Advanced Programming

Zeinab Zali

References: (1) "C++ How to program" Deitel&Deitel, (2) "A Tour of C++" Bjarne Stroustrup, (3) Other useful learning pages such as geeksforgeeks and tutorialpoints

ECE Department, Isfahan University of Technology

Multiple bits units

$2^0 = 1$

$2^1 = 2$

$2^2 = 4$

$2^3 = 8$ (Byte)

$2^4 = 16$

$2^5 = 32$

$2^6 = 64$

$2^7 = 128$

$2^8 = 256$

$2^9 = 512$

$2^{10} = 1024$ (kilo)

$2^{20} = (2^{10})^2$ (Mega)

$2^{30} = (2^{10})^3$ (Giga)

$2^{40} = (2^{10})^4$ (Tera)

$2^{50} = (2^{10})^5$ (Peta)

Decimal to Binary Conversion

$4096 < 5000 < 8192$

$2^{12} \qquad 2^{13}$

1,0000,0000,0000

1,0010,0000,0000

1,0011,0000,0000

1,0011,1000,0000

1,0011,1000,1000 ✓

$5000 - 4096 = 904$, $512 < 904 < 1024$

$904 - 512 = 392$, $256 < 392 < 512$

$392 - 256 = 136$, $128 < 136 < 256$

$136 - 128 = 8$

13 bits

Decimal to Hex Conversion

Convert each four bits of the binary number to its equavalent hex digit (from right)

Decimal to Hex Conversion

$5007 = 5000 + 7 \Rightarrow$

$(0000,0000,0000,0000,0001,0011,1000,1000)_2 + (111)_2$

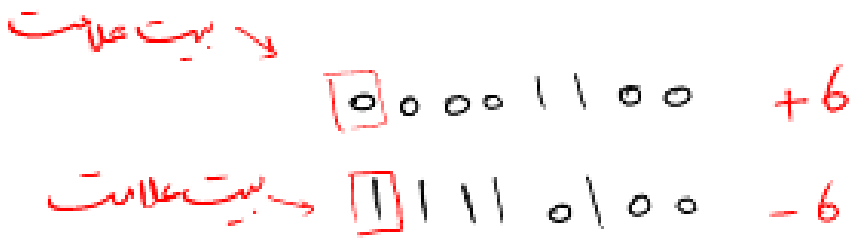
$0000,0000,0000,0000,0001,0011,1000,1111$

$0, 0, 0, 0, 1, 3, 8, F$ Hex

Signed Binary Numbers

Signed Binary Numbers use the Most Significant Bit as a sign bit to display a range of either positive numbers or negative numbers, So:

- an 8 bits unsigned number can have a value ranging from 0 to 255
- an 8 bits signed number can have a value ranging from -128 to 127

