# Fundamentals of Programming

## Lecture 3

**Chamila Karunatilake**
Department of Information and Communication Technology
Faculty of Technology
University of Sri Jayewardenepura
***chamilakarunatilake@sjp.ac.lk***

## Data Representation in C

## Lecture Outline

- Variables
- Constants
- Data Types

## Data Representations in C

- The purpose of a computer program is to **manipulate data** to **perform a specific task** (or a set of tasks).
- This data can be varied and the internal representation of that data in the program could be different depending on the purpose of that data.
- There are mainly two types of representations in C programming language (also in  many other programming languages).
- They are **Variables** and **Constants**.

## Variables

- In a mathematical function, variable is a symbol, commonly an alphabetic character, that represents a **changeable value**.
- Similarly, a variable in computer programming can be considered as a **named memory location** reserved for one data(or collection of data) that we wish to use in our program.
- The **value** of the variable **can be changed**, hence it is called a variable. It can hold **one value at a time**. If we add a new value to it, the old value will be replaced by the new.
- The size of the reserved space in the memory for the variable depends on **the type of the data** it is supposed to hold. Therefore, when the variable is created, we have to mention the data type as well.

## Variables

C0313
C0317
C0321
C0325
C0329
C0333
C0314
C0318
C0322
C0326
C0330
C0334
C0327
C0331
C0335

## Variable Declaration

- When working with variables, first, a space for our variable should be reserved. We ask the operating system for a piece of memory.
- The size of the piece of memory we need depends on the size of the data that we intend to store there, and the size of the data depends on the type of the data, hence, when the memory piece is reserved, we must mention the data type as well.
- A name can be given to this piece of memory.  The purpose of naming the variable is that we can refer the data stored in that variable inside our program using its name. There are several rules of naming variables.
- This process is called variable declaration.

Examples:

```
int age;
float salary ;
```

## Variable Assignment

- Even the variable is declared, still it does not contain any value. It is still empty.
- The process of adding a value to it is called variable assignment (or assigning a value to a variable)
- We use equal sign(=) to assign a value to a variable. Therefore, in C programming language, = sign is called assignment operator.

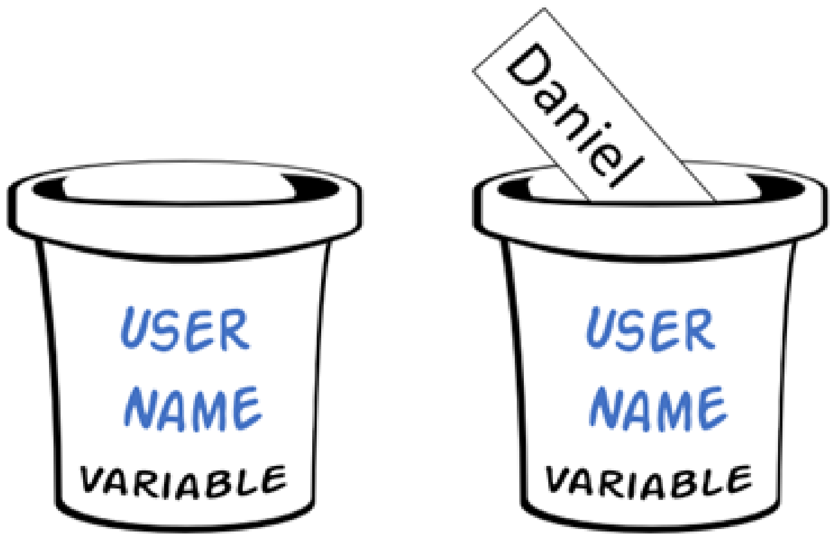
Examples:

```
age = 25;
salary = 30000.0;
```

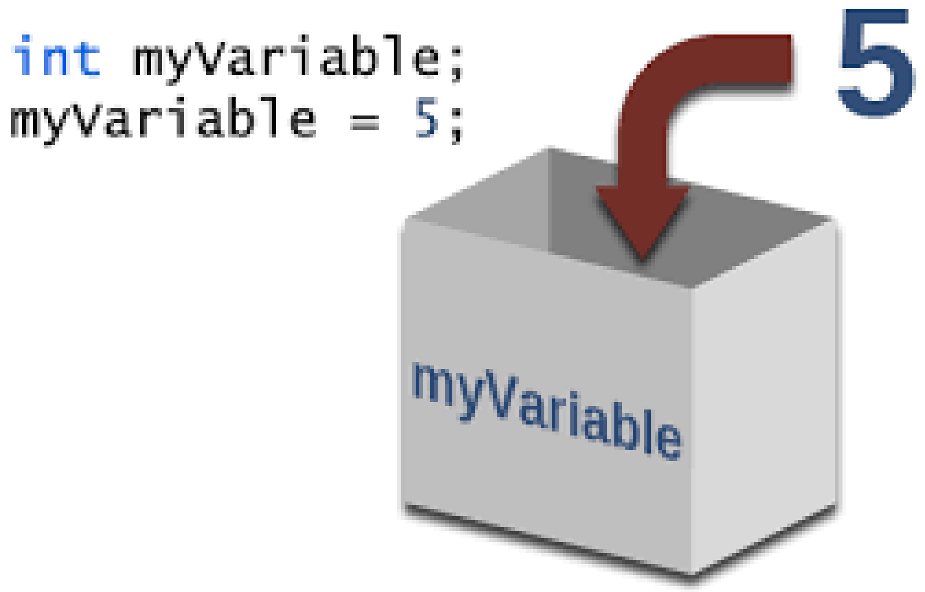
- Variable declaration and assignment can be done in the same line.

```
int age = 25;
float salary = 30000.0;
```

# Variable Declaration and Assignment



The variable has no value until the user enters a name.



