C programming

Variable in C

Author: KhoTV

Lesson Objectives

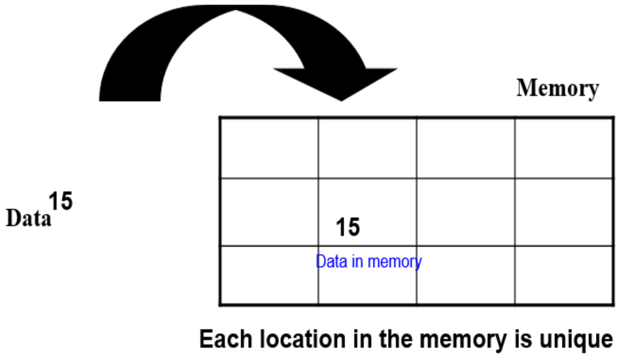
- Introductory question
- Basic Data Types
- Store Class
- Key words for variable
- Struct data type
- Casting

Section 1

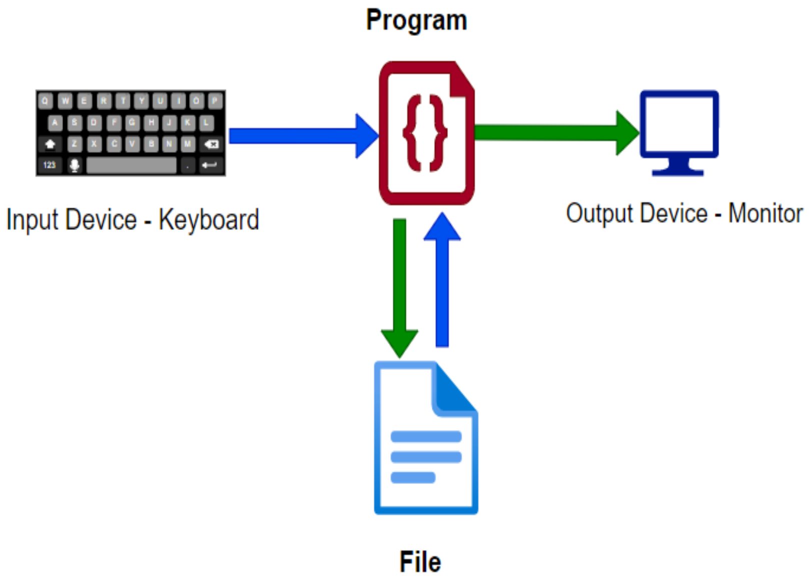
INTRODUCTORY QUESTION

Introductory question

- ✓ ***What is variable?***
- ✓ ***Why do need variable in programming?***



Variables allow to provide a meaningful name for the location in memory



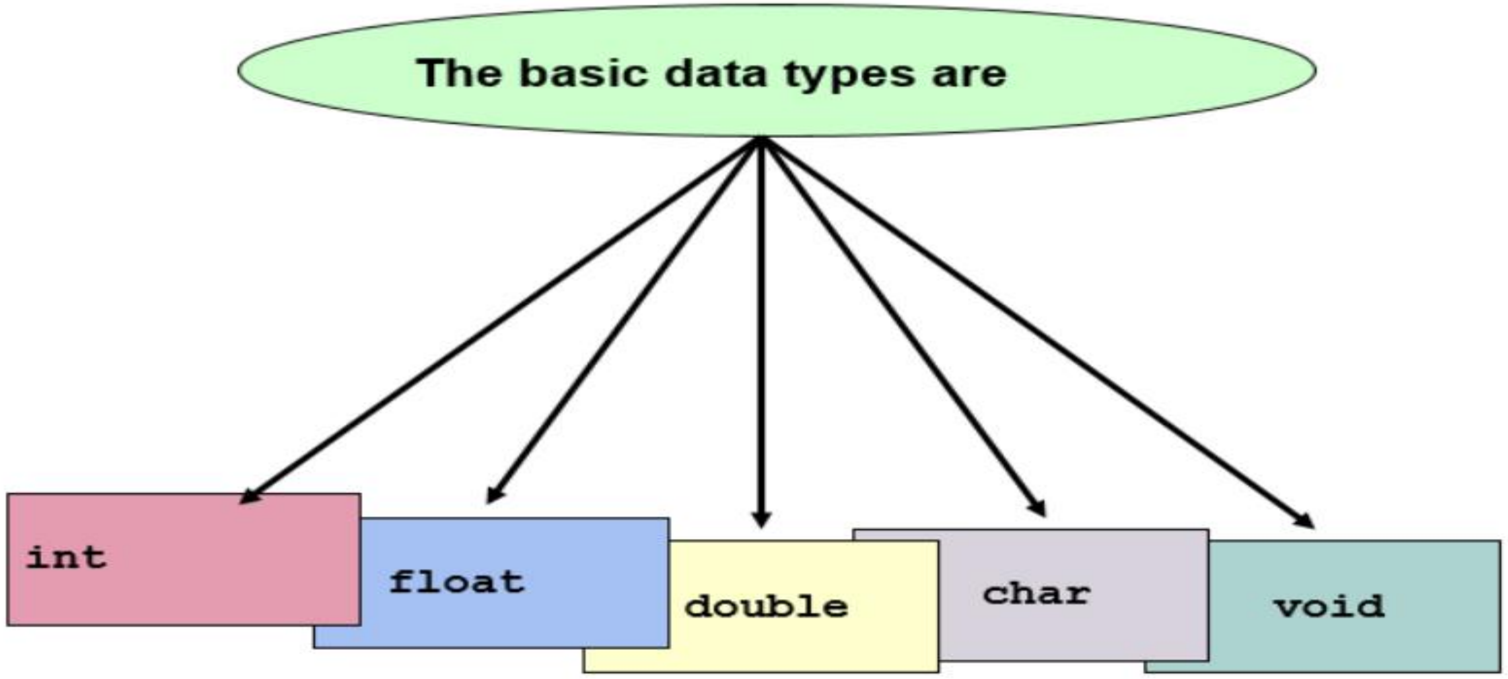
Section 2

BASIC DATA TYPES

Basic Data Types. Agenda

