<0)
                num = -num;
        printf("The absolute value is %g\n", num);
}
```

If-else Condition

- Please write a program that gets 2 numbers and display the smaller one (5 min)

If-else Condition - solution

```
int first, second, min;
if (first < second) {
   min = first;
   printf ("The first number is smaller than the second.\n");
}
else {
   min = second;
   printf ("The second number is smaller than the first\n");
}
printf("The smaller number is equal to %d\n", min);
```

?: operator

- $expr_1$ **?** $expr_2$ **:** $expr_3$
- If expr1 is true (non-zero), expr2 is evaluated. Otherwise, expr3 is evaluated
- Please write the same program that we wrote before (finding the smaller number out of 2 numbers), but this time with the ?: operator

?: operator - solution

```
#include <stdio.h>
void main() {
        int i, j, min;

        printf("Please enter two numbers: ");
        scanf("%d%d", &i, &j);

        min =  i<j  ?  i  :  j;
        printf("The minimum between %d and %d is %d\n", i, j, min);

}
```

Switch condition

- A multi-way conditional statement similar to the if-else if-else … "statement"
- Structure -

```
switch (expression) {
  case const-expr:
                 statements
  case const-expr:
                 statements
  …
  default:
                 statements
}
```

Example

- Without the “break” the program will continue running

```
switch (grade/10) {
        case 9:
                printf ("A\n");
                break;
        case 8:
                printf ("B\n");
                break;
        case 7:
                printf ("C\n");
                break;
        case 6:
                printf ("D\n");
                break;
        default:
                printf ("F\n");
}
```

while loops

-
  ```
  while (condition)
      //statement or block
  ```

Why do we need to define x if we ask the user to define it later ?

```
#include <stdio.h>
void main ()
{
        int x;
        x=0;
        while (x<1 || x>10)  // value is outside the range
        {
                printf(“Enter a number from 1 to 10”);
                scanf(“%d”,&x);
        }
}
```

do while loops

-
  ```
  do
          //statement or block
  while (condition)
  ```

```
#include <stdio.h>
void main ()
{
        int x;
        do      {
                printf("Enter a number from 1 to 10");
                scanf("%d",&x);
                }
        while (x<1 || x>10) { // if the value is not in the range
        …..
        }
}
```

- What is the difference between while and do - - while ?

In do-while the condition is checked at the end of the loop – therefore the loop will run at least once.

for loops

- for (x=0,y=0; x<10 && y<5; x++,x+=2)

  //statement or block

- for loop consists of 3 parts –
  - Parameters initialization – x=0,y=0
  - Condition - x<10 && y<5
  - Parameters progress - x++,x+=2