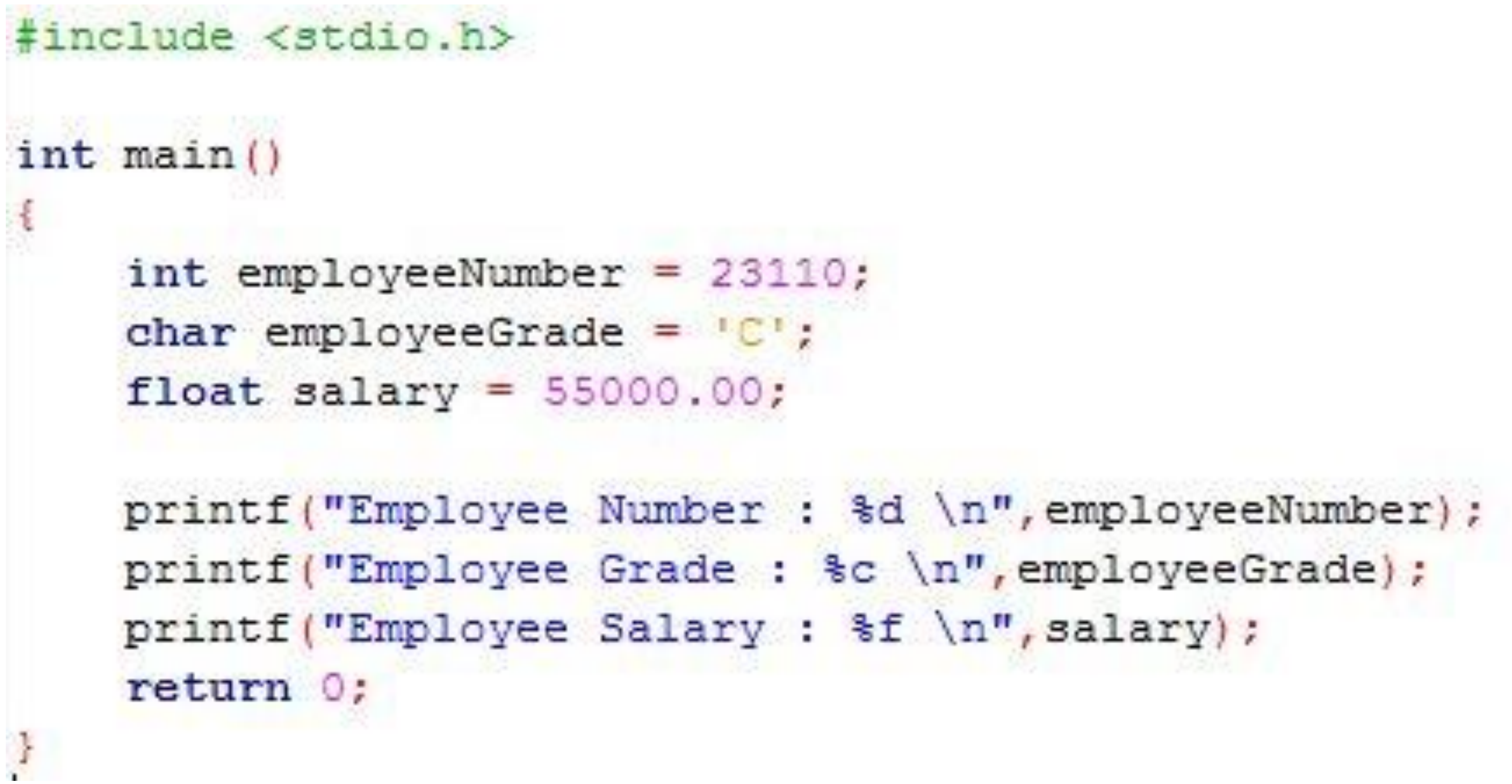
```
int myVariable;
myVariable = 5;
```

## Variable Names

- The name of a variable can be composed of letters, digits, and the underscore character(_). Spaces are not allowed in variable names.
- It must begin with either a letter or an underscore but not with a digit.
- Upper case and lower-case letters are distinct, so x and X are two different names. Traditional C practice is to use lower case for variable names, and all upper case for symbolic constants.
- Cannot use C Keywords(reserved words) such as if, else, while, for, etc. as a variable name.

| Acceptable Variable Names | Unacceptable Variable Names |
|---|---|
| Grade<br>firstName<br>last_name<br>telephone1<br>distance2school | 1st_Name<br>last name<br>Street-name<br>case |

## Variables : sample Program



```
#include <stdio.h>

int main()
{
    int employeeNumber = 23110;
    char employeeGrade = 'C';
    float salary = 55000.00;

    printf("Employee Number : %d \n",employeeNumber);
    printf("Employee Grade : %c \n",employeeGrade);
    printf("Employee Salary : %f \n",salary);
    return 0;
}
```

# Constants

- Constants refer to fixed values that the program may not alter during its execution. These fixed values are also called **literals**.
- Constants can be of any of the basic data types like an integer, a floating point, a character, or a string.
- Constants are treated just like regular variables except that their values cannot be modified after their definition.
- Constants are utilized when unchangeable values are needed to be used in our program. For example, when value of Pi(3.1415926) is used in our program for the calculations, we can use it as a constant.
- C has three types of constants: **Literal constants, Preprocessor constants and constants using const keyword**.

# Literal Constants

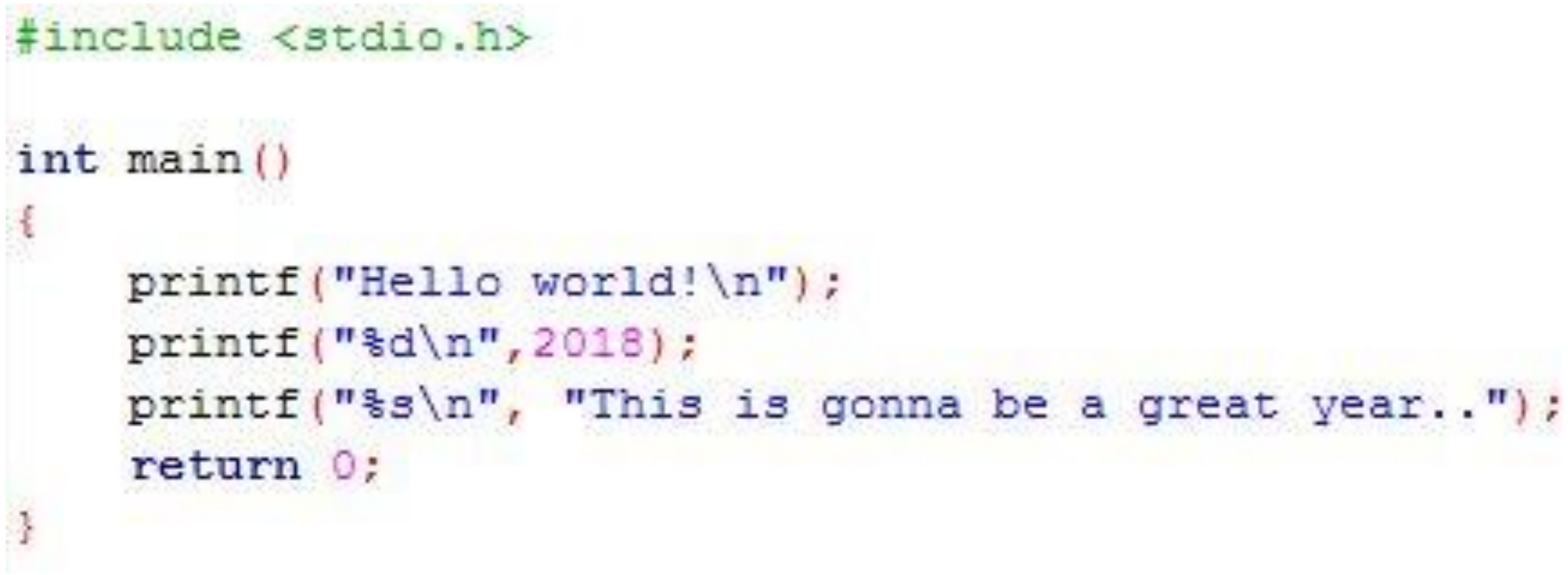
- A literal constant is a value that we use directly in our code.

```
int x = 10;

char c = 'A';

Int y = 20 * 2;
```

Here, 10, ‘A’, 20 and 2 are literal constants.