- *Basic data types in C*
- *Derived Data Types*

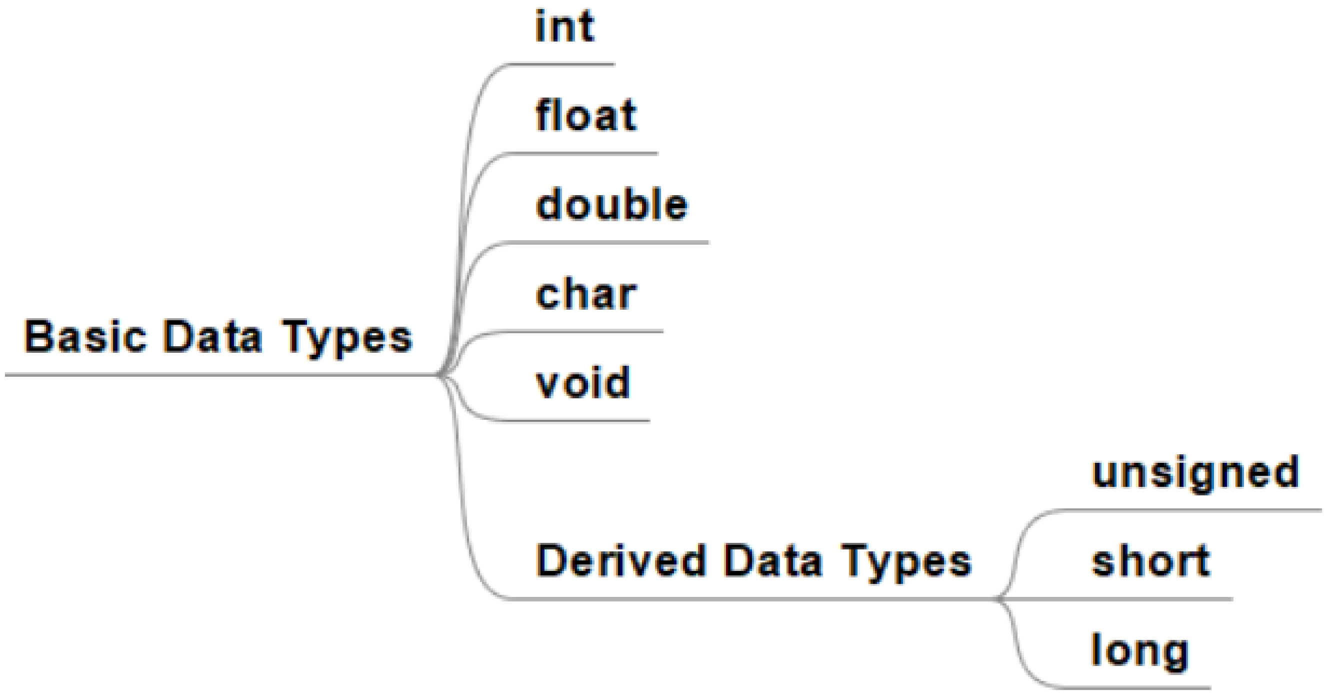
Basic Data Types in C



Derived Data Types

Data type Modifiers	+	Basic Data types	=	Derived data type
unsigned	+	int	=	unsigned int (Permits only positive numbers)
short	+	int	=	short int (Occupies less memory space than int)
long	+	int/double	=	Long int /longdouble (Occupies more space than int/double)

Basic Data Types. Summary



Section 3

STORE CLASS

Store Class. Agenda

- *Time life of a variable*
- *Scope of a variable*
- *Variable in memory*
- *Global and Local discriminative*