```
#include <stdio.h>
void main ()
{
        int x, y;
        for (x=0,y=0; x<10 && y<5; x++,x+=2)
                printf (“%d, %d”, x,y)
}
```

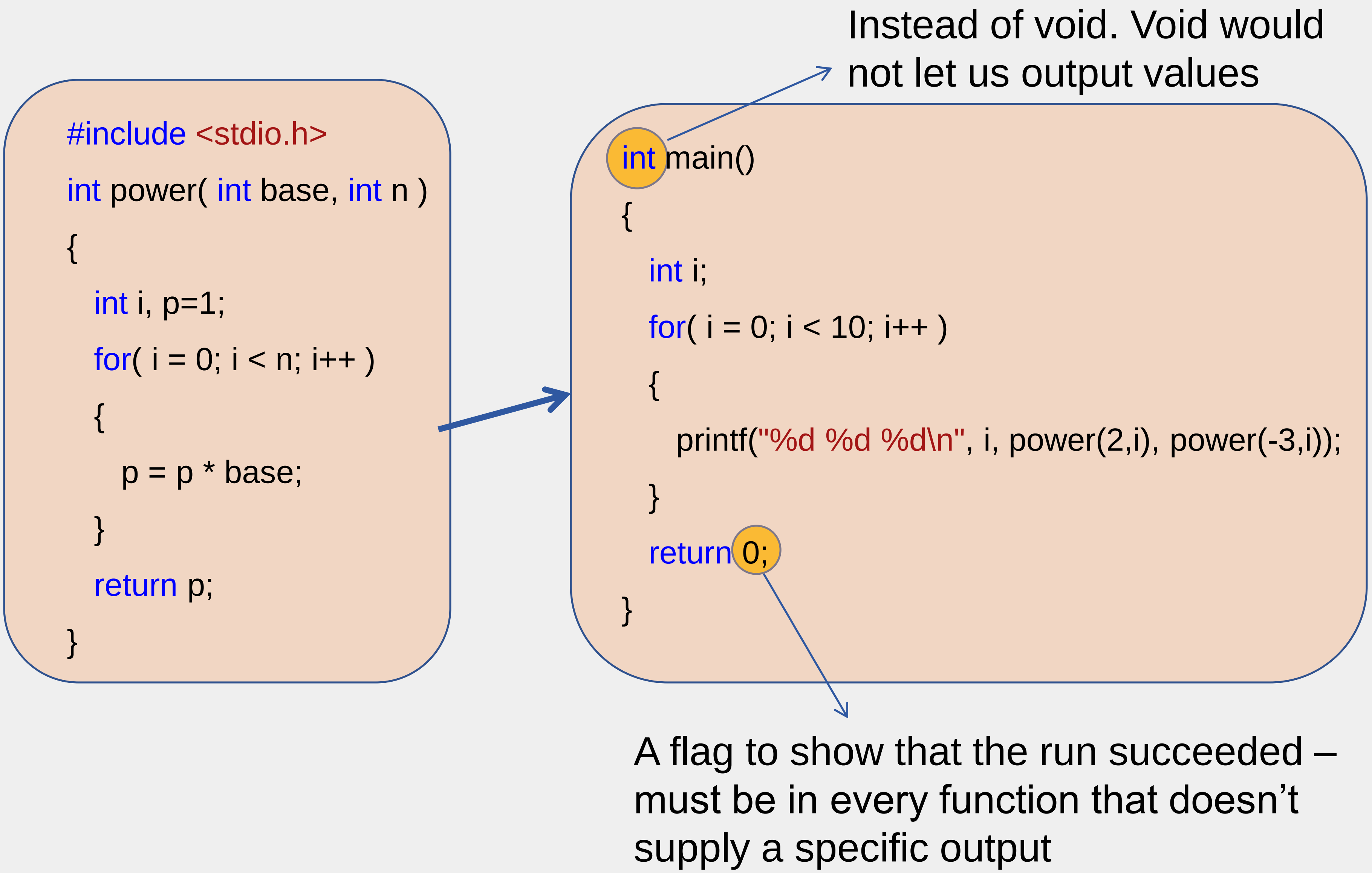
C Functions

- Output type (e.g. int) - functions that outputs values / statements
- void – functions that are not required to return anything to the caller of the function

```
int func(int a, int b)
{
 …
 return a;
}
```

```
void func(int a, int b)
{
      …
     …
 }
```

Internal and external functions



```
#include <stdio.h>
int power( int base, int n )
{
   int i, p=1;
   for( i = 0; i < n; i++ )
   {
      p = p * base;
   }
   return p;
}
```

```
int main()
{
   int i;
   for( i = 0; i < 10; i++ )
   {
      printf("%d %d %d\n", i, power(2,i), power(-3,i));
   }
   return 0;
}
```

Instead of void. Void would not let us output values

A flag to show that the run succeeded – must be in every function that doesn’t supply a specific output

Functions Declaration

