2023/2024(2)
EF234201 Data Structure

Lecture #5

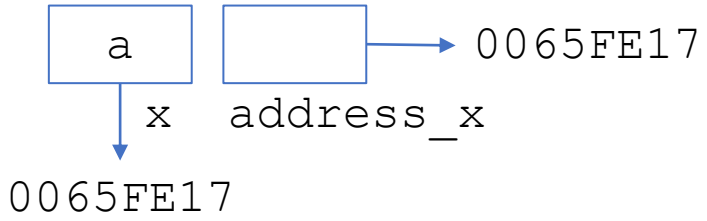
Pointer & Function

Misbakhul Munir IRFAN SUBAKTI
司馬伊凡
Мисбакхул Мунир Ирфан Субакти

Pointer: The Definition

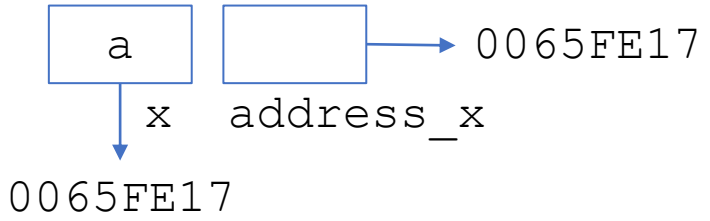
- A pointer is a (**pointer**/pointing/indexing) **variable**, containing a value that designates the address of a particular memory location
- So, the pointer does **not contain data values**, but instead contains a **memory address** or is **null** if it does not contain data
- Uninitialized pointers are called **dangling pointers**
- The memory location can be represented by a **variable** or can also be a **direct memory address** value

Pointer: The Illustration



- We have a variable `x` which contains the character value '`a`'
- In the C compiler, the value '`a`' will be stored at a certain address in memory
- The address of variable `x` can be accessed using the `&x` statement
- If we want to store the address of this variable `x`, we can use a variable
  - E.g., `char address_x = &x;`
- `address_x` is a variable that contains the address where the value `x`, namely '`a`' is stored
- The `address_x` variable is called a pointer variable or often just called a **pointer**

Pointer: Program Example




```c
#include <stdio.h>

void main() {
    char *address_x;
    char x;
    x = 'a';
    address_x = &x;
    printf("x has value %c, stored in address %p or in hexadecimal %x",
        x, address_x, address_x);
}
```

- Format `%p` is used to show the pointer address

```
x has value a, stored in address 000000000065FE17 or in hexadecimal 65fe17
```

Pointer vs Regular Variable

Regular Variable	Pointer
It contains the data/value	It contains the address of particular variable
Operations: it uses regular operator as `+`, `−`, `*`, `/`	1. It needs a special operator `&` for pointing the address of the particular variable. `&` operator can only be applied to a variable. And it results in its address. E.g., `p = &a;` 2. Operator `*`. This operator uses the value from the address variable pointed by its variable. E.g., `int *p;`
Static	Dynamic
Declaration: `int a;`	Declaration: `int *a;`

Pointer Operator

- Operator *
  - To get the value of a pointer variable
  - E.g.,

```
int *address;
int value = 10;
address = &value;
printf("%d", *address);
```

→ Result: 10

- Operator &
  - To get the memory address of the pointer variable
  - E.g.,

```
int *address;
int value = 10;
address = &value;
printf("%p", address);
```

→ Result: 33FF50

Example

- A pointer is declared by

```
data_type *pointer_variable_name;
```

- E.g., pointer initialisation

```c
#include <stdio.h>

void main() {
    float value;
    float *address = &value;
    value = 13.4;
    printf("The value of %.2f stored in memory address %p or %x in hexadecimal\n",
        value, address, address);
    printf("The value of pointer variable is %.2f", *address);
}
```

```
The value of 13.40 stored in memory address 000000000065FE14 or 65fe14 in hexadecimal
The value of pointer variable is 13.40
```

Rule

- A pointer variable can be declared with **any data type**
- Declaring a pointer variable with a certain data type is used to store a memory address that contains data according to the declared data type, **not to contain values** of a certain data type
- The data type is used as the data width for memory allocation (for example char means the data width is 1 byte, etc.)
  - If a pointer variable is declared to be of type float, it means that the pointer variable **can only** be used to point to a memory address that contains a value of type float as well

Warning Example

main() : void

```c
#include <stdio.h>

void main() {
    long int value = 202320242;
    int *warning_address;
    warning_address = &value;
    printf("The value is %ld", *warning_address);
}
```