Time life of a variable

- *When was a variable allocated in memory?*
- *When was a variable destroyed?*

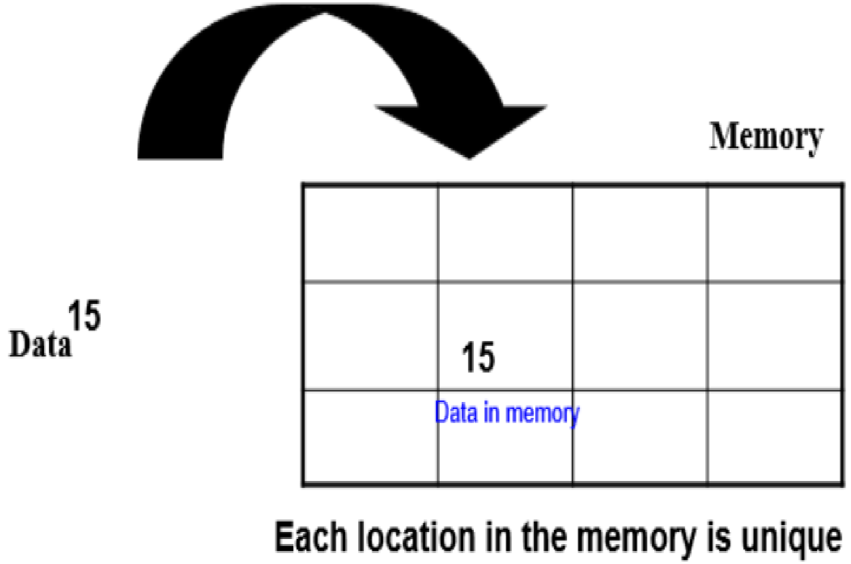
The **lifetime** of a variable is the time period in which the variable has valid memory.

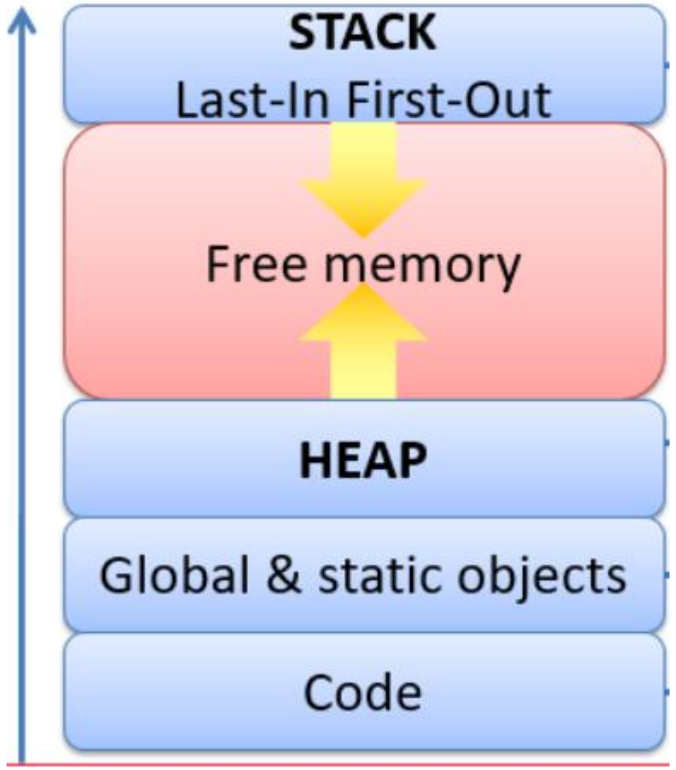
Scope of a variable

- *Where was a variable declared in the program?*

The **scope** of a declaration is the part of the program for which the declaration is in effect.

Variable in memory



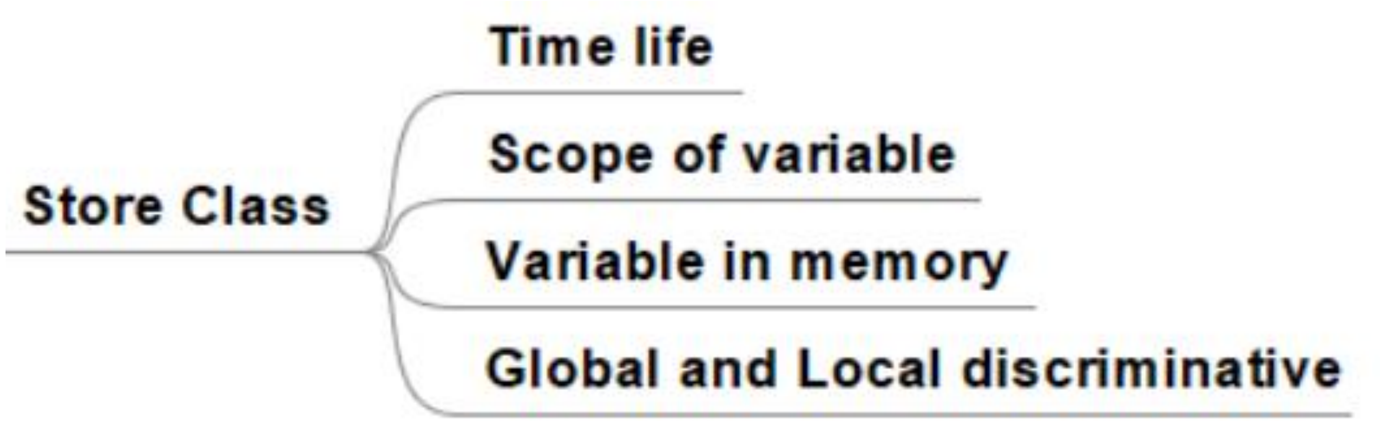


Global and Local variable discriminative

- Global variables are declared outside any function, and they can be accessed (used) on any function in the program. Local variables are declared inside a function, and can be used only inside that function. It is possible to have local variables with the same name in different functions.