- In our first C program we printed a string, “Hello, world” on screen. It can also be considered as a literal constant.
- Literal constants are used when that value is used only at one place in our program. The value is not stored in the memory for further use within the program.
- Literal constants do not have names.

## Literal Constants : Sample Program



```c
#include <stdio.h>

int main()
{
    printf("Hello world!\n");
    printf("%d\n",2018);
    printf("%s\n", "This is gonna be a great year..");
    return 0;
}
```

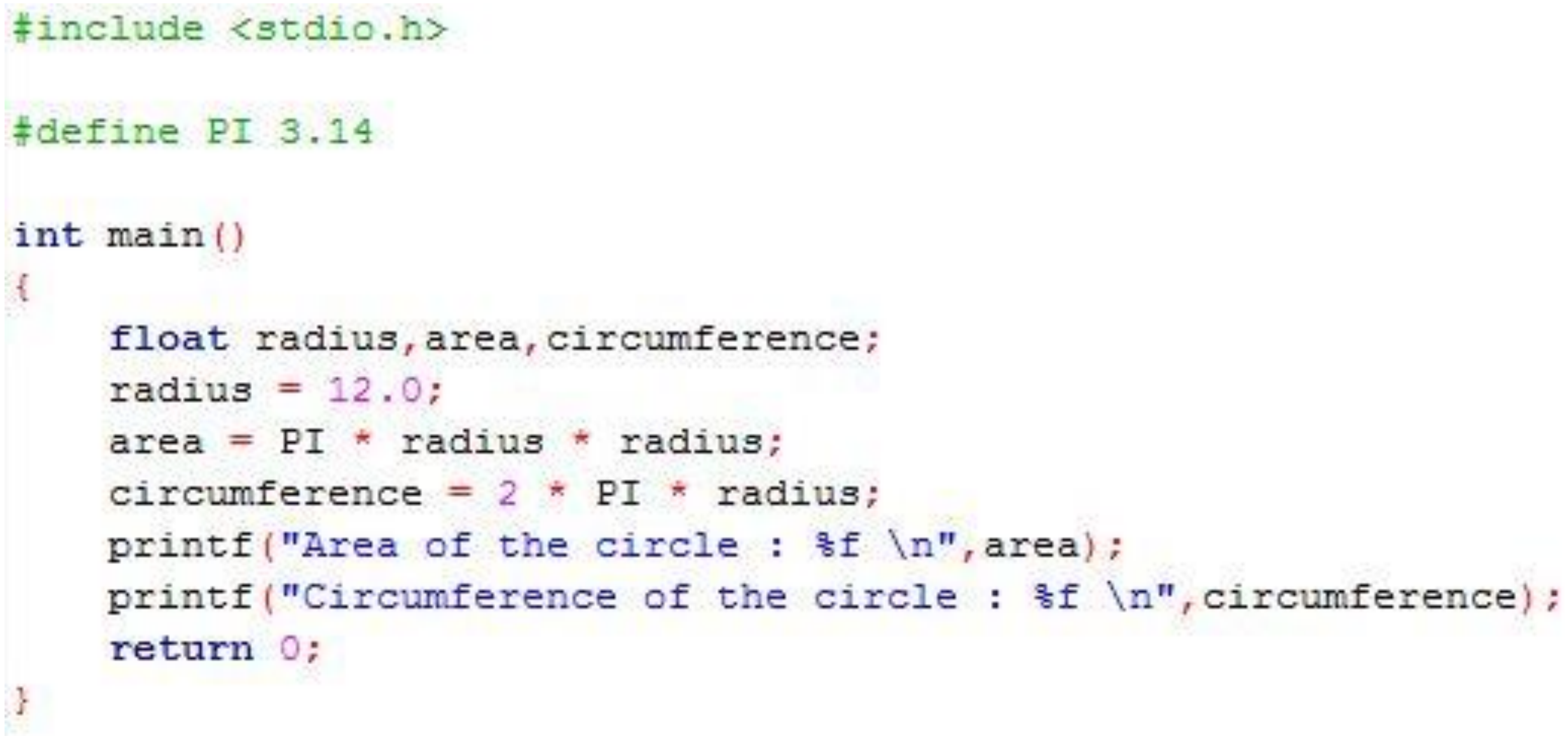
## Preprocessor Constants

- In this type of constants, there are no memory locations involved.
- In this mechanism, the constant is defined on top area(preprocessor area before main function).
- Use **#define** directive followed by the **name** and the **value**. No assignment sign is used to set the value since it is not a variable (the value is not stored in the memory).
- By convention, upper cases are used for preprocessor constants.
- The basic syntax is as followed:

```
#define MAXAGE 35
```

- At the end of the line, no semi colon (;) is used. Also, no type is mentioned.
- When these constants are used, all the occurrences of the MAXAGE in the program are replaced with 35 before it is sent to the compiler.

