A. Lý thuyết

What is a Pointer?

- (1) A pointer is a variable, which contains the address of a memory location of another variable
- (2) If one variable contains the address of another variable, the first variable is said to point to the second variable
- (3) A pointer provides an indirect method of accessing the value of a data item
- (4) Pointers can point to variables of other fundamental data types like int, char, or double or data aggregates like arrays or structures

What are Pointers used for?

Some situations where pointers can be used are

- To return more than one value from a function
- To pass arrays and strings more conveniently from one function to another
- To manipulate arrays easily by moving pointers to them instead of moving the arrays itself
- To allocate memory and access it (Direct Memory Allocation)

Variable characteristics

type name value

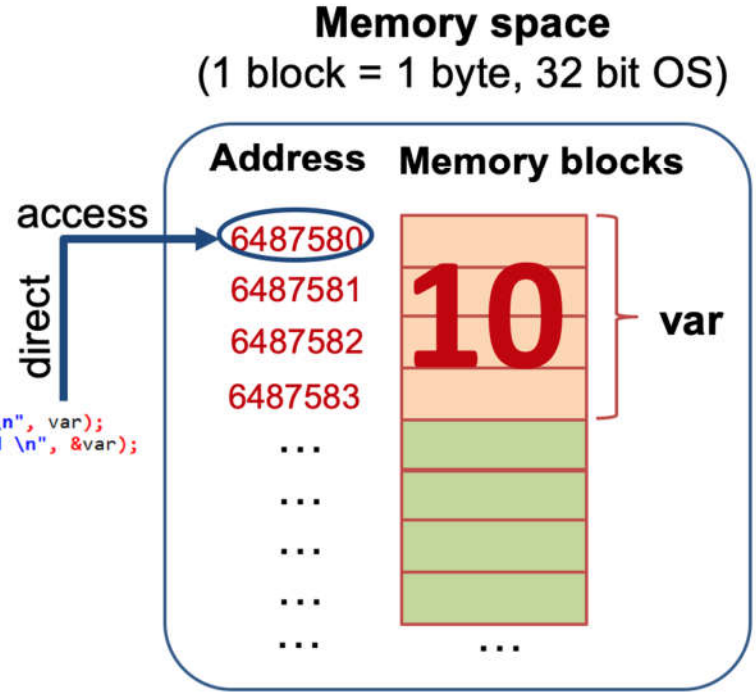
int var = 10;

- Source code:

```c
#include <stdio.h>

int main(){
    int var = 10;

    printf("var's value: %d \n", var);
    printf("var's address: %d \n", &var);

    return 0;
}
```

- Result:

```
var's value: 10
var's address: 6487580
```



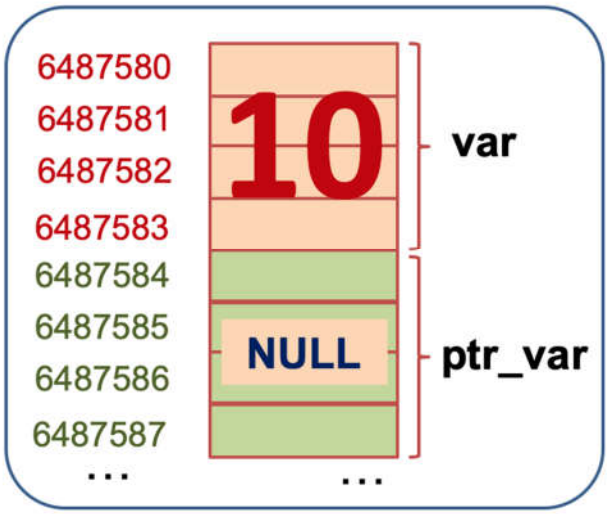
Pointer Variables

General declaration syntax is :

```
datatype *name;
```

Example:

```
int var = 10;
int *ptr_var;
```



Pointer Operators - 1

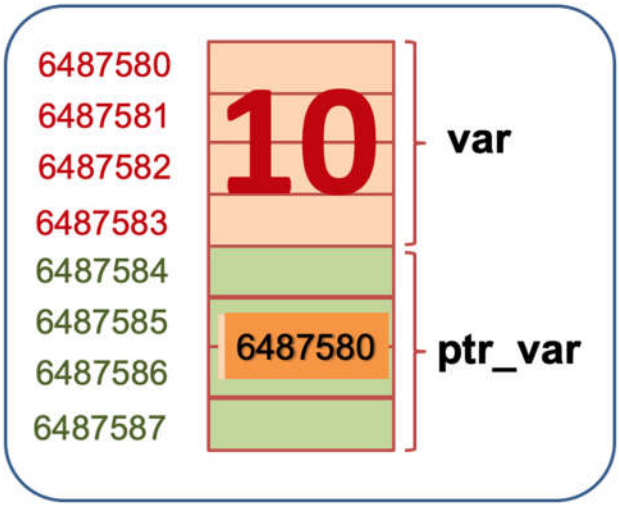
- There are 2 special unary operators which are used with pointers