- Time life ?
- *Scope ?*
- *Memory ?*

Store class. Summary



Section 4

KEY WORD FOR VARIABLE

Key word for variable. Agenda

- *static*
- *extern*
- *register*
- *volatile*

Key word: static

- In some programming languages such as C (and its close descendants like C++, Objective-C, and Java), static is a reserved word controlling both lifetime (as a static variable) and visibility (depending on linkage). The effect of the keyword varies depending on the details of the specific programming language.
  (wiki)
- Static global variable ?
- Static function ?
- Static local variables ?

Key word: extern

- **Extern** is a keyword in C programming language which is used to declare a global variable that is a variable without any memory assigned to it. It is used to declare variables and functions in header files. Extern can be used access variables across C files.
- **Extern** variable ?
- **Extern** function ?

Key word: register

- Register variables tell the compiler to store the variable in CPU register instead of memory. Frequently used variables are kept in registers and they have faster accessibility. We can never get the addresses of these variables. “register” keyword is used to declare the register variables.

(wikis)

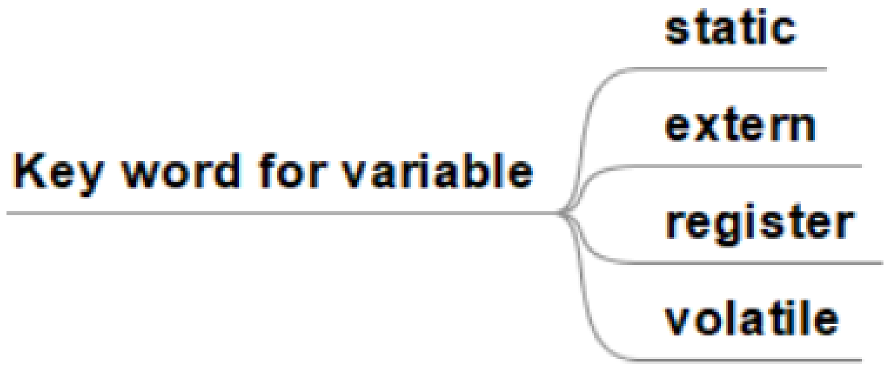
Key word: volatile

- The volatile keyword is intended to prevent the compiler from applying any optimizations on objects that can change in ways that cannot be determined by the compiler.

User case

- *What is the key words: static,…?*
- *When will use static, extern, register, volatile key words?*

Key word for variable. Summary



Section 6

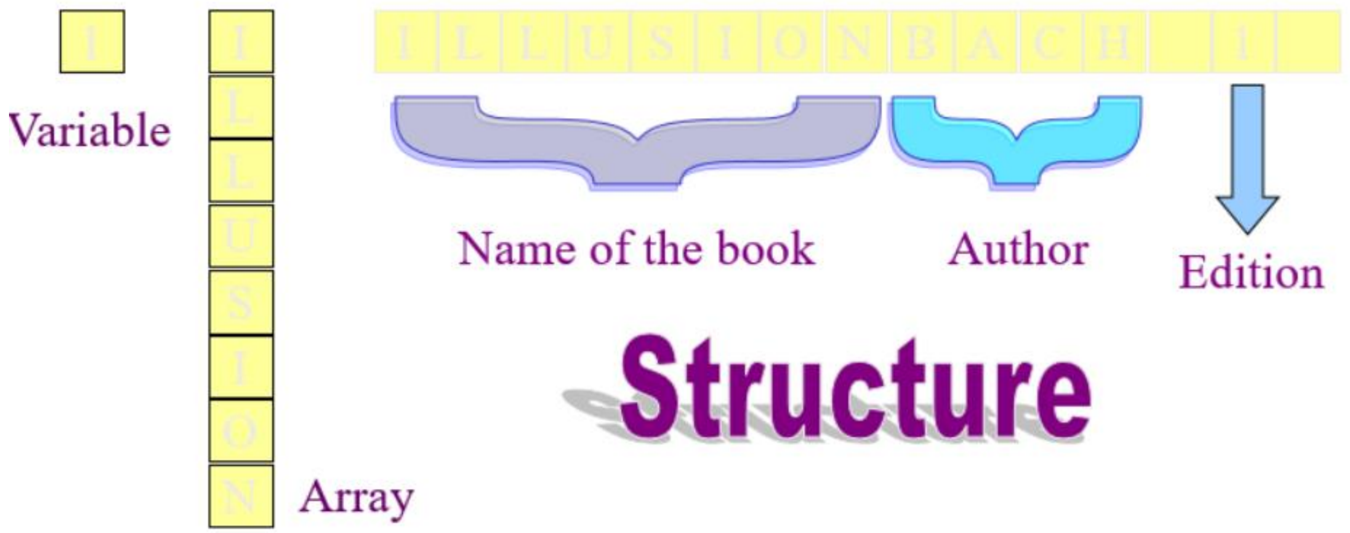
STRUCTURE DATA TYPE

Structure data type. Agenda

- *Structure variable*
- *Union*
- *Enum*

What is structure data type?