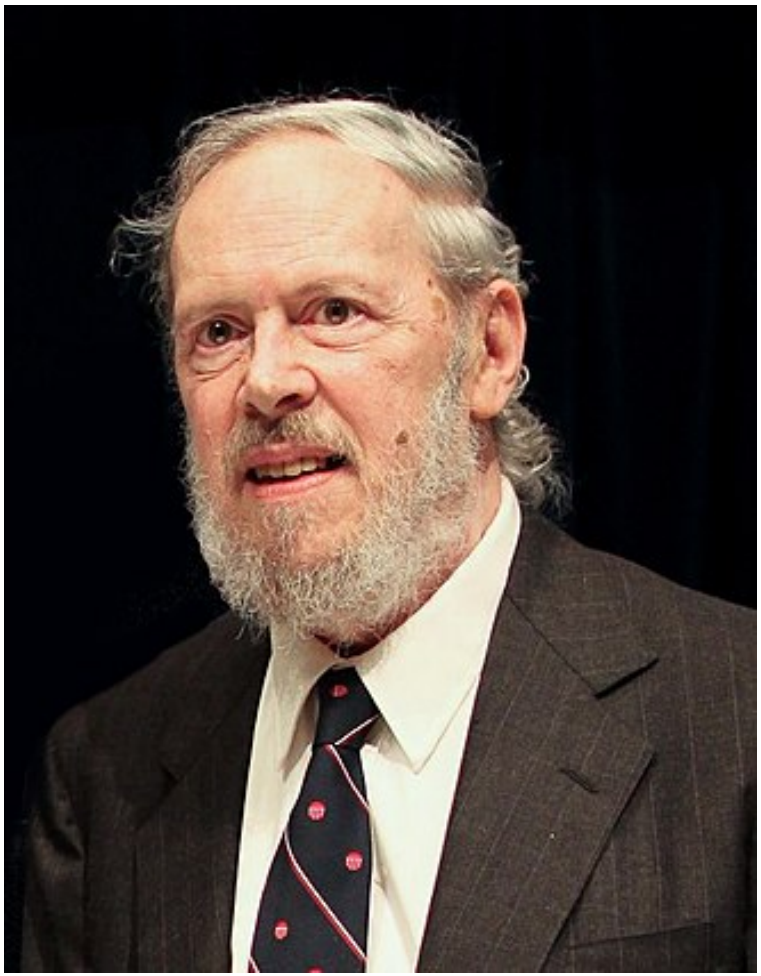


Binary Representation of Numbers in C++

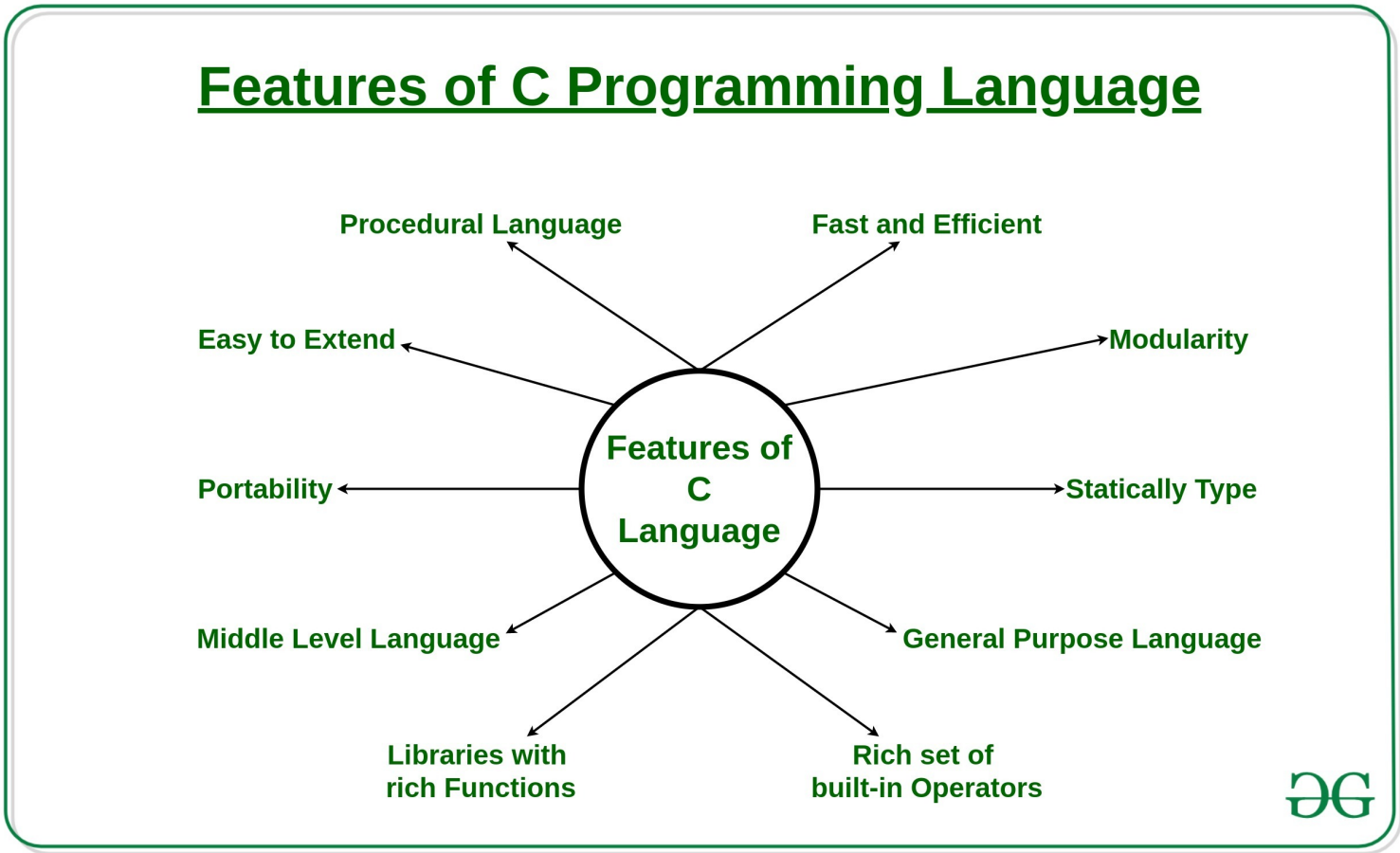
```cpp
#include <bitset>
#include <iostream>
using namespace std;
int main(){
    char x=127;
    char y = -128;
    cout << std::bitset<8>(x)<<endl;
    cout << std::bitset<8>(y)<<endl;
}
```