& and *****

- The **&** operator used to returns the memory address of the operand

Example:

```
int var = 10;
int *ptr_var;
ptr_var = &var;
```



Pointer Operators - 2

- The * operator used to returns the value contained in the memory location pointed to by the pointer variable's value

temp = *ptr_var;

Example:

```
int var = 10, temp;
int *ptr_var = &var;
temp = *ptr_var;
```

temp = 10;

Assigning Values To Pointers-1

- Values can be assigned to pointers through the **&** operator.

```
ptr_var = &var;
```

- Here the address of var is stored in the variable ptr_var
- It is also possible to assign values to pointers through another pointer variable pointing to a data item of the same data type

```
ptr_var = &var;
ptr_var2 = ptr_var;
```

Assigning Values To Pointers-2

- Variables can be assigned values through their pointers as well

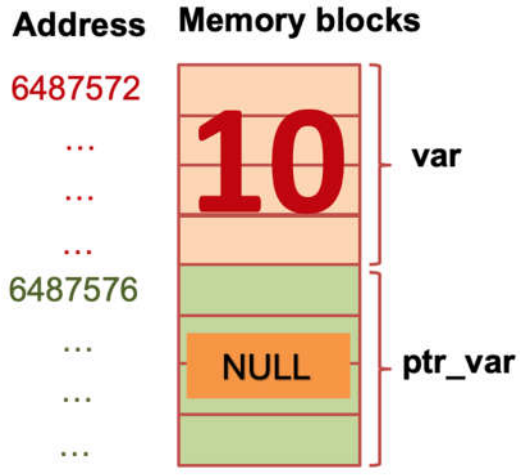
***ptr_var = 10;**

- The above declaration will assign 10 to the variable var if ptr_var points to var

Pointer Example - 1

```c
#include <stdio.h>
int main(){
    int var = 10;
    int *ptr_var;
    ptr_var = &var;
    printf("var's value: %d \n", var);
    printf("var's address: %d \n", &var);
    printf("ptr_var's value: %d \n", ptr_var);
    printf("Value in memory pointed by ptr_var: %d \n", *ptr_var);
    *ptr_var = 20;
    printf("var's value after change: %d", var);
    return 0;
}
```

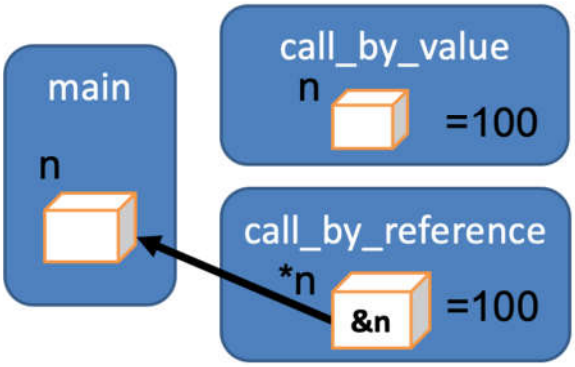
```
var's value: 10
var's address: 6487572
ptr_var's value: 6487572
Value in memory pointed by ptr_var: 10
var's value after change: 20
```



Pointer Example - 2

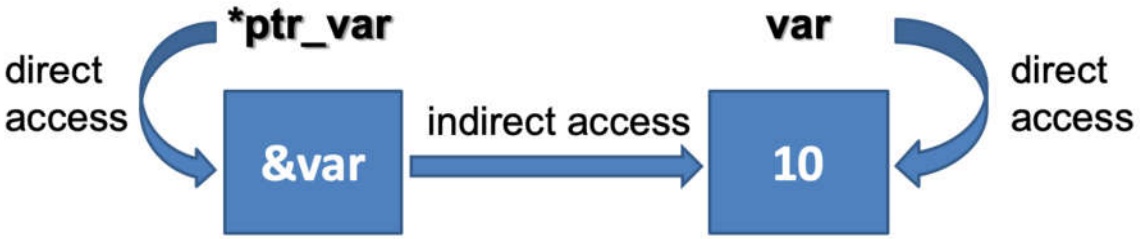
```c
#include <stdio.h>

void call_by_value(int n);
void call_by_reference(int *n);

int main(){
    int n = 0;
    call_by_value(n);
    printf("n's value after call_by_value: %d \n", n);
    call_by_reference(&n);
    printf("n's value after call_by_reference: %d \n", n);
    return 0;
}

void call_by_value(int n){
    n = 100;
}

void call_by_reference(int *n){
    *n = 100;
}
```

```
n's value after call_by_value: 0
n's value after call_by_reference: 100
```