- A structure consists of a number of data items, which need not be of the same data type, grouped together
- The structure could hold as many of these items as desired



Define a structure data type.

- A structure definition forms a template for creating structure variables
- The variables in the structure are called structure elements or structure members
- Example:

```
struct cat
    {char bk_name[25];0x00-0x24;25,26,27
     char author [20];  0x28-0x47;
     int edn;           0x48-0x51;
     float price;       0x52-0x55;
    };
```

Declare structure data type variables.

- Once the structure has been defined, one or more variables of that type can be declared
- Example: **struct cat books1;**
- The statement sets aside enough memory to hold all items in the structure

Other ways

```
struct cat {    char bk_name[25];
                char author[20];
                int edn;
                float price;
          } books1, books2;
```

```
struct cat books1, books2;
```

or

```
struct cat books1;
struct cat books2;
```

Accessing Structure Elements

- Structure elements are referenced through the use of the **dot operator** (.), also known as the **membership operator**
- Syntax:

```
structure_name.element_name
```

- Example:

```
scanf(“%s”, books1.bk_name);
```

Size of structure data type.

- A new data type name can be defined by using the keyword typedef
- It does not create a new data type, but defines a new name for an existing type
- Syntax:

```
typedef type name;
```

- Example:

```
typedef float deci;
```

- typedef cannot be used with storage classes

Size of structure data type.

- Size of structure data type ?
- Data structure alignment ?
- User case ?

Union variable

- What is Union?
- Define Union.
- Declare Union variables.
- Size of Union?
- User case?

Union variable: What is Union?

A **union** is a special data type available in C that allows to store different data types in the same memory location. You can define a union with many members, but only one member can contain a value at any given time. Unions provide an efficient way of using the same memory location for multiple-purpose.

Union variable: Define Union?

```
union [union tag] {
   member definition;
   member definition;
   ...
   member definition;
} [one or more union variables];
```

Union variable?

- Declare Union variables:

union <union tag> <union variable>;

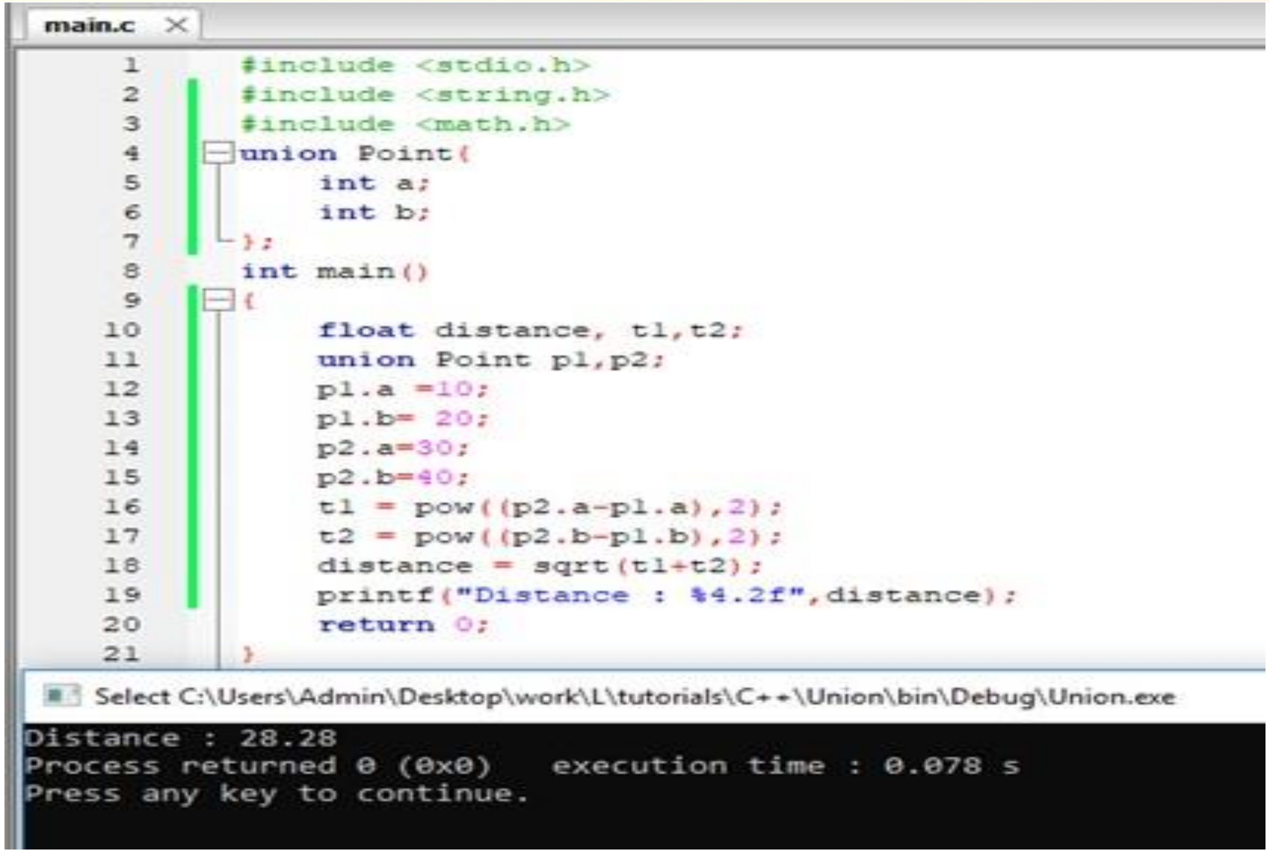
- Size of Union?