## Preprocessor Constants : Sample Program



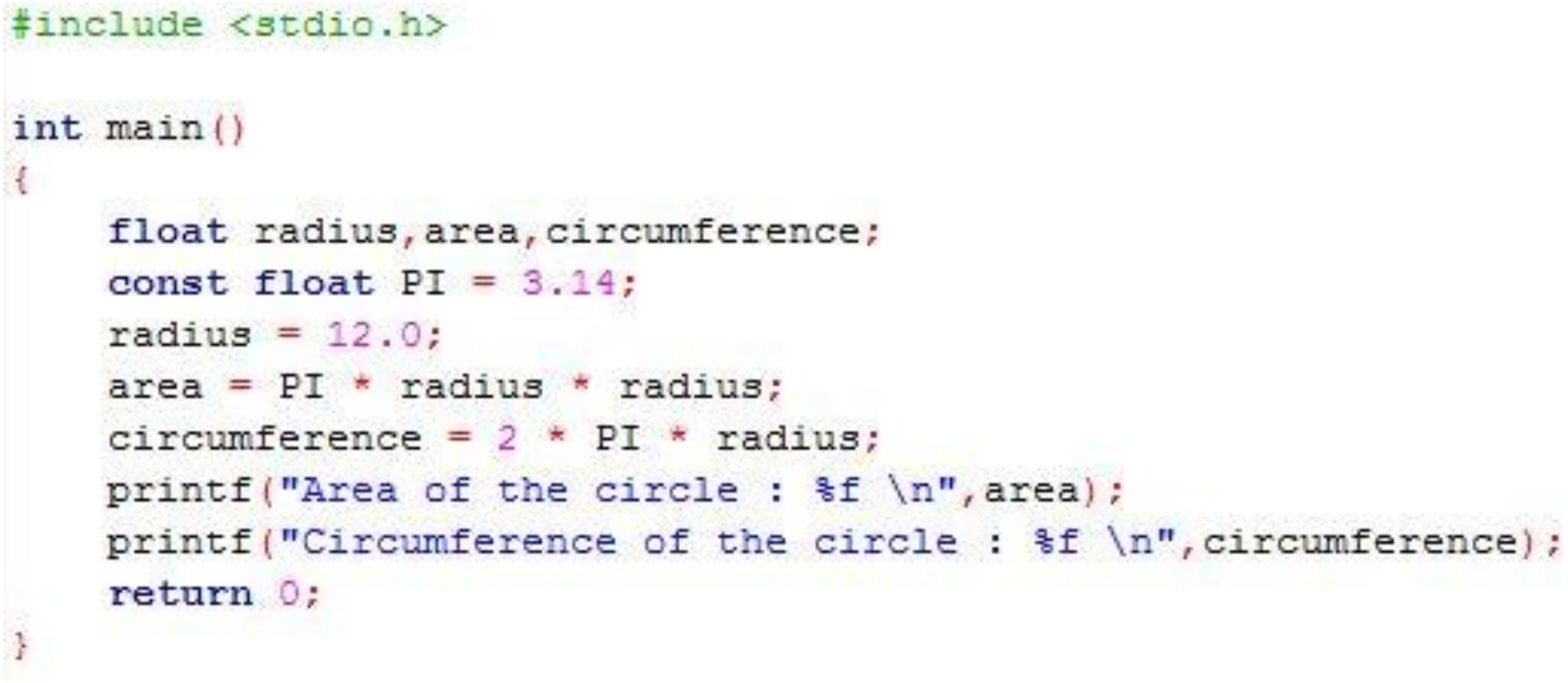
```c
#include <stdio.h>

#define PI 3.14

int main()
{
    float radius,area,circumference;
    radius = 12.0;
    area = PI * radius * radius;
    circumference = 2 * PI * radius;
    printf("Area of the circle : %f \n",area);
    printf("Circumference of the circle : %f \n",circumference);
    return 0;
}
```

## Constants with *const* keyword

- These constants are very similar to a regular variable.  It is a named memory location to store a data that we use in our program.
- These constants have names and data types.
- The only difference between this kind of constant and a variable is that constant's value cannot be modified after assigning.
- By convention, only upper-case letters are used for the name.
- The basic syntax is as followed:

```
const int MAXAGE = 35;
```

## *const* : Sample Program



```
#include <stdio.h>

int main()
{
    float radius,area,circumference;
    const float PI = 3.14;
    radius = 12.0;
    area = PI * radius * radius;
    circumference = 2 * PI * radius;
    printf("Area of the circle : %f \n",area);
    printf("Circumference of the circle : %f \n",circumference);
    return 0;
}
```

## Character and String Constants

- Character constant is a single alphabet, a single digit or a single special symbol enclosed within single quotes.
- Maximum length of a character constant is 1 character.

```
    'A'    'a'    '5'    '@'    '!'
```

- String constants are enclosed within double quotes.

```
    "Hello world!"

    "This is a string constant \n"
```

## Escape sequences

- An escape sequence is a sequence of characters that does not represent itself when used inside a character or string,
- It is translated into another character or a sequence of characters that may be difficult or impossible to represent directly.
- In C, all escape sequences consist of two or more characters. The first character is always a backslash (\ escape character). It is followed by characters determine the interpretation of the escape sequence.

| Backslash Character | Meaning | Backslash Character | Meaning | Backslash Character | Meaning |
|---|---|---|---|---|---|
| \b | Backspace | \t | Horizontal tab | \0 | Null |
| \f | Form feed | \” | Double quote | \a | Alert or bell |
| \n | New line | \’ | Single quote | \? | Question Mark |
| \r | Carriage return | \\ | Backslash | | |

