CS 332/532 Systems Programming

Lecture 2

- Introduction to C Programming -

Professor : Mahmut Unan – UAB CS

Agenda

- History of the C programming language
- Hello World
- Compilation
- Variables & Data Types
- const, define, extern…
- printf
- scanf

History of C

- It was developed at Bell Labs in the early 1970s
  - Same time as *Unix* operating system was developed (in assembly)

- Dennis Richie decided to rewrite *Unix* code in another language

The ANSI Standard

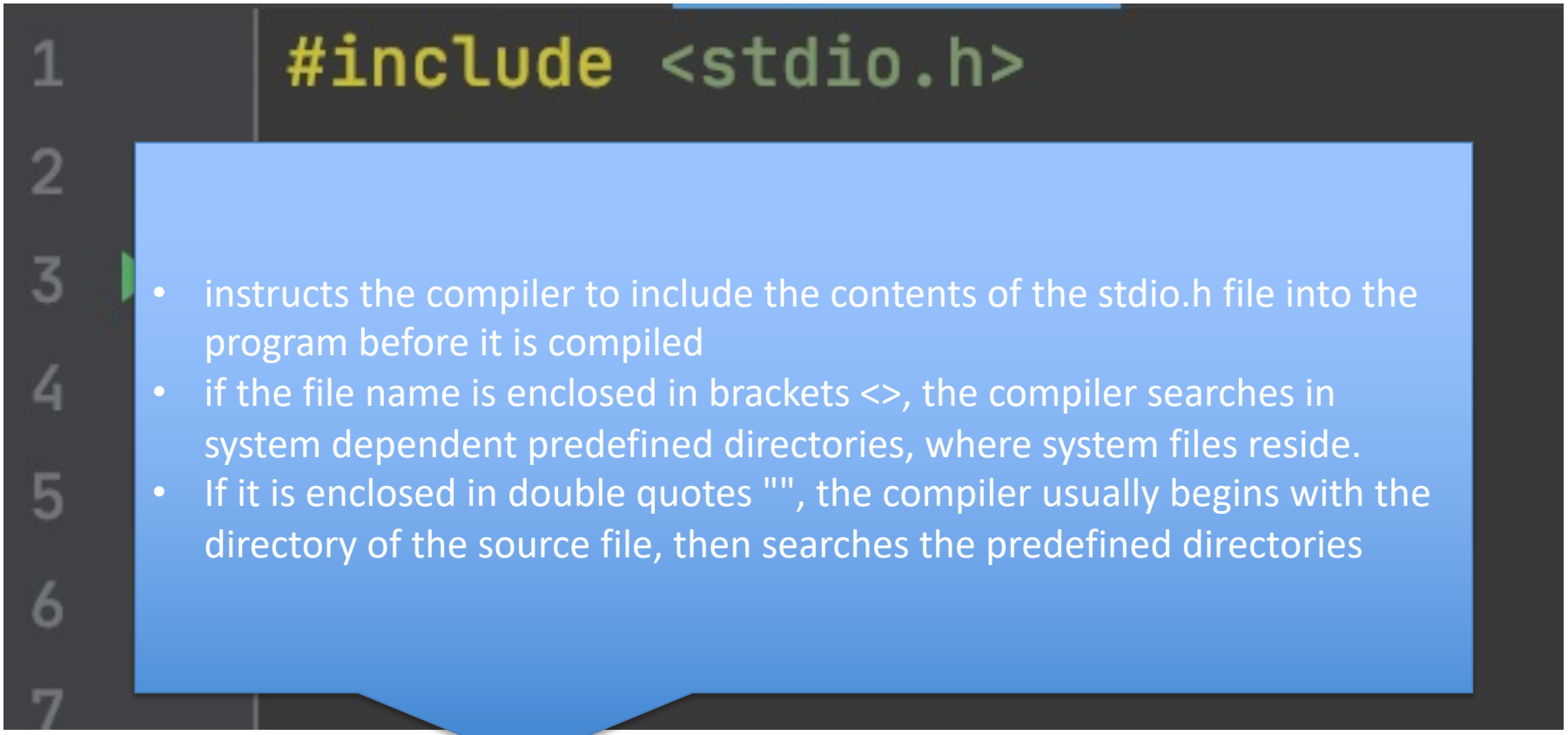
- The rapid expansion of the C language
  - Many companies developed their own C compilers
- In 1983, the American National Standard Institute (ANSI) began the development of the C standard that was completed and formally approved, in 1989, as ANSI C or Standard C
    - C89
    - C90
    - C95
    - C99
    - C11
    - C18

Let’s get started

- VSCode
- CLion(Jetbrains)
- Code::Blocks
- Eclipse
- NetBeans
- CodeLite
- …….