The memory occupied by a union will be large enough to hold the largest member of the union.

- Accessing Union Members.

*To access any member of a union, we use the **member access operator (.)**.*

User case?



main.c

```c
#include <stdio.h>
#include <string.h>
#include <math.h>
union Point{
     int a;
     int b;
};
int main()
{
     float distance, t1,t2;
     union Point p1,p2;
     p1.a =10;
     p1.b= 20;
     p2.a=30;
     p2.b=40;
     t1 = pow((p2.a-p1.a),2);
     t2 = pow((p2.b-p1.b),2);
     distance = sqrt(t1+t2);
     printf("Distance : %4.2f",distance);
     return 0;
}
```

Select C:\Users\Admin\Desktop\work\L\tutorials\C++\Union\bin\Debug\Union.exe

```
Distance : 28.28
Process returned 0 (0x0)   execution time : 0.078 s
Press any key to continue.
```

Enum variable

- What is enum?
- Define enum.
- Declare enum variables.
- Size of enum?
- User case?

Enum variable: What is enum?

Enumeration is a user defined datatype in C language. It is used to assign names to the integral constants which makes a program easy to read and maintain. The keyword “enum” is used to declare an enumeration.

(wiki)

Enum variable

- Define enum:

The syntax of enum in C language:

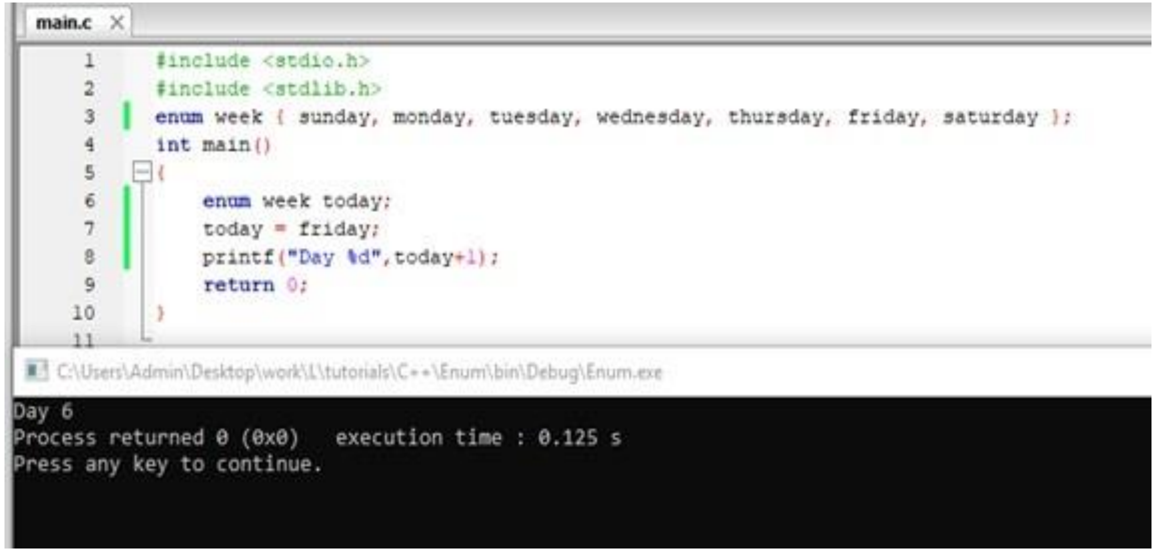
enum enum_name{const1, const2, ... };

```
/*! @brief GPIO direction definition*/
typedef enum _gpio_pin_direction {
   kGpioDigitalInput  = 0U,
   kGpioDigitalOutput = 1U
} gpio_pin_direction_t;
```

Enum variable

- Declare enum variables:

enum enum_name enum_variable;