Compiler (2) | Resources | Compile Log | Debug | Find Results | Console | Close

Line	Col	File	Message
		D:\Main\Course\2024\02 Data Structure\Pr...	**In function 'main':**
6	18	D:\Main\Course\2024\02 Data Structure\Progra...	[Warning] assignment to 'int *' from incompatible pointer type 'long int *' [-Wincompatible-pointer-types]

```
The value is 202320242
```

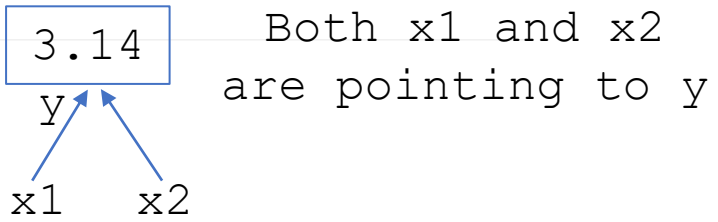
Operations on Pointer: Assignment

- Between pointer variables, assignment operations can be carried out
  - Ex. 1: Assignment and an address can be pointed to by more than one pointer
  - Ex. 2: Filling a variable with the value pointed to by a pointer variable
  - Ex. 3: Operate on the contents of a variable by calling its address with a pointer
  - Ex. 4: Filling and replacing the variable pointed to by the pointer

Ex. 1: Assignment and an address can be pointed to by more than one pointer

```c
#include <stdio.h>

void main() {
    float y, *x1, *x2;
    y = 3.14;
    x1 = &y;
    x2 = x1; // Assignment operation
    printf("The value of y pointed by x1 is %2.2f at the address of %p\n", y, &y);
    printf("The value of y pointed by x2 is %2.2f at the address of %p\n", *x2, x2);
}
```



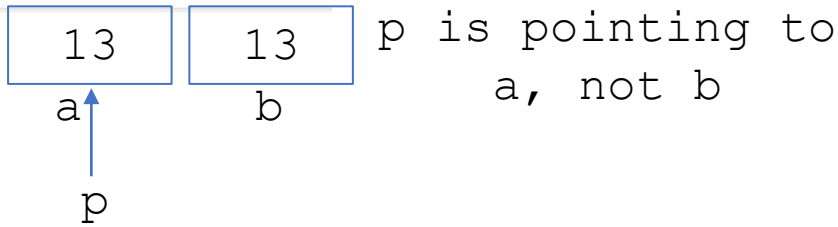
Both x1 and x2 are pointing to y

```
The value of y pointed by x1 is 3.14 at the address of 000000000065FE0C
The value of y pointed by x2 is 3.14 at the address of 000000000065FE0C
```

Ex. 2: Filling a variable with the value pointed to by a pointer variable

```c
#include <stdio.h>

void main() {
    int *p;
    int a = 13;
    int b;
    p = &a;
    b = *p;
    printf("The value of a = %d at the address of %p\n", a, p);
    printf("The value of b = %d at the address of %p\n", b, p);
}
```



p is pointing to a, not b

```
The value of a = 13 at the address of 000000000065FE10
The value of b = 13 at the address of 000000000065FE10
```

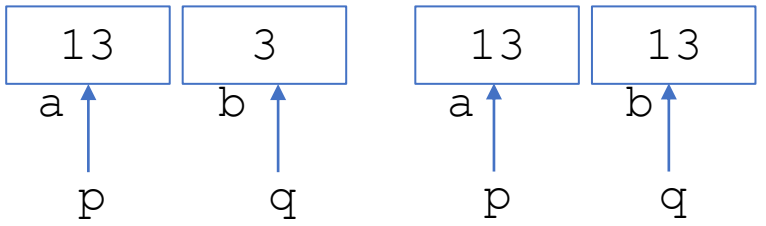
Ex. 3: Operate on the contents of a variable by calling its address with a pointer

```c
#include <stdio.h>

void main() {
    int a = 13;
    int b = 3;
    int *p;
    int *q;
    p = &a;
    q = &b;
    printf("p = &a\n");
    printf("q = &b\n");
    printf("The value pointed by p = %d at the address of %p\n", *p, p);
    printf("The value pointed by q = %d at the address of %p\n", *q, q);
    *q = *p;
    printf("\nAfter assigning *q = *p\n");
    printf("The value pointed by p = %d at the address of %p\n", *p, p);
    printf("The value pointed by q = %d at the address of %p", *q, q);
}
```

```
p = &a
q = &b
The value pointed by p = 13 at the address of 000000000065FE0C
The value pointed by q = 3 at the address of 000000000065FE08

After assigning *q = *p
The value pointed by p = 13 at the address of 000000000065FE0C
The value pointed by q = 13 at the address of 000000000065FE08
```