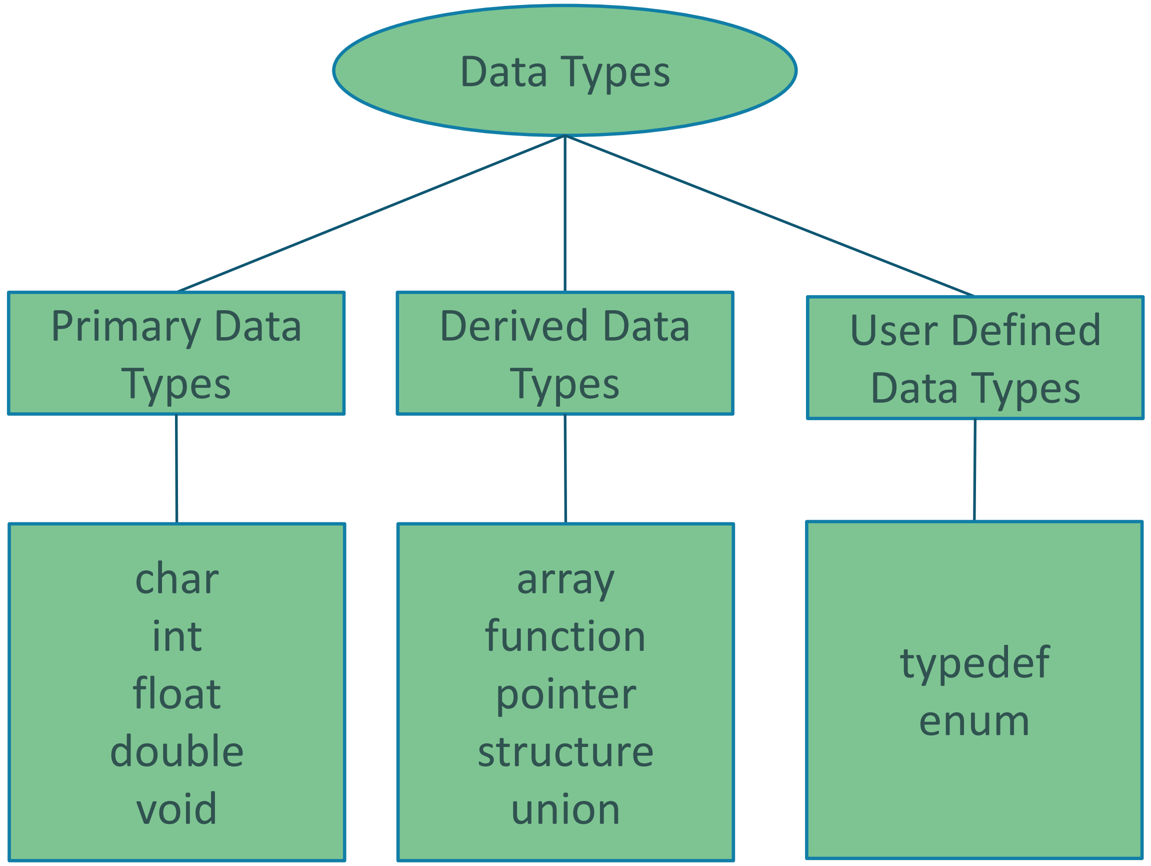
# C Data Types

- In previous examples, we have observed that there are various kind of data can be used in a C program.
- When they are used, in most of the cases, we have to tell the compiler what kind of data it is. Depending on our statement, the compiler treat the data differently.
- C programming language defines “Data Types” to indicate the nature of the data which is used in a program.
- C is a **typed language**.
- Each variable is given a specific type which defines **what values it can represent**, **how its data is stored in memory**, and **what operations can be performed** on it.
- The verity of data types allow the programmer to select the appropriate data type to satisfy the need of application as well as the needs of different machines.

# C Data Types

There are three classes of Data-Types in C.

- Primary Data Type
- Derived Data Type
- User Defined Data Type



## Primary Data Types

There are number of **qualifiers** such as **short**, **long**, **signed** and **unsigned** can be applied to these primary data types.

| Type | Qualifiers |
|---|---|
| char | signed, unsigned |
| int | short, long, signed, unsigned |
| float | No qualifier |
| double | long |
| void | No qualifier |

## Primary Data Types

| Type | Size(Bytes) | Range |
|---|---|---|
| char / signed char | 1 | -128 to 127 |
| unsigned char | 1 | 0 to 255 |
| short int / signed short int | 2 | -32768 to 32767 |
| unsigned short int | 2 | 0 to 65535 |
| int / signed int | 4 | -2147483648 to 2147483647 |
| unsigned int | 4 | 0 to 4294967295 |
| long int / signed long int | 4 | -2147483648 to 2147483647 |
| unsigned long int | 4 | 0 to 4294967295 |
| float | 4 | 1.8e-38 to 3.4e+38 |
| double | 8 | 2.2e-308 to 1.8e+308 |
| long double | 16 | |

## Format Specifiers

- Format Specifiers are used to inform the compiler in which format the data should be printed or should be read.
- They start with a percentage ***%*** operator and followed by a special character for identifying the type of the data.

| Format Specifier | Description | Format Specifier | Description |
|---|---|---|---|
| %d | Integer | %u | Unsigned Integer |
| %f | Floating point | %x | Hexadecimal |
| %c | Character | %0 | Octal |
| %s | Character String | %ld | Long integer |

```
Int x = 10;
printf("The value is %d", x);
```

➡

**The value is 10**

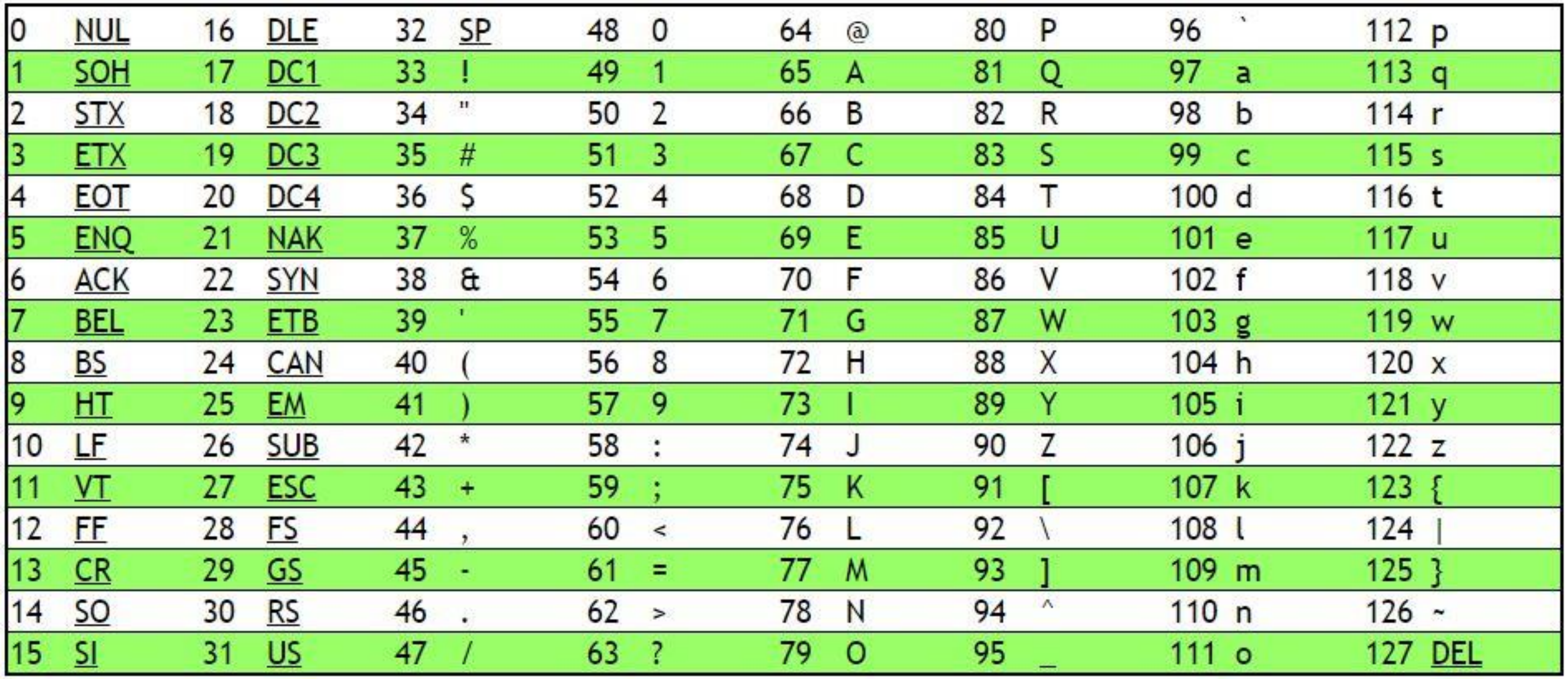
## Characters

- Character data type is used to represent ASCII character data.
- Each character is represented with 8 bits (1 byte).
- Each character is associated with an integer value, therefore char data type is also considered as an integer data type.
- Unsigned characters (unsigned char) represents values from 0 to 255 and while 0-127 values associated with ASCII characters, 128-255 values represent extended ASCII character set.
- Signed characters(signed char or char) represents values from -128 to 127. As analogues to unsigned char, 0-127 values associated with ASCII characters however,  extended ASCII characters are represented by values from -128 to -1.
- In both cases, the internal value in binary is the same but depending on the data type of the value, the interpretation is different.
- If it is unsigned char, the binary is translated into regular decimal value but in signed char, the binary is translated into decimal value using 2’s complement.