The Creator

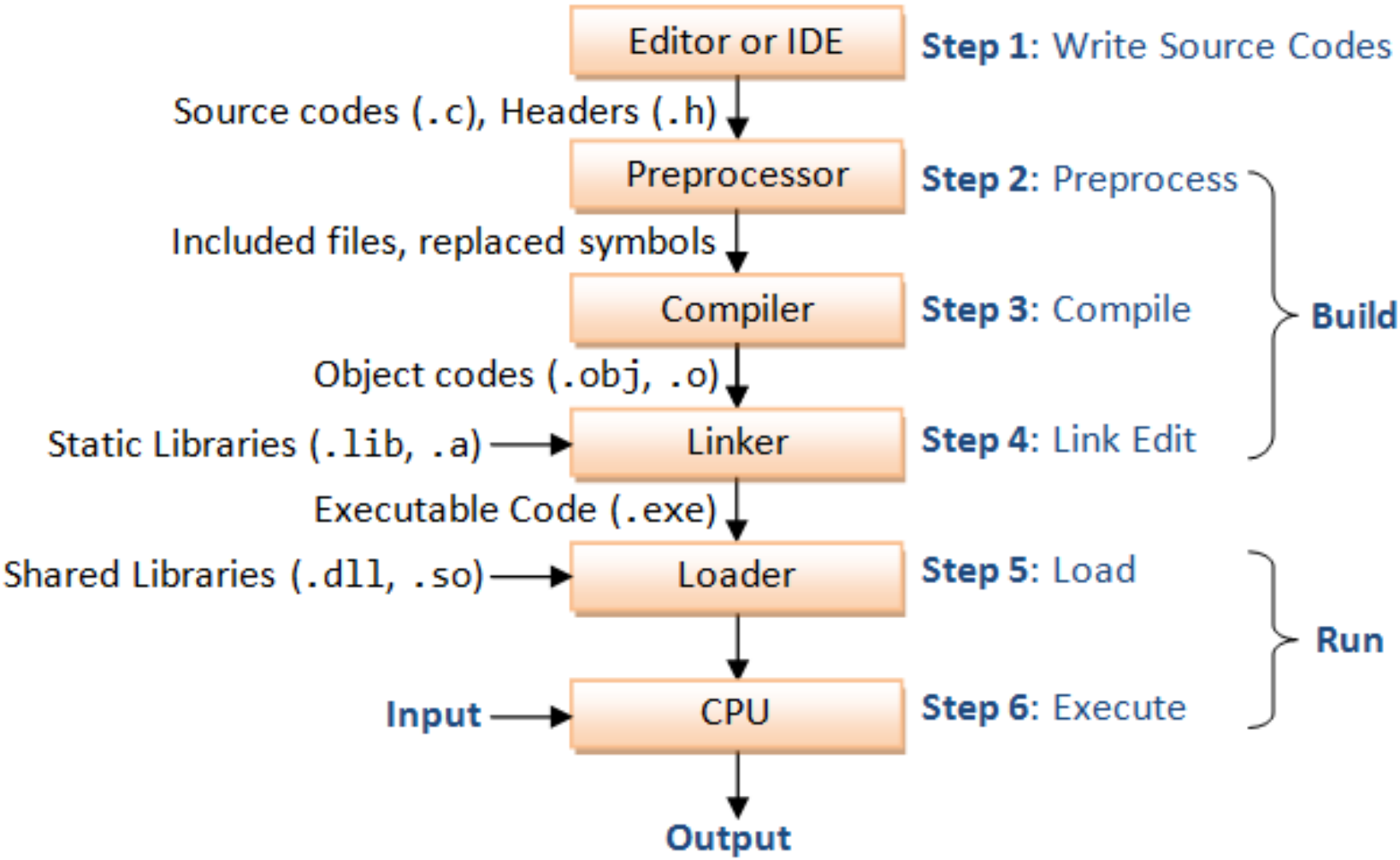
C is a general-purpose computer programming language created in the 1970s by Dennis Ritchie.