Ex. 4: Filling and replacing the variable pointed to by the pointer

```c
#include <stdio.h>

void main() {
    int a;
    int *p;
    p = &a;
    *p = 13;
    printf("The value of a = %d", a);
}
```

```
The value of a = 13
```

Operations on Pointer: Arithmetic

- On pointer, arithmetic operations can be performed that will point to a new memory address
- Only **integer** values can be operated on pointer variables
- Usually only **addition/subtraction** operations
- For instance, if the pointer `x` is of type `int` (2 bytes), then `x + 1` will point to the current address (e.g., `1000`) added by `sizeof(x)`, i.e., `2`, resulting in `1002`
- See the next program example

Program: Arithmetic operations on pointer

```c
#include <stdio.h>
void main() {
    char s[] = "Danaya";
    char *p;

    // 1st method
    p = s; // Directly pointing to the array's name

    // 2nd method
    // p = &s[0]; // Pointing to the first character address of array

    printf("The value of s = ");
    for (int i = 0; i < 6; i++) {
        printf("%c", *p);
        p++;
    }
    // Try this, what is the result?
    printf("\n\nTest 1. The value of s = ");
    for (int i = 0; i < 6; i++) {
        printf("%c", *p);
        p++;
    }
    // How about this?
    printf("\n\nTest 2. The value of s = ");
    for (int i = 0; i < 6; i++) {
        printf("%c", *p);
        p--;
    }
    // And this?
    printf("\n\nTest 3. The value of s = ");
    p = s;
    for (int i = 0; i < 6; i++) {
        printf("%c", *p);
        p++;
    }
}
```

```
The value of s = Danaya

Test 1. The value of s = H

Test 2. The value of s = H

Test 3. The value of s = Danaya
```

Pointer on Array

- In an array, the pointer only needs to point to the address of the **first element** because the array addresses are already sequential in memory.
- Pointer variables only need to **increment**
- See the next examples

1D Array

```c
#include <stdio.h>

void main() {
    int a[5] = {1, 2, 3, 4, 5};
    int *p;
    p = a;
    printf("The first p = %d\n", *p);
    p = a + 1;
    printf("The next p = %d\n", *p);
    p = a + 2;
    printf("The next p = %d\n", *p);
    p = a + 3;
    printf("The next p = %d\n", *p);
    p = a + 4;
    printf("The next p = %d", *p);
}
```

```
The first p = 1
The next p = 2
The next p = 3
The next p = 4
The next p = 5
```

- `p = a` means pointer `p` has assigned the address of array `a`. The address can be represented by the first element, i.e., `a[0]`
- It also can be written as `p = &a[0]` which means the same as `p = a`

1D Array (continued)

```c
#include <stdio.h>

void main() {
    int a[5] = {1, 2, 3, 4, 5};
    int *p;

    p = a; // Reset
    printf("The value of a = ");
    for (int i = 0; i < 5; i++) {
        printf("%d ", *p);
        p++;
    }

    p = &a[0]; // Reset
    printf("\n\nUpdate the items...\n");
    for (int i = 0; i < 5; i++) {
        *p = i * 10;
        p++;
    }

    p = a; // Reset
    printf("\nThe value of a = ");
    for (int i = 0; i < 5; i++) {
        printf("%d ", *p);
        p++;
    }
}
```

```
The value of a = 1 2 3 4 5

Update the items...

The value of a = 0 10 20 30 40
```

Function: Review

- A function is a part of a program that has a unique **name**
- It is used to do a certain task
- It is located **separately** from the part of the program that uses/calls the function

Function: Advantage

- Can use a **top-down** and **divide-and-conquer** approach: large programs can be split into small programs
- Can be done by several people so coordination is easy
- Ease of finding errors because the logic flow is clear and errors can be localized within a particular module
- Program modifications can be made to a particular module only without disturbing the overall program
- Makes documentation easier
- **Reusability**: A function can be reused by other programs or functions

Function in C: Review

- Standard Library Functions
  - These are functions that have been provided by C in its header or library files
  - For instance: `printf()`, `getch()`
  - For this function we must first declare the library that will be used, namely by using the directive preprocessor: `#include <stdio.h>`, `#include <conio.h>`
- Programmer-Defined Function
  - A function created by the programmer himself
  - This function has a specific name that is unique to the program
  - It is located separately from the main program, and can be integrated into a library created by the programmer which is then also included for its use