# Character Representation

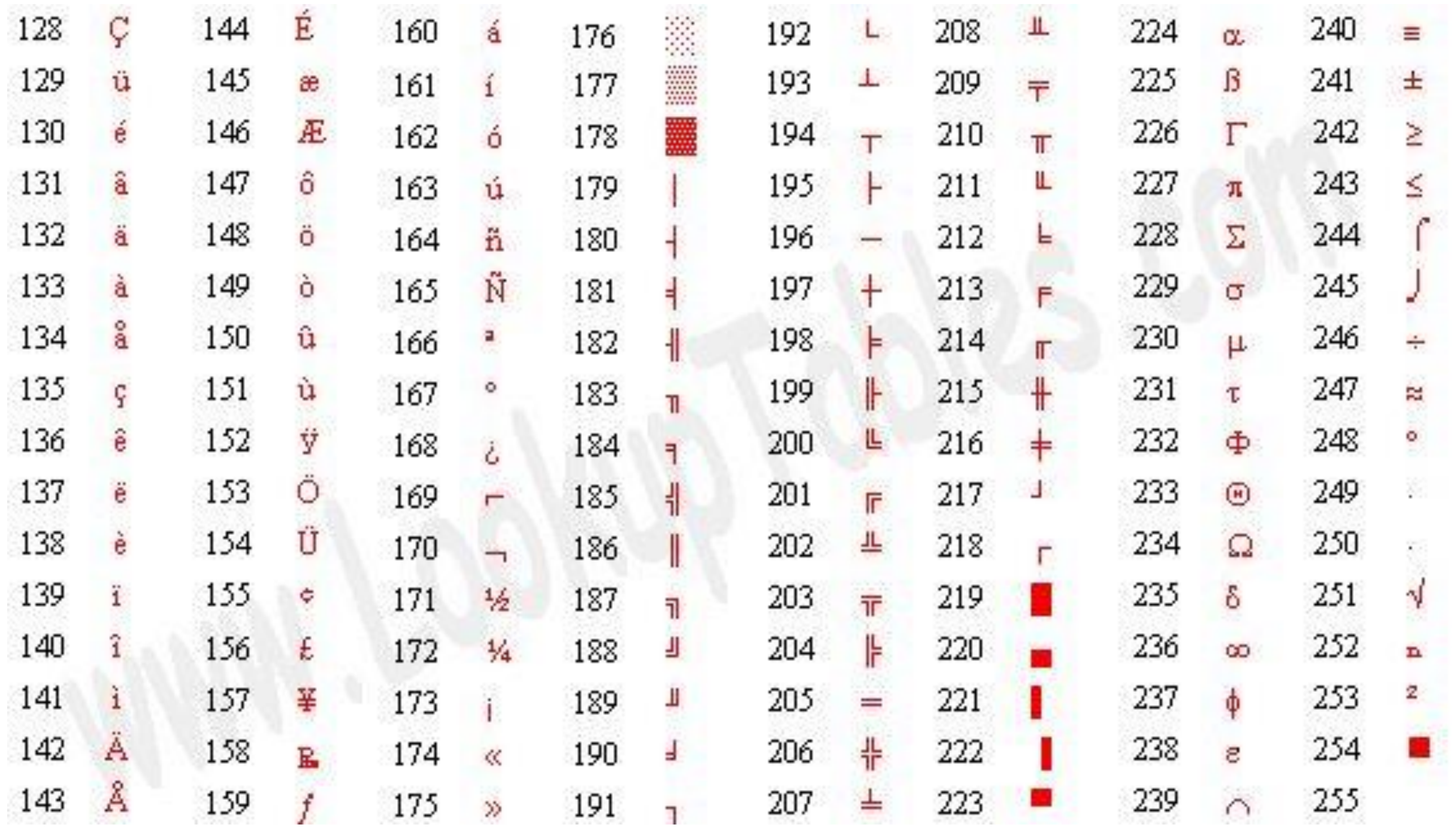
| | Character | unsigned | signed |
|---|---|---|---|
| 0 0 0 0 0 0 0 0 | NUL (null) | 0 | 0 |
| 0 0 0 0 0 0 0 1 | SOH (start of heading) | 1 | 1 |
| .. | | .. | .. |
| | | .. | .. |
| 0 1 1 1 1 1 1 1 | ~ | 127 | 127 |
| 1 0 0 0 0 0 0 0 | DEL (delete) | 128 | -128 |
| .. | | .. | .. |
| | | .. | .. |
| 1 1 1 1 1 1 1 0 | ■ | 254 | -2 |
| 1 1 1 1 1 1 1 1 | nbsp ( Non-breaking space) | 255 | -1 |

# ASCII Chart



| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | NUL | 16 | DLE | 32 | SP | 48 | 0 | 64 | @ | 80 | P | 96 | ` | 112 | p |
| 1 | SOH | 17 | DC1 | 33 | ! | 49 | 1 | 65 | A | 81 | Q | 97 | a | 113 | q |
| 2 | STX | 18 | DC2 | 34 | " | 50 | 2 | 66 | B | 82 | R | 98 | b | 114 | r |
| 3 | ETX | 19 | DC3 | 35 | # | 51 | 3 | 67 | C | 83 | S | 99 | c | 115 | s |
| 4 | EOT | 20 | DC4 | 36 | $ | 52 | 4 | 68 | D | 84 | T | 100 | d | 116 | t |
| 5 | ENQ | 21 | NAK | 37 | % | 53 | 5 | 69 | E | 85 | U | 101 | e | 117 | u |
| 6 | ACK | 22 | SYN | 38 | & | 54 | 6 | 70 | F | 86 | V | 102 | f | 118 | v |
| 7 | BEL | 23 | ETB | 39 | ' | 55 | 7 | 71 | G | 87 | W | 103 | g | 119 | w |
| 8 | BS | 24 | CAN | 40 | ( | 56 | 8 | 72 | H | 88 | X | 104 | h | 120 | x |
| 9 | HT | 25 | EM | 41 | ) | 57 | 9 | 73 | I | 89 | Y | 105 | i | 121 | y |
| 10 | LF | 26 | SUB | 42 | * | 58 | : | 74 | J | 90 | Z | 106 | j | 122 | z |
| 11 | VT | 27 | ESC | 43 | + | 59 | ; | 75 | K | 91 | [ | 107 | k | 123 | { |
| 12 | FF | 28 | FS | 44 | , | 60 | < | 76 | L | 92 | \ | 108 | l | 124 | \| |
| 13 | CR | 29 | GS | 45 | - | 61 | = | 77 | M | 93 | ] | 109 | m | 125 | } |
| 14 | SO | 30 | RS | 46 | . | 62 | > | 78 | N | 94 | ^ | 110 | n | 126 | ~ |
| 15 | SI | 31 | US | 47 | / | 63 | ? | 79 | O | 95 | _ | 111 | o | 127 | DEL |