C Language Features



From Code to Runnable Program



C Preprocessor

C provides certain language facilities by means of a preprocessor, which is conceptually a separate first step in compilation. The two most frequently used features are

- **#include:** it is used to include the contents of a file during compilation
- **#define:** it is used to replace a token by an arbitrary sequence of characters (Macros)
- Other features include conditional compilation

C Compiler

There are four phases for a C program to become an executable

- Pre-processing(.i), Compilation(.s), Assembly(.o), Linking(executable binary)
- Run this command to see all intermediate files during compile time: gcc -Wall -save-temps filename.c –o filename

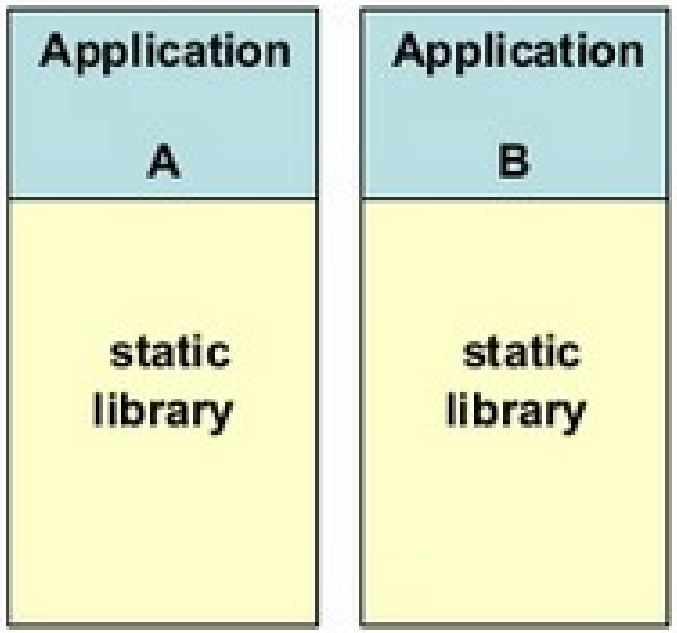
Others terms

- **Object file (.o):** They are produced by the compiler and consist of function definitions in binary form, but they are not executable by themselves.
- **Linker:** It is a program in a system which helps to link object modules of a program (and all used library functions) into a single object file
- **Loader:** It is the program of the operating system which loads the executable from the disk into the primary memory(RAM) for execution

Static Library (.lib or .a)

A collection of object files implementing some useful functions which are introduced (as some prototypes) in the header files.