```
void funcA()
{
   ...
}
void funcB()
{
   funcA();
}
void funcC()
{
   funcB();
   funcA();
   funcB();
}
```

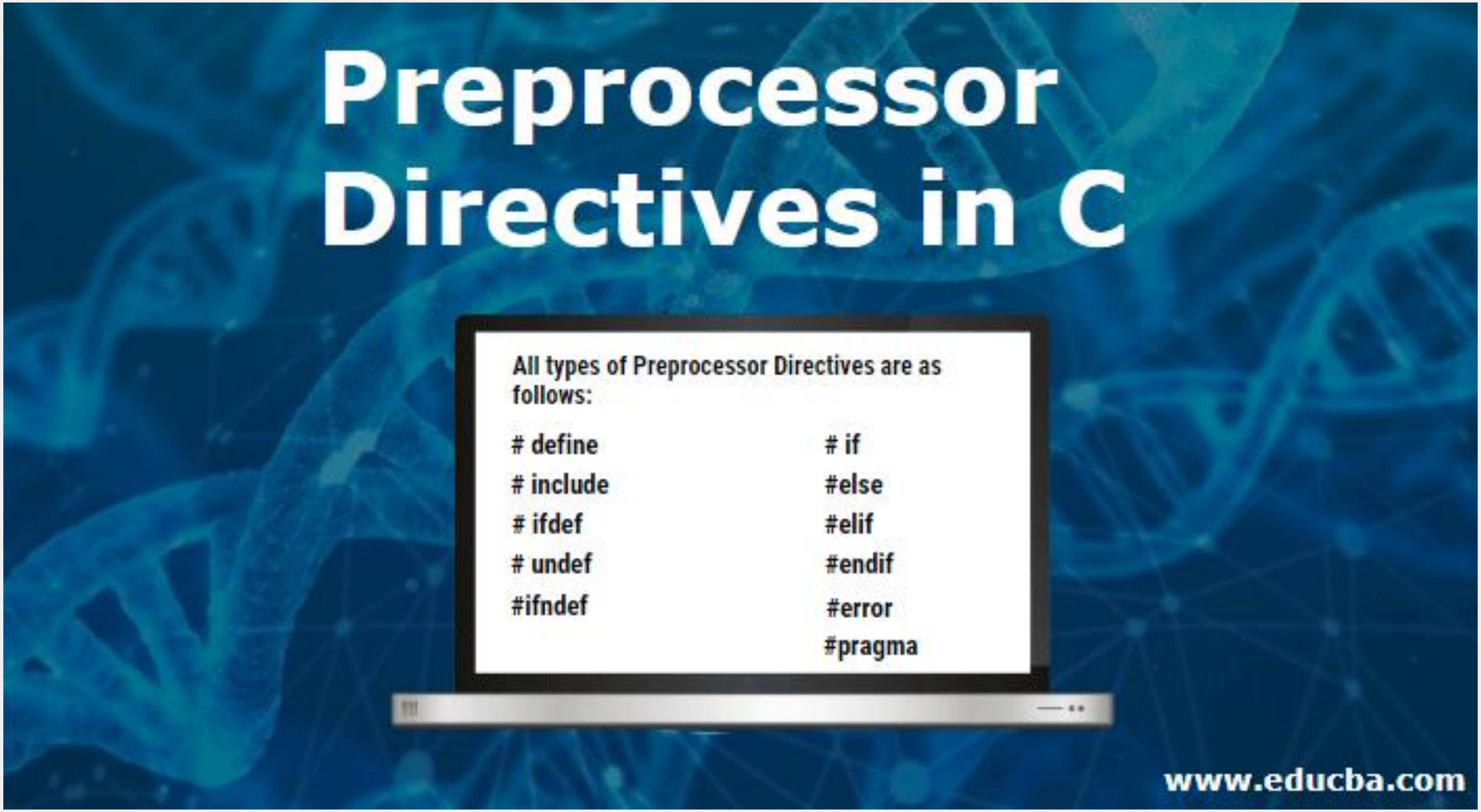
```
void funcA()
{
   ...
}
void funcB()
{
   funcC();
}
void funcC()
{
   funcB();
}
```

“Rule 1”: A function “knows” only functions which were declared above it.

Error:
funcC is not known yet.

C Preprocessor

- Begins with #
- The C preprocessor is a program that processes our source program before it is passed to the compiler.
- Preprocessor is a separate program which transforms C source code containing preprocessor directives into source code with the directives removed.
- **gcc -E filename** will show the preprocessed file.
- Preprocessor commands are known as directives.
- Possible actions –
  - Inclusion of other files - #**include** stdio.h
  - Definition of symbols and macros - #**define** PI 3.14
  - Performs conditional compilations - #if, #elsif …



Preprocessor Directives in C

All types of Preprocessor Directives are as follows:

define
include
ifdef
undef
#ifndef

if
#else
#elif
#endif
#error
#pragma

www.educba.com

#include directive for the preprocessor

- #include <stdio.h> - standard library directory with standard functions as printf (), scanf (), putchar(), getchar()….
- #include <conio.h> - library that includes functions that relate to the screen such as clrscr(), gotoxy() – function that can go to a specific line
- #include "foo.h" - any function that we build by ourselves and must be called from **current directory**

#define

Directive

```
#define foo 1
int x = foo;
```

is equivalent to

```
int x = 1;
```

With arguments

```
#define square(x) x*x
b = square (a);
```

is the same as

```
b = a*a;
```

#define – cautions

```
#define Square(x) (x*(x))

void main()
{
        int x = 5;
        printf(“%d”, Square(x+3));
}
```

What is the answer of this code ?

5+3*(5+3) =29

```
#define Square(x) ((x)*(x))

void main()
{
        int x = 5;
        printf(“%d”, Square(x+3));
}
```

What is the answer of this code ?

(5+3)*(5+3) =64

#define

- Multi-line - all preprocessor directive effect one line. To insert a line-break, use “\”:

```
#define x (5 + \
        5)
// x == 10 !
```

#define – cautions

```
#define SQUARE (x) ((x)*(x))
b = SQUARE (a+1);
c = SQUARE (a++);
```

Is it what we intended?