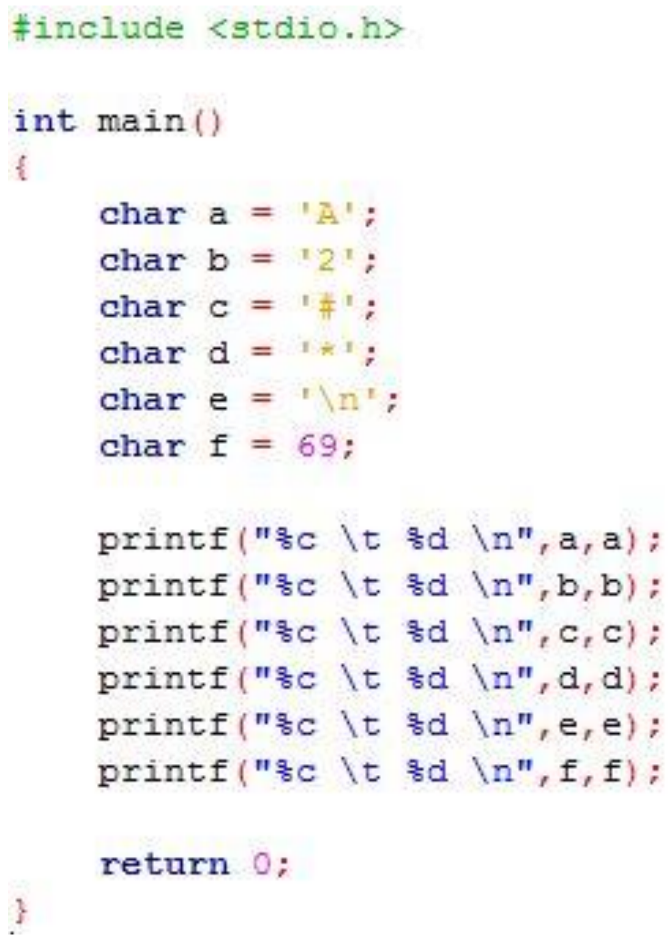
# Extended ASCII Chart



| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 128 | Ç | 144 | É | 160 | á | 176 | ░ | 192 | └ | 208 | ╨ | 224 | α | 240 | ≡ |
| 129 | ü | 145 | æ | 161 | í | 177 | ▒ | 193 | ┴ | 209 | ╤ | 225 | ß | 241 | ± |
| 130 | é | 146 | Æ | 162 | ó | 178 | ▓ | 194 | ┬ | 210 | ╥ | 226 | Γ | 242 | ≥ |
| 131 | â | 147 | ô | 163 | ú | 179 | │ | 195 | ├ | 211 | ╙ | 227 | π | 243 | ≤ |
| 132 | ä | 148 | ö | 164 | ñ | 180 | ┤ | 196 | ─ | 212 | ╘ | 228 | Σ | 244 | ⌠ |
| 133 | à | 149 | ò | 165 | Ñ | 181 | ╡ | 197 | ┼ | 213 | ╒ | 229 | σ | 245 | ⌡ |
| 134 | å | 150 | û | 166 | ª | 182 | ╢ | 198 | ╞ | 214 | ╓ | 230 | µ | 246 | ÷ |
| 135 | ç | 151 | ù | 167 | º | 183 | ╖ | 199 | ╟ | 215 | ╫ | 231 | τ | 247 | ≈ |
| 136 | ê | 152 | ÿ | 168 | ¿ | 184 | ╕ | 200 | ╚ | 216 | ╪ | 232 | Φ | 248 | ° |
| 137 | ë | 153 | Ö | 169 | ⌐ | 185 | ╣ | 201 | ╔ | 217 | ┘ | 233 | Θ | 249 | ∙ |
| 138 | è | 154 | Ü | 170 | ¬ | 186 | ║ | 202 | ╩ | 218 | ┌ | 234 | Ω | 250 | · |
| 139 | ï | 155 | ¢ | 171 | ½ | 187 | ╗ | 203 | ╦ | 219 | █ | 235 | δ | 251 | √ |
| 140 | î | 156 | £ | 172 | ¼ | 188 | ╝ | 204 | ╠ | 220 | ▄ | 236 | ∞ | 252 | ⁿ |
| 141 | ì | 157 | ¥ | 173 | ¡ | 189 | ╜ | 205 | ═ | 221 | ▌ | 237 | φ | 253 | ² |
| 142 | Ä | 158 | ₧ | 174 | « | 190 | ╛ | 206 | ╬ | 222 | ▐ | 238 | ε | 254 | ■ |
| 143 | Å | 159 | ƒ | 175 | » | 191 | ┐ | 207 | ╧ | 223 | ▀ | 239 | ∩ | 255 | |