Void Function: Review

- Function that is void are often called procedure
- It is called void because the function does not return an output value obtained from the process of the function
- Characteristic
  - No **return** keyword
  - No **data type** in the function declaration
  - It uses the **void** keyword
- The result cannot be displayed immediately
- It has no function return value
- Example: `printf()`

Non-Void Function: Review

- Non-void function is also called **function**
- It is called non-void because it returns a return value that comes from the output of the function process
- Characteristic
  - There is a **return** keyword
  - There is a **data type** that begins the function declaration
  - No **void** keyword
- It has a return value
- It can be analogous to a variable that has a certain data type so that the results can be displayed immediately
- Example: `sin(),getch()`

`main` Function: Review

- The simplest program in C, **in order to be executed**, must consist of at least 1 function, namely the `main()` function
- When a C program is executed, the C compiler will look for the `main()` function and carry out the instructions there
- It is often declared in two forms:

```
int main()
void main()
```

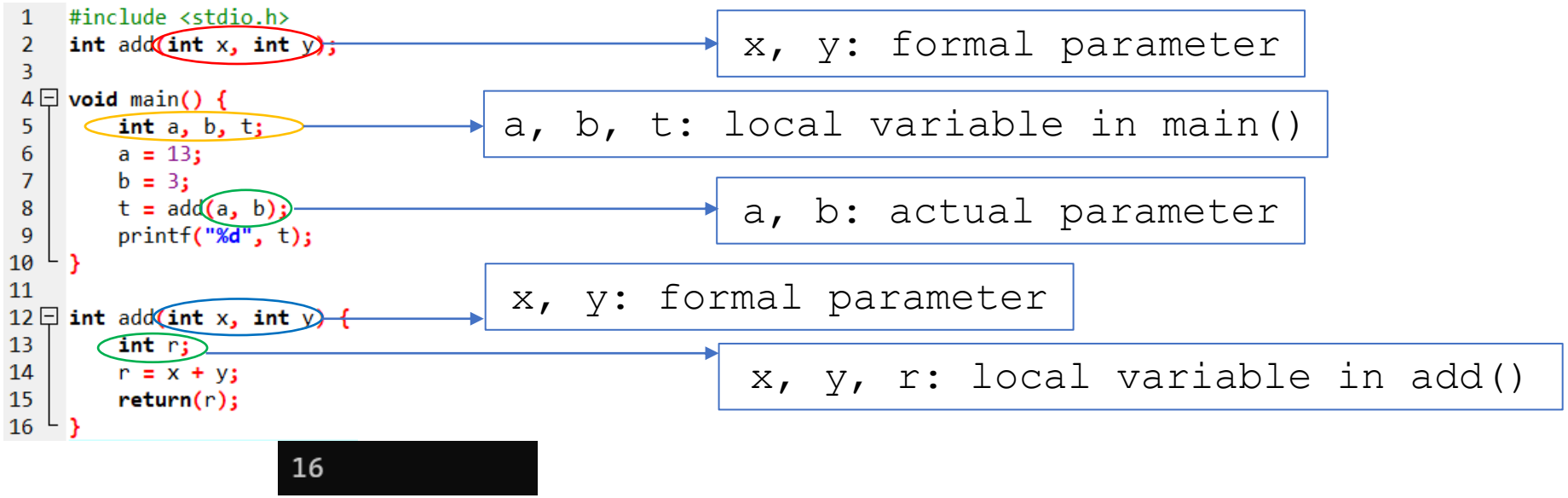
`main` Function Review: `int` and `void`

- `int main()` means that in the main function, there must be a **return** keyword at the end of the function and it returns a value of the data type `int`
- Why does the return result have to be of type `int` too? because the data type that precedes the `main()` function above is declared `int`
- If a C program is executed, the program execution status will be returned, if "*terminated successfully*" then the status will be returned 0, whereas if "*terminated unsuccessfully*" the status value will be returned not 0
- `void main()` means a function that is void and does not return a program status value so the program status value cannot be known

Argument in Function: Review

- A function can have optional arguments
- These arguments function as input parameters in the form of variables for the function (local variable)
- Arguments must be of a specific data type
- There are 2 types of parameters:
  - **Formal** parameters: parameters written in the function declaration
  - **Actual** parameters: parameters entered in the program calling the function. It can be a variable or directly a certain value according to the data type declared for each function parameter