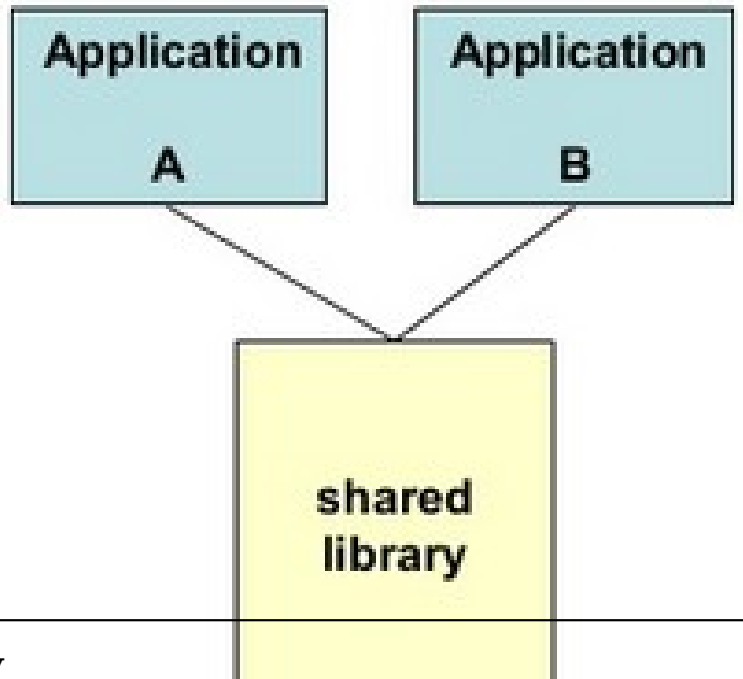
- We include their header files in our programs and then we can use their functions and definitions.
- They are copied and linked to our program in linking phase, so our executable program containes a copy of them.



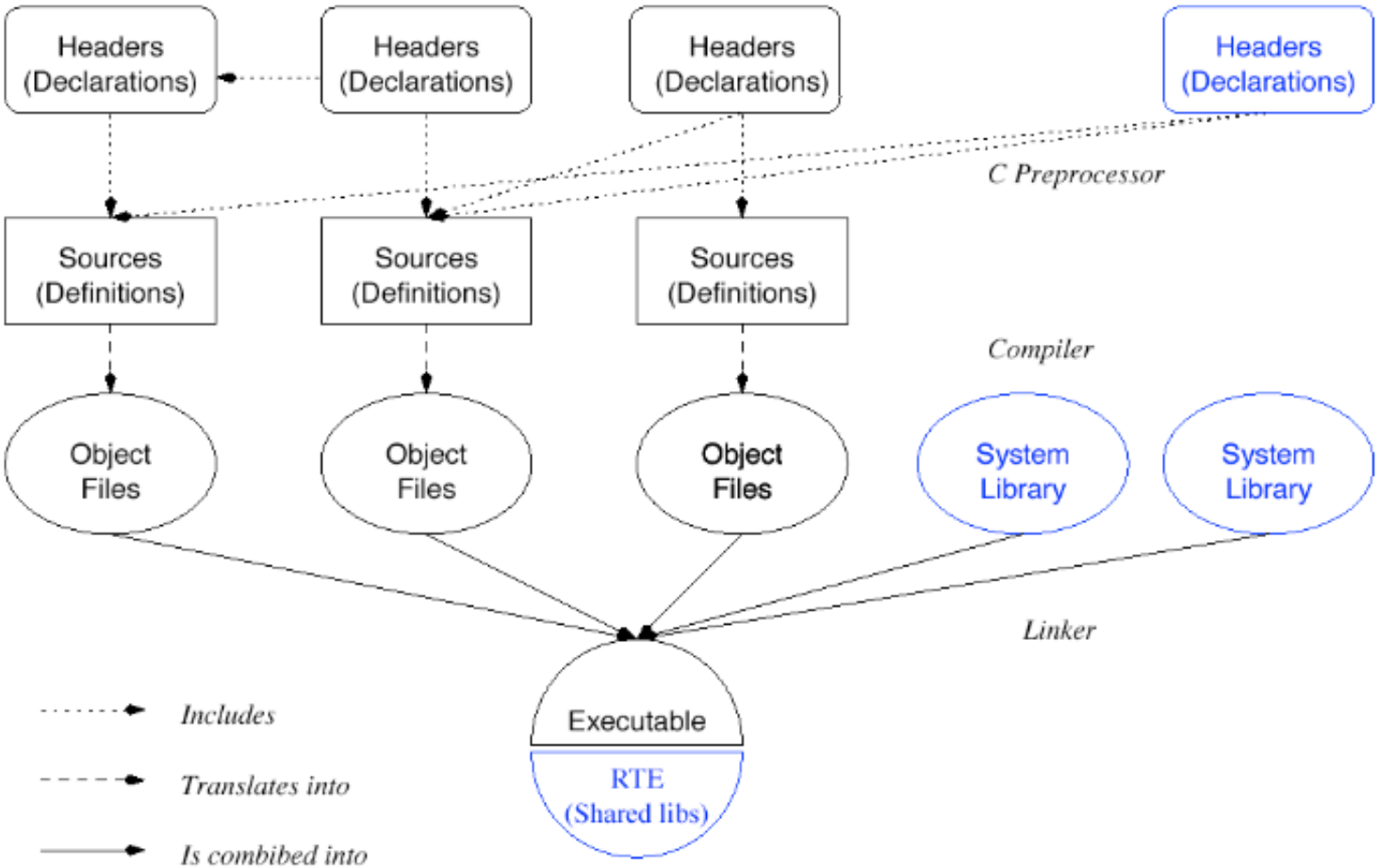
Dynamic Library (.so or .dll)