Hello World !

#include


1
#include <stdio.h>
2
3
4
5
6
7
• instructs the compiler to include the contents of the stdio.h file into the program before it is compiled
• if the file name is enclosed in brackets <>, the compiler searches in system dependent predefined directories, where system files reside.
• If it is enclosed in double quotes "", the compiler usually begins with the directory of the source file, then searches the predefined directories

The `main()` function

```
3  int main() {
4      printf("Hello World!");
5      return 0;
6  }
7
```

- Every C program must contain a function named main()
- main function will be automatically called when the program runs
- int indicates that the main function will return an integer
  - How about void?

Comments

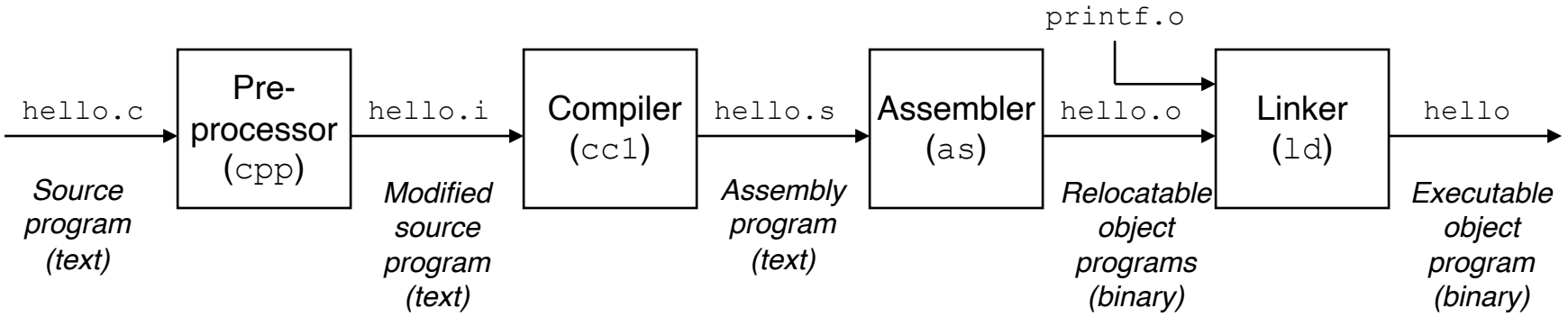
```c
#include <stdio.h>
/* This program calls printf() to display a message on the screen. */
int main(void)
{
        printf("Hey Ho, Let's Go\n");
        return 0;
}
```

Nested comments are not allowed.
For example, the following code is illegal,
and the compiler will raise an error message:

```c
/*
/* Another comment. */
*/
```

Compilation

- `unix> gcc –o hello hello.c`



Compilation System

Compiler Options

- You can compile your C program with various levels of optimization turned on (*e.g., -O, -O3, -Ofast*). Here are some useful/popular compiler and optimization options:
- **The most basic form:** `gcc hello.c` executes the complete compilation process and outputs an executable with name a.out
- **Use option –o:** `gcc hello.c -o hello` produces an output file with name 'hello'.
- **Use option –Wall:** `gcc -Wall hello.c -o hello` enables all the warnings in GCC.
- **Use option –E:** `gcc -E hello.c > hello.i` produces the output of preprocessing stage
- **Use option –S:** `gcc -S hello.c > hello.S` produces only the assembly code
- **Use option –C:** `gcc -C hello.c` produces only the compiled code (without linking)
- **Use option –O:** `gcc -O hello.c` sets the compiler's optimization level.

Running Hello Object File

- Running hello object file on the shell

```
unix> ./hello
hello, world
unix>
```

Variables

- A *variable* in C is a memory location with a given name. The value of a variable is the content of its memory location. A program may use the name of a variable to access its value.
- Here are some basic rules for naming variables. These rules also apply for function names. Be sure to follow them or your code won't compile:
  - The name can contain letters, digits, and *underscore characters* _.
  - The name must begin with either a letter or the underscore character.
  - C is *case sensitive*, meaning that it distinguishes between uppercase and lowercase letters.

Variables / 2

- The following keywords cannot be used as variable names because they have special significance to the C compiler.