```
b = ((a+1)*((a+1));
//Now ok
c =  ((a++)*(a++));
//Did not help: c = a*a; a+=2;
```

#define

`#define` directive should be used with caution!

Alternative to macros:

- **Constants**

```
enum { FOO = 1 };
```

or

```
const int FOO = 1;
```

- **Functions** – inline functions (C99,C++)

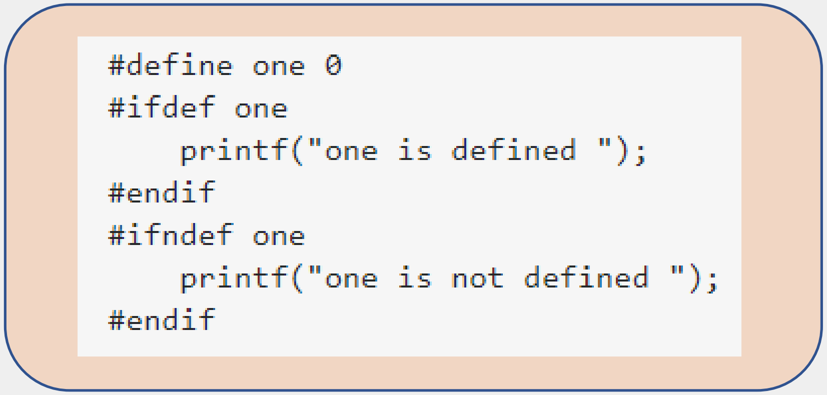
#if directive

- The #if directive, with the #elif, #else, and #endif directives, controls compilation of portions of a source file.
- #if defined(var) – will be compiled only if *var* will be predefined

```
#if defined(Credit)     // Credit is different than credit
    credit();
#elif defined(Debit)    // Debit is different than debit
    debit();
#else
    printerror();
#endif
```

#ifdef / #ifndef directives

- Text inside an #ifdef / #endif or #ifndef / #endif *pair* will be left in or removed by the pre-processor depending on the condition.
- #ifdef means "if the following is defined" while #ifndef means "if the following is *not* defined".



```
#define one 0
#ifdef one
    printf("one is defined ");
#endif
#ifndef one
    printf("one is not defined ");
#endif
```

This is equivalent to printf (“one is defined”);

#ifndef – header safety

Complex.h (revised):

```
#ifndef COMPLEX_H
#define COMPLEX_H
struct Complex
{
...
}
#endif
```

Main.c:

```
#include "MyStuff.h"
#include "Complex.h" // no error this time
```

#pragma once

- The compiler includes the header file only once, when compiling a source code file.
- It can reduce build times, as the compiler won't open and read the file again after the first #include.
- It also helps to prevent violations of the ”*one definition” rule*, the requirement that all templates, types, functions, and objects have no more than one definition in your code.

```
// header.h
#pragma once
#ifndef HEADER_H_     // equivalently, #if !defined HEADER_H_
#define HEADER_H_
// Code placed here is included only once per translation unit
#endif // HEADER_H_
```

Debugging

assert.h

- Evaluates an expression and, when the result is **false**, prints a diagnostic message and aborts the program.
- Parameters –
  - *Expression* - scalar expression (including pointer expressions) that evaluates to nonzero (**true**) or 0 (**false**).
  - *Message* - the message to display.
  - *Filename* - the name of the source file the assertion failed in.
  - *Line* - the line number in the source file of the failed assertion

```
assert(expression);
void _assert(
   char const* message,
   char const* filename,
   unsigned line
);
```

assert.h

```
#include <stdio.h>
#include <assert.h>
#include <string.h>

void analyze_string(char *string);   // declaration

int main( void )
{
   char  test1[] = "abc", *test2 = NULL, test3[] = "";

   printf ( "Analyzing string '%s'\n", test1 );
   analyze_string( test1 );
   printf ( "Analyzing string '%s'\n", test2 );
   analyze_string( test2 );
   printf ( "Analyzing string '%s'\n", test3 );
   analyze_string( test3 );
}

// Tests a string to see if it is NULL, empty, or longer than 0 characters.
void analyze_string( char * string )
{
   assert( string != NULL );        // Cannot be NULL
   assert( *string != '\0' );       // Cannot be empty
   assert( strlen( string ) > 2 );  // Length must exceed 2
}
```

Output –

```
Analyzing string 'abc'
Assertion failed: string != NULL, file crt_assert.c, line 25
```

Using a function inside the main will enforce us to do one of the 2 options –

1) Declaration of the function before the main and define it after the main (as it appears here)
2) Just define before the main and that’s it