www.LookupTables.com

## Characters : Sample Program

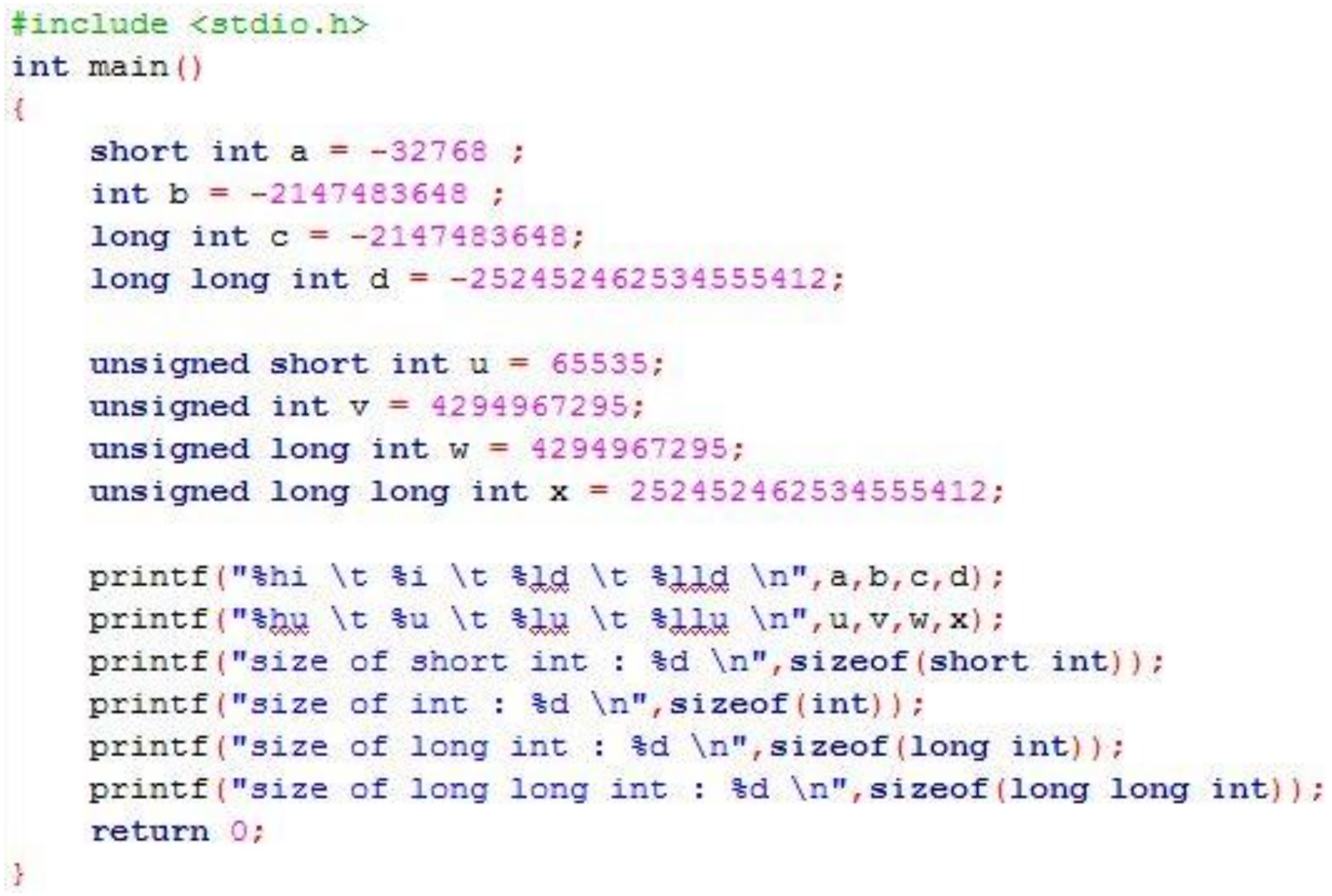


```c
#include <stdio.h>

int main()
{
    char a = 'A';
    char b = '2';
    char c = '#';
    char d = '*';
    char e = '\n';
    char f = 69;

    printf("%c \t %d \n",a,a);
    printf("%c \t %d \n",b,b);
    printf("%c \t %d \n",c,c);
    printf("%c \t %d \n",d,d);
    printf("%c \t %d \n",e,e);
    printf("%c \t %d \n",f,f);

    return 0;
}
```

## Integers

- Usually, Integers are represented with 2 or 4 bytes.
- In 32-bit systems, int and long int are the same but standard states that int should be at least 16 bit and long int should be at least 32 bit.
- For the sake of the portability of the program, int and long should be used accordingly.
- The signed and unsigned representation of Integers is same as in characters. When signed int is in action, the two’s complement is applied.
- char type can also be used as integers since every char is involved with an integer value, however it is not recommended due to the compiler overhead of type conversion.
- C99 introduces **long long int** type which is at least 64 bits.

# Integers : Sample Program



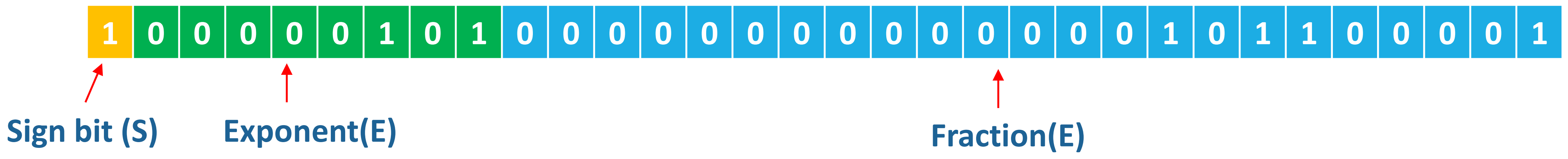
```c
#include <stdio.h>
int main()
{
    short int a = -32768 ;
    int b = -2147483648 ;
    long int c = -2147483648;
    long long int d = -252452462534555412;

    unsigned short int u = 65535;
    unsigned int v = 4294967295;
    unsigned long int w = 4294967295;
    unsigned long long int x = 252452462534555412;

    printf("%hi \t %i \t %ld \t %lld \n",a,b,c,d);
    printf("%hu \t %u \t %lu \t %llu \n",u,v,w,x);
    printf("size of short int : %d \n",sizeof(short int));
    printf("size of int : %d \n",sizeof(int));
    printf("size of long int : %d \n",sizeof(long int));
    printf("size of long long int : %d \n",sizeof(long long int));
    return 0;
}
```

## Floating Points

- Real numbers are represented in C by the floating-point types float, double, and long double.
- Since the representation is performed using a bounded number of bytes the floating-point types can't represent all real numbers.
- Integer representation can handle all the numbers within the range; however, floating point representation cannot handle all the real numbers within the range. It can only provide the best estimation for some numbers.
- The three floating point types provide different precisions for their representations depending on the space they utilize.
- Integers and floating-point numbers are treated differently in computers. They have different representation and are processed differently.

# Floating Points Representation

- A floating-point number is typically expressed in the scientific notation, with a fraction (F), and an exponent (E) of a certain radix (r), in the form of F×r^E.
- Decimal numbers use radix of 10 (F×10^E); while binary numbers use radix of 2 (F×2^E).
- In 32-bit single-precision floating-point representation:
  - The most significant bit is the sign bit (S), with 0 for positive numbers and 1 for negative numbers.
  - The following 8 bits represent exponent (E).
  - The remaining 23 bits represents fraction (F).



## Boolean Datatype

- Generally, Boolean data type is used to handle binary state values such as true and false and the size is 1 bit.
- In C programming language, primarily, no data type is defined for Boolean.
- Initially, int was used to handle Boolean values (1 and 0 for true and false).
- stdbool.h library provides data type for Boolean as bool.

```
#include <stdbool.h>

int main()
{
        bool b = true;
}
```

## Questions?