main.c

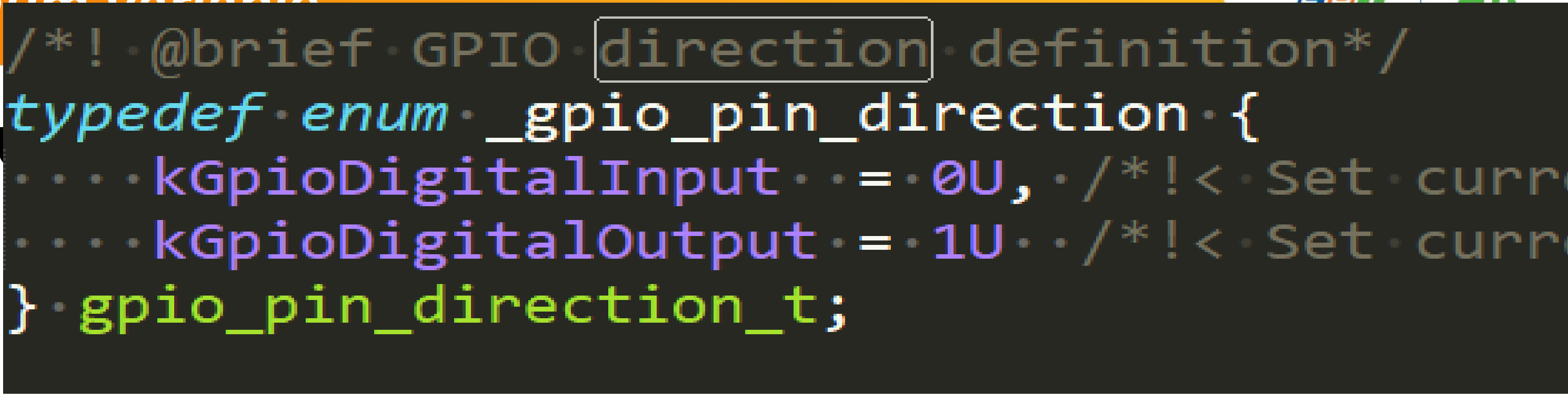
```c
#include <stdio.h>
#include <stdlib.h>
enum week { sunday, monday, tuesday, wednesday, thursday, friday, saturday };
int main()
{
    enum week today;
    today = friday;
    printf("Day %d",today+1);
    return 0;
}
```

C:\Users\Admin\Desktop\work\L\tutorials\C++\Enum\bin\Debug\Enum.exe

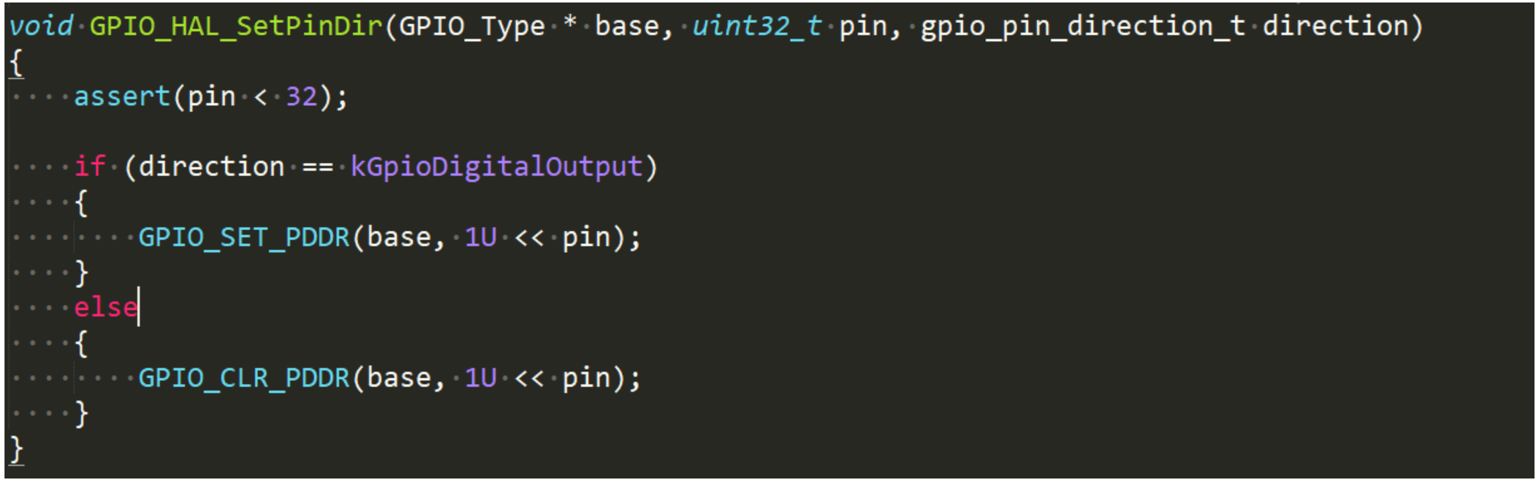
```
Day 6
Process returned 0 (0x0)   execution time : 0.125 s
Press any key to continue.
```