A collection of object files implementing some useful functions which are introduced (as some prototypes) in the header files.

- We include their header files in our programs and then we can use their functions and definitions.
- They are not copied for every program and are shared between all the programs that use them. So they exist as separate files outside of our executable files.



What Multiple files?



How Multiple files?

- **Header files:** Put the definitions and the prototypes of functions in the header file
- **Source files:** Include the header file and implement all their fuctions in the source file
- **Usage:** When you require to call a function or use a type, it is sufficient to include the header file containing its definition or prototype.
- **Compiling:** Compile all the source files besides their header files

Why Multiple files?

- **Reusability:** Implement once and use in different projects
- **Structured Programming:** Write clean, nice, not dirty and not messy codes
- **Team Working:** Partition a project related to its functionalities and operations

Code Selectively

It is possible to control preprocessing itself with conditional statements that are evaluated during preprocessing.

- This provides a way to include code selectively, depending on the value of conditions evaluated **during compilation**.
- For example, to make sure that the contents of a header file are included only once:

```
#ifndef HEADER_FILE
#define HEADER_FILE

//the entire header file file

#endif
```

Code Selectively

The special operator defined is used in '#if' and '#elif' expressions to test whether a certain name is defined as a macro

```
 9  #if defined(_WIN32)
10  #include <windows.h>
11
12  void my_sleep(int s)
13  {
14      Sleep(s * 1000);
15  }
16
17  #else /* __unix__ */
18  #include <unistd.h>
19
20  void my_sleep(int s)
21  {
22      sleep(s);
23  }
```

Variables and Basic Types

Variables and constants are the basic data objects manipulated in a program.

- **Declarations** list the variables to be used, and state what type they have and perhaps what their initial values are.
- The type of an object determines the set of values it can have and what operations can be performed on it.
- **Basic Types:** char, int, float, double
- some qualifiers can be applied to these basic types
  - Examples: short int, long int, signed, unsigned
- useful fuctions and headerfiles: sizeof(), limits.h

User-Defined Data Types

- typedef
- struct
- union
- enum

typedef

typedef is a facility called typedef for creating new data type names.

```c
typedef char string[100];

int main(){
    string name;
    printf("sizeof(name):%d\n", sizeof(name)); //sizeof(name):
}
```

struct

A structure is a collection of one or more variables, possibly of different types, grouped together under a single name.

```c
struct node {
    char key[20];
    int val;
    struct node *next;
};
typedef struct node Node;

int main(){
    Node n;
    printf("sizeof(Node):%d\n", sizeof(n));
}
```

union

A union is a variable that may hold (at different times) objects of different types and sizes