Argument in Function: Example



Variable Scope: Review

- Global variables: known in all parts
- Local variables: known only in certain parts
- Static variable: the value is fixed and the last value will be saved
- The scope above depends on the perspective of a variable

Pass by Value: Review

- What is sent to the function is the **value**, *not* the **memory address** where the data is located
- The function that receives this value will store its value at a **separate address** from the original value used by the program that called the function
- That's why changing the value in the function **will not affect** the original value in the program that calls the function even though both use the same variable name
- **One-way** nature of passing, from the calling program to the called function only.
- Parameters can be expressions (statements)
- See the next example

Pass by Value: Example

```c
#include <stdio.h>

int a = 3;

void aGlobal() {
    printf("The value of a in aGlobal() is %d at the address of %p\n", a, &a);
}

void passByValue(int a) {
    a = a * 2;
    printf("The value of a in passByValue() is %d at the address of %p\n", a, &a);
}

int main() {
    int a = 13;
    aGlobal();
    printf("The value of a in main() is %d at the address of %p\n", a, &a);
    passByValue(a);
    printf("The value of a in main() after passByBalue() called is %d at the address of %p\n", a, &a);
    return 0;
}
```

```
The value of a in aGlobal() is 3 at the address of 0000000000403010
The value of a in main() is 13 at the address of 000000000065FE1C
The value of a in passByValue() is 26 at the address of 000000000065FDF0
The value of a in main() after passByBalue() called is 13 at the address of 000000000065FE1C
```

a in `aGlobal()`	a in `main()`	a in `passByValue()`	a in `main()` after `passByValue()` called
Value: 3	Value: 13	Value: 26	Value: 13
Address: 00403010	Address: 0065FE1C	Address: 0065FDF0	Address: 0065FE1C

Pass by Reference: Review

- What is sent is the **memory address** where the data value is located, *not* the data **value**
- Functions that receive this parameter will use/access data with the **same address** as the address of the data value
- That's why changing the value in the function **will also change** the original value in the program calling the function
- Passing parameters by reference is **two-way** passing, namely from the calling program to the function and vice versa from the function to the calling program
- Passing parameters by reference **cannot be used** for an expression (statement), only for variables, constants or array elements
- See the next example

Pass by Reference: Example

```c
#include <stdio.h>

int a = 3;

void aGlobal() {
    printf("The value of a in aGlobal() is %d at the address of %p\n", a, &a);
}

void passByRef(int *a) {
    *a = *a * 2;
    printf("The value of a in passByRef() is %d at the address of %p\n", *a, a);
}

int main() {
    int a = 13;
    aGlobal();
    printf("The value of a in main() is %d at the address of %p\n", a, &a);
    passByRef(&a);
    printf("The value of a in main() after passByRef() called is %d at the address of %p\n", a, &a);
    return 0;
}
```

It uses asterisk (*)

It uses asterisk (*)

It uses the address of its variable (&)

```
The value of a in aGlobal() is 3 at the address of 0000000000403010
The value of a in main() is 13 at the address of 000000000065FE1C
The value of a in passByRef() is 26 at the address of 000000000065FE1C
The value of a in main() after passByRef() called is 26 at the address of 000000000065FE1C
```

a in `aGlobal()`	a in `main()`	a in `passByRef()`	a in `main()` after `passByRef()` called
Value: 3	Value: 13	Value: 26	Value: 26
Address: 00403010	Address: 0065FE1C	Address: 0065FE1C	Address: 0065FE1C

Array as Parameter

- Passing parameters in the form of an array is passing **by reference**, what is sent is the **address of the first element** of the array, not all the array values
- In formal parameters, the address of the first element of the array can be written as the array name alone without its index (empty index)
- In actual parameters, writing is done by just writing the array name

Array as Parameter: Example

```c
#include <stdio.h>

void fill(int data[]) {
    for (int i = 0; i < 10; i++) {
        data[i] = i + 1;
    }
}

void show(int data[]) {
    for (int i = 0; i < 10; i++) {
        printf("%d ", data[i]);
    }
}

int main() {
    int data[10];
    printf("Fill the data...\n");
    fill(data);
    printf("The data has been filled.\n");
    printf("Show the data...\n");
    show(data);
    return 0;
}
```

```
Fill the data...
The data has been filled.
Show the data...
1 2 3 4 5 6 7 8 9 10
```

Exercise

- Create a function to calculate factorial
- Create a function to calculate the power (x^y)
- Create a function to find out whether a number is a prime number or not, then create a function to display all prime numbers from a certain data range and use the prime number checking function that was created previously

- NEXT:
  - Pointer Implementation: Linked List