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	

Declaring Variables

- **`data_type`** `name_of_variable;`

C Data Types

Data Type	Usual Size (bytes)	Range of Values(min–max)
`char`	1	−128…127
`short int`	2	−32.768…32.767
`int`	4	−2.147.483.648…2.147.483.647
`long int`	4	−2.147.483.648…2.147.483.647
`float`	4	Lowest positive value: $1.17*10^{-38}$ Highest positive value: $3.4*10^{38}$
`double`	8	Lowest positive value: $2.2*10^{-308}$ Highest positive value: $1.8*10^{308}$
`long double`	8, 10, 12, 16	
`unsigned char`	1	0…255
`unsigned short int`	2	0…65535
`unsigned int`	4	0…4.294.967.295
`unsigned long int`	4	0…4.294.967.295

What is the output ?

```c
#include <stdio.h>

int main() {

    float a = 3.1;
    if (a == 3.1)
        printf("Yes\n");
    else
        printf("No\n");
    printf("%.9f\n", a - 3.1);
    return 0;
}
```

What is the output ?

```c
#include <stdio.h>

int main() {

    float a = 3.1;
    if (a == 3.1)
        printf("Yes\n");
    else
        printf("No\n");
    printf("%.9f\n", a - 3.1);
    return 0;
}
```

```
No
-0.000000095
```

extern keyword

- The `extern` keyword extends the function's visibility to the whole program, the function can be used (called) anywhere in any of the files of the whole program, provided those files contain a declaration of the function

- For more info: https://www.geeksforgeeks.org/understanding-extern-keyword-in-c/

```c
#include <stdio.h>
// Declarations
extern int a, b;
extern int c;
extern float f;
extern char s[];
extern double d;
int somefunction();

```

```c
int main () {

    /* variable definition: */
    int a, b;
    int c;
    float f;
    char s[15]="sample string";
    double d;

    /* actual initialization */
    a = 10;
    b = 20;
    d=25.0;

    c = a + b;
    printf("value of c : %d \n", c);

    f = 70.0/3.0;
    printf("value of f : %f \n", f);

    d= 50.0/d;
    printf("value of d : %f \n", d);

    printf("value of s : %s \n", s);

    int functionResult = somefunction();
    printf("value of function call : %d\n",functionResult);
    return 0;
}
```

```
38  int somefunction(){
39      int a=5;
40      int b=7;
41      return (a+b);
42
43  }
```

```
value of c : 30
value of f : 23.333334
value of d : 2.000000
value of s : sample string
value of function call : 12
```

Arithmetic Conversions

```
char c;
short s;
int i;
unsigned int u;
float f;
double d;
long double ld;
i = i+c; /* c is converted to int. */
i = i+s; /* s is converted to int. */
u = u+i; /* i is converted to unsigned int. */
f = f+i; /* i is converted to float. */
f = f+u; /* u is converted to float. */
d = d+f; /* f is converted to double. */
ld = ld+d; /* d is converted to long double. */
```

const

- A variable whose value cannot change during the execution of the program is called *constant*.

```
const int a = 10;
const double PI = 3.14;
int somefunction(const int *data, size_t size);
```

#define

- The `#define` directive is used to define a macro. In most cases, a macro is a name that represents a numerical value. To define such a macro, we write:

```
#define macro_name value
```

For example

```
#define NUM 100
```

```c
#include <stdio.h>
#define NUM 100
int main(void)
{
        int a, b, c;
        a = 20 - NUM;
        b = 20 + NUM;
        c = 3 * NUM;
        return 0;
}
```

`printf()`

- The `printf()` function is used to display data to stdout (*st*andar*d* *out*put stream).
- The `printf()` function accepts a variable list of parameters.
  - The first argument is a format string, that is, a sequence of characters enclosed in double quotes, which determines the output format.
- The format string may contain escape sequences and conversion specifications.

Escape Sequences

- Escape sequences are used to represent nonprintable characters or characters that have a special meaning to the compiler. An escape sequence consists of a backslash (\) followed by a character.

Escape Sequences

`\a`	Audible alert.
`\b`	Backspace.
`\f`	Form feed.
`\n`	New line.
`\r`	Carriage return.
`\t`	Horizontal tab.
`\v`	Vertical tab.
`\\`	Backslash.
`\'`	Single quote.
`\"`	Double quote.
`\?`	Question mark.