- It provides a way to manipulate different kinds of data in a single area of storage
- You can define a union with many members (maybe of different types), but only one member can contain a value at any given time

union

```
union grade{
        int gpa;
        float score;
};
struct student {
    char ID[20];
    union grade g;
};
int main(){
    struct student s;
    s.g.gpa = 3;
    printf("sizeof(Node)=%d\n", sizeof(s));
    printf("student gpa=%d\n", s.g.gpa);//g contains value of gpa
    s.g.score = 15.5;
    printf("sizeof(Node)=%d\n", sizeof(s));
    printf("student gpa=%d\n", s.g.gpa);
    //Not valid, because g contains score value
}
```

enum

An enumeration is a list of constant integer values.

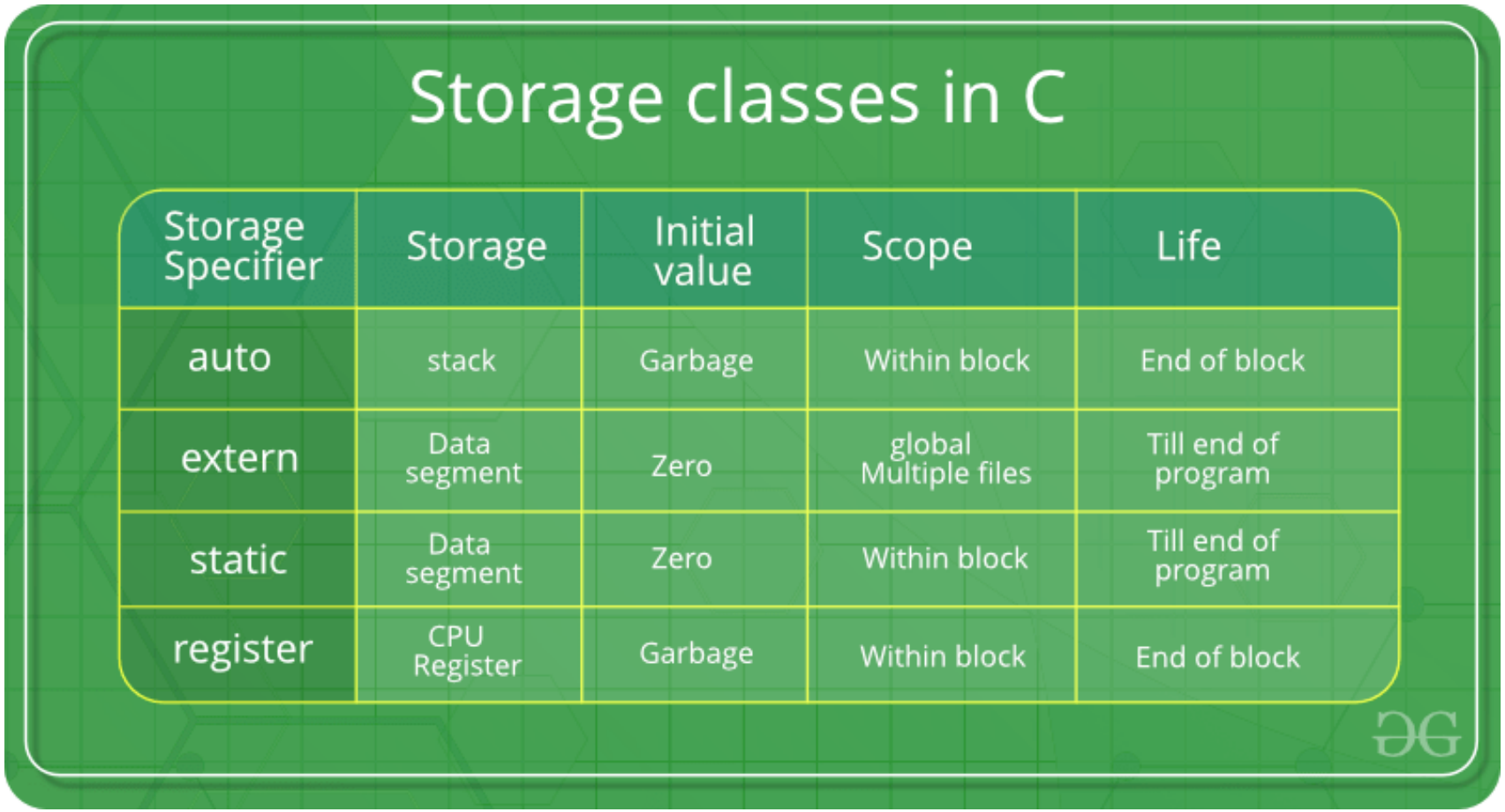
```
enum degree {undergrad, grad, phd};
struct student {
    char ID[20];
    union grade g;
    enum degree deg;
};
int main(){
    struct student s;
    s.deg = phd;
    printf("student degree:%d\n", s.deg);//student degree:2

}
```