Enum variable

▪ U



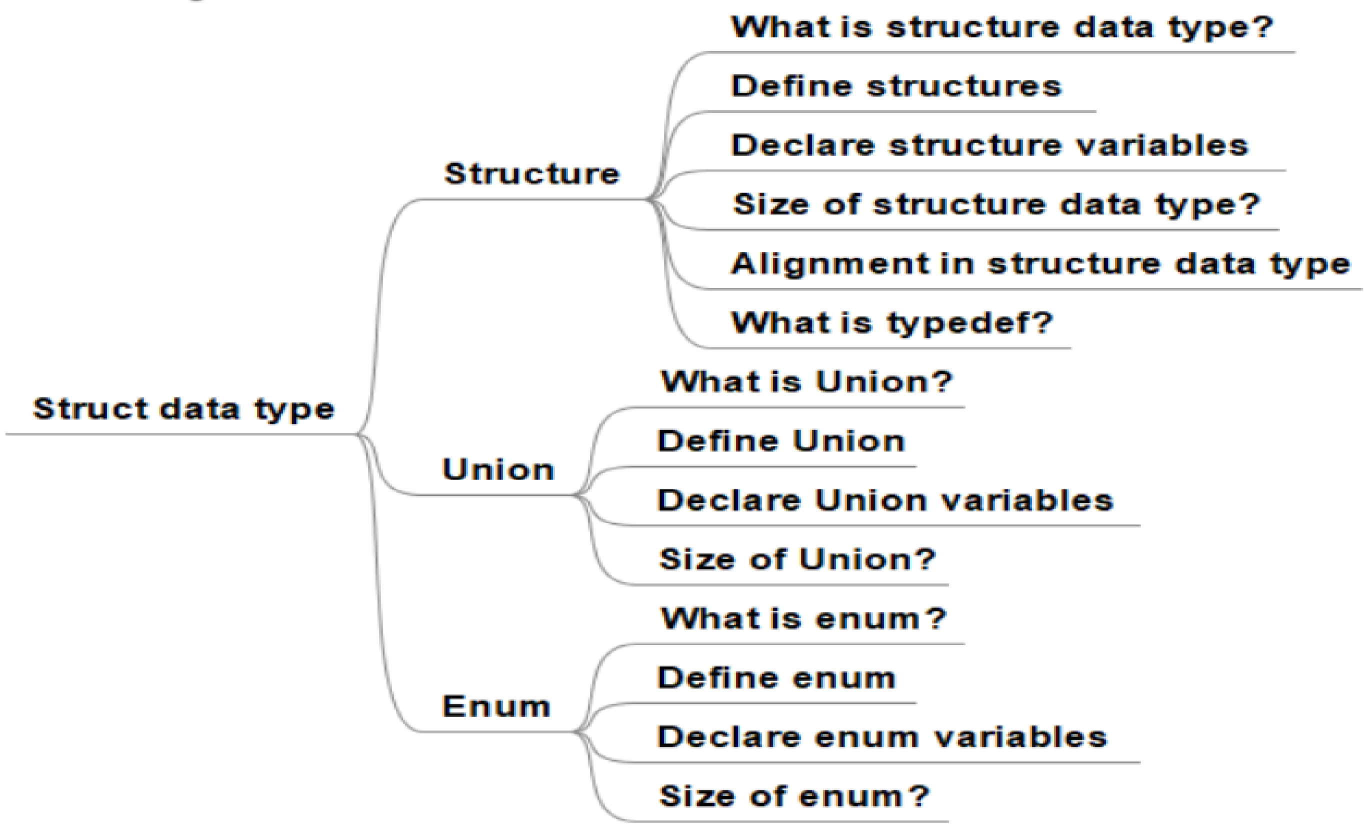
```
/*! @brief GPIO direction definition*/
typedef enum _gpio_pin_direction {
    kGpioDigitalInput  = 0U, /*!< Set curr
    kGpioDigitalOutput = 1U  /*!< Set curr
} gpio_pin_direction_t;
```



```
void GPIO_HAL_SetPinDir(GPIO_Type * base, uint32_t pin, gpio_pin_direction_t direction)
{
    assert(pin < 32);

    if (direction == kGpioDigitalOutput)
    {
        GPIO_SET_PDDR(base, 1U << pin);
    }
    else
    {
        GPIO_CLR_PDDR(base, 1U << pin);
    }
}
```

Structure data type. Summary



Section 6

CASTING

Type casting

- ☐ All objects in C have specified type
  - ✓ Type variable char, int, float, double, …
  - ✓ Pointers point to type char, int, float, double, …
- ☐ Expression with many types
  - ✓ C language automatic cast the types (casting).
  - ✓ User cast the types.

Type casting

- ☐ Increase level (data type) in expression
  - ✓ Elements with the same type
    - The result is general type
    - int / int ➔ int, float / float ➔ float
    - Example: 2 / 4 ➔ 0, 2.0 / 4.0 ➔ 0.5
  - ✓ Elements with the diffirent type
    - The result is cover type
    - char < int < long < float < double
    - float / int ➔ float / float, …
    - Example: 2.0 / 4 ➔ 2.0 / 4.0 ➔ 0.5
    - Note: temporary casting

Type casting

- ☐ Assign <left expression> = <right expression>;
  - ✓ The right expression is increased level (or reduced level) temporary as the same type with right expression type.

```
int i;
float f = 1.23;

i = f;  // ➔ f temporary is int
f = i;  // ➔ i temporary is float
```

  - ✓ May be the accurate of real will be lost ➔ limited!

```
int i = 3;
float f;
f = i;  // ➔ f = 2.999995
```

Type casting

- ☐ Meaning
  - ✓ Type casting to avoid wrong result.
- ☐ Syntax

```
(<new type>)<expression>
```

- ☐ Example

```
int x1 = 1, x2 = 2;
float f1 = x1 / x2;       // ➔ f1 = 0.0
float f2 = (float)x1 / x2; // ➔ f2 = 0.5
float f3 = (float)(x1 / x2);  // ➔ f3 = 0.0
```

Lesson Summary

- Introductory question
- Basic Data Types
- Store Class
- Key words for variable
- Struct data type
- Casting

Thank you