Conversion Specifications

- A conversion specification begins with the % character, and it is followed by one or more characters with special significance. In its simplest form, the % is followed by one of the conversion specifiers below

Conversion Specifier	Meaning
c	Display the character that corresponds to an unsigned integer value.
d ή i	Display a signed integer.
u	Display an unsigned integer.
f	Display a floating-point number. The default precision is six digits.
s	Display a sequence of characters.
e, E	Display a floating-point number in scientific notation. The exponent is preceded by the chosen specifier e or E.
g, G	%e or %E form is selected if the exponent is less than −4 or greater than or equal to the precision. Otherwise, the %f form is used.
P	Display the value of a pointer variable.
x, X	Display an unsigned integer in hex form; %x displays lowercase letters (a–f), while %X displays uppercase letters (A–F).
O	Display an unsigned integer in octal.
n	Nothing is displayed. The matching argument must be a pointer to integer; the number of characters printed so far will be stored in that integer.
%	Display the character %.

```c
#include <stdio.h>
int main(void)
{
        int len;
        printf("%c\n", 'w');
        printf("%d\n", -100);
        printf("%f\n", 1.56);
        printf("%s\n", "some text");
        printf("%e\n", 100.25);
        printf("%g\n", 0.0000123);
        printf("%X\n", 15);
        printf("%o\n", 14);
        printf("test%n\n", &len);
        printf("%d%%\n", 100);
        return 0;
}
```

The program outputs:

`w` (the character constant must be enclosed in single quotes).
`-100`
`1.560000`
`some text` (the string literal must be enclosed in double quotes).
`1.002500e+002` (equivalent to $1.0025 \times 10^2 = 1.0025 \times 100 = 100.25$).
`1.23e-005` (because the exponent is less than `-4`, the number is displayed in scientific form).
`F` (the number `15` is equivalent to `F` in hex).
`16` (the number `14` is equivalent to `16` in octal).
`test` (since four characters have been printed before `%n` is met, the value `4` is stored into `len`.
`100%` (to display the `%` character, we must write it twice).

Precision

```c
#include <stdio.h>
int main(void)
{
        float a = 1.2365;
        printf("%f\n", a);
        printf("%.2f\n", a);
        printf("%.*f\n", 3, a);
        printf("%.f\n", a);
        return 0;
}
```

```
1.236500
1.24
1.237
1
```

scanf()

- The scanf() function is used to read data from stdin (*st*andar*d* *in*put stream) and store that data in program variables.

- The scanf() function accepts a variable list of parameters. The first is a format string similar to that of printf(), followed by the memory addresses of the variables in which the input data will be stored.

- Typically, the format string contains only conversion specifiers. The conversion characters used in scanf() are the same as those used in printf().

scanf()

```
int i;
float j;
scanf("%d%f", &i, &j);

char str[100];
scanf("%s", str);
```

```c
#include <stdio.h>
int main(void)
{
    char ch;
    int i;
    float f;
    printf("Enter character, int and float: ");
    scanf("%c%d%f", &ch, &i, &f);
    printf("\nC:%c\tI:%d\tF:%f\n", ch, i, f);
    return 0;
}
```

```
Enter character, int and float: s 17 22.6

C:s     I:17    F:22.600000
```

References

- https://www.tutorialspoint.com/cprogramming/c_constants.htm
- C From Theory to Practice - 2nd edition, Nikolaos D. Tselikas and George S. Tselikis

Data Types in C

Integer Types

The following table provides the details of standard integer types with their storage sizes and value ranges –

Type	Storage size	Value range
char	1 byte	-128 to 127 or 0 to 255
unsigned char	1 byte	0 to 255
signed char	1 byte	-128 to 127
int	2 or 4 bytes	-32,768 to 32,767 or -2,147,483,648 to 2,147,483,647
unsigned int	2 or 4 bytes	0 to 65,535 or 0 to 4,294,967,295
short	2 bytes	-32,768 to 32,767
unsigned short	2 bytes	0 to 65,535
long	4 bytes	-2,147,483,648 to 2,147,483,647
unsigned long	4 bytes	0 to 4,294,967,295

https://www.tutorialspoint.com/cprogramming/c_data_types.htm

Floating-Point Types

The following table provide the details of standard floating-point types with storage sizes and value ranges and their precision –

Type	Storage size	Value range	Precision
float	4 byte	1.2E-38 to 3.4E+38	6 decimal places
double	8 byte	2.3E-308 to 1.7E+308	15 decimal places
long double	10 byte	3.4E-4932 to 1.1E+4932	19 decimal places

The header file float.h defines macros that allow you to use these values and other details about the binary representation of real numbers in your programs. The following example prints the storage space taken by a float type and its range values –

https://www.tutorialspoint.com/cprogramming/c_data_types.htm

C - Type Casting

- https://www.tutorialspoint.com/tpcg.php?p=LlUZrX
- https://www.tutorialspoint.com/tpcg.php?p=VkqrXY
- https://www.tutorialspoint.com/tpcg.php?p=MImFCX

https://https://www.tutorialspoint.com/cprogramming/c_type_casting.htm