Lets remember C with an example

Implementing a linked list

Storage Classes in C

A storage class represents the visibility and a location of a variable. It tells from what part of code we can access a variable.



Storage Classes in C

- **auto:** Auto variables can be only accessed within the block/function they have been declared and not outside them.
  - all variables created there are already autohardly ever used
- **extern:** An extern variable is nothing but a global variable initialized with a legal value where it is declared in order to be used elsewhere
  - The main purpose of using extern variables is that they can be accessed between two different files which are part of a large program.

Storage Classes in C

- **static:** Static variables have the property of preserving their value even after they are out of their scope!
- **register:** the compiler tries to store these variables in the register of the microprocessor if a free registration is available.
  - This makes the use of register variables to be much faster than that of the variables stored in the memory during the runtime of the program.

Declaring a Variable as Constant

- A **constant value** should be determined when defined and can not be changed after.
- A **constant pointer** cannot change the address of the variable to which it is pointing
- A **pointer to constant** is a pointer through which the value of the variable that the pointer points cannot be changed.

```
int main(){
    const int a = 10;  //a const value
    int b=100, c =200;
    int * const p1=&b; //a constant pointer
    p1 = &c;           //compile error
    const int* p2;     //pointer to a constant value
    p2 = &a;           //valid
    p2 = &b;
    *p2 = 300;         //compile error
}
```

constant in fuction arguments

- in Call by reference there is only a copy of an address (maximum 64 bytes)
- in Call by value, there is a copy of an argument (may be more than 64 bytes)
- So Call by reference speeds up the fuction call
  - We can use call by referece even if the value of an argument will not be changed in the function
  - Declaring function arguments const indicates that the function promises not to change these values

const in fuction arguments

```c
#include <stdio.h>

struct Node{
    long v1;
    double v2;
    char v3[100];
};
void print(const struct Node *n){
    pritf("%d,%f,%s",n->v1, n->v2, n->v3);
}
```

Pointer to function

A function pointer points to code, not data. Typically a function pointer stores the start of executable code.

```
//code from https://www.geeksforgeeks.org/
void add(int a, int b){
    printf("Addition is %d\n", a+b);
}
void subtract(int a, int b){
    printf("Subtraction is %d\n", a-b);
}
void multiply(int a, int b){
    printf("Multiplication is %d\n", a*b);
}
```

Pointer to function

```c
//code from https://www.geeksforgeeks.org/
int main()
{
    // fun_ptr_arr is an array of function pointers
    void (*fun_ptr_arr[])(int, int) = {add, subtract, multiply};
    unsigned int ch, a = 15, b = 10;
    printf("Enter Choice: 0 for add, 1 for subtract and 2 "
           "for multiply\n");
    scanf("%d", &ch);
    if (ch > 2) return 0;
    (*fun_ptr_arr[ch])(a, b);
    return 0;
}
```

Pointer to function

A simple C program to show function pointers as parameter

```c
//code from https://www.geeksforgeeks.org/
void fun1() { printf("Fun1\n"); }
void fun2() { printf("Fun2\n"); }
// A function that receives a simple function
// as parameter and calls the function
void wrapper(void (*fun)()){
    fun();
}
int main(){
    wrapper(fun1);
    wrapper(fun2);
    return 0;
}
```