NOTE - 1

- A pointer provides an indirect method of accessing the value of a data item



NOTE - 2

- The size of the pointer variable is not related to the type of variable it points to, because it depends on types the operating system

```c
#include <stdio.h>

int main(){
    char    c = 'a';
    int     n = 1;
    double  d = 0.5;
    char    *pc = &c;
    int     *pn = &n;
    double  *pd = &d;

    printf("Size of c, n, d: %d byte, %d byte, %d byte \n",
           sizeof(c), sizeof(n), sizeof(d));
    printf("Size of pc, pn, pd: %d byte, %d byte, %d byte \n",
           sizeof(pc), sizeof(pn), sizeof(pd));
    return 0;
}
```

Result:

```
Size of c, n, d: 1 byte, 4 byte, 8 byte
Size of pc, pn, pd: 8 byte, 8 byte, 8 byte
```

Pointer Arithmetic-1

- Addition and subtraction are the only operations that can be performed on pointers

```
int var, *ptr_var;
ptr_var = &var;
var = 500;
ptr_var++ ;
```

- Let us assume that **var** is stored at the address **1000**
- Then ptr_var has the value 1000 stored in it. Since integers are 2 bytes long, after the expression "ptr_var++;" ptr_var will have the value as 1002 and not 1001

Example – Pointer Arithmetic

```
pc, pn, pd with +0: 6487559, 6487552, 6487544

pc, pn, pd with +1: 6487560, 6487556, 6487552

pc, pn, pd with +0: 6487561, 6487560, 6487560
```

```c
#include <stdio.h>

int main(){
    char    c = 'a';
    int     n = 1;
    double  d = 0.5;
    char    *pc = &c;
    int     *pn = &n;
    double  *pd = &d;

    printf("\n\npc, pn, pd with +0: %d, %d, %d", pc, pn, pd);
    printf("\n\npc, pn, pd with +1: %d, %d, %d", pc+1, pn+1, pd+1);
    printf("\n\npc, pn, pd with +0: %d, %d, %d", pc+2, pn+2, pd+2);
    return 0;
}
```

Pointer Arithmetic - 2

++ptr_var or ptr_var++	points to next **integer** after var
—ptr_var or ptr_var—	points to **integer** previous to var
ptr_var + i	points to the ith integer after var
ptr_var - i	points to the ith integer before var
++*ptr_var or (*ptr_var)++	will increment **var** by 1
*ptr_var++	will fetch the value of the next integer after var

- Each time a pointer is incremented, it points to the memory location of the next element of its base type
- Each time it is decremented it points to the location of the previous element
- All other pointers will increase or decrease depending on the length of the data type they are pointing to

Pointer Comparisons

- Two pointers can be compared in a relational expression provided both the pointers are pointing to variables of the same type
- Consider that ptr_a and ptr_b are 2 pointer variables, which point to data elements a and b. In this case the following comparisons are possible:

ptr_a < ptr_b	Returns true provided **a** is stored before **b**
ptr_a > ptr_b	Returns true provided **a** is stored after **b**
ptr_a <= ptr_b	Returns true provided **a** is stored before **b** or ptr_a and ptr_b point to the same location
ptr_a >= ptr_b	Returns true provided **a** is stored after **b** or ptr_a and ptr_b point to the same location.
ptr_a == ptr_b	Returns true provided both pointers ptr_a and ptr_b points to the same data element.
ptr_a != ptr_b	Returns true provided both pointers ptr_a and ptr_b point to different data elements but of the same type.
ptr_a == NULL	Returns true if ptr_a is assigned NULL value (zero)

Pointer and one-dimensional array

Source code:

```c
#include <stdio.h>
int main(){
    int arr[5] = {6, 5, 6, 7, 8};

    printf("value of arr: %d \n", arr);
    printf("address of arr[0]: %d\n", &arr[0]);

    return 0;
}
```

Result:

```
value of arr: 6487552
address of arr[0]: 6487552
```

Source code:

```c
#include <stdio.h>
int main(){
    int arr[5] = {6, 5, 6, 7, 8};
    int *p_arr = arr; //or: int *p_arr = &arr[0]
    int i;

    for(i=0; i<5; i++){
        printf("*(arr+%d) = %d, p_arr[%d]= %d \n",i, *(arr+i), i, p_arr[i]);
    }
    return 0;
}
```

Result:

```
*(arr+0) = 6, p_arr[0]= 6
*(arr+1) = 5, p_arr[1]= 5
*(arr+2) = 6, p_arr[2]= 6
*(arr+3) = 7, p_arr[3]= 7
*(arr+4) = 8, p_arr[4]= 8
```

Pointer and one-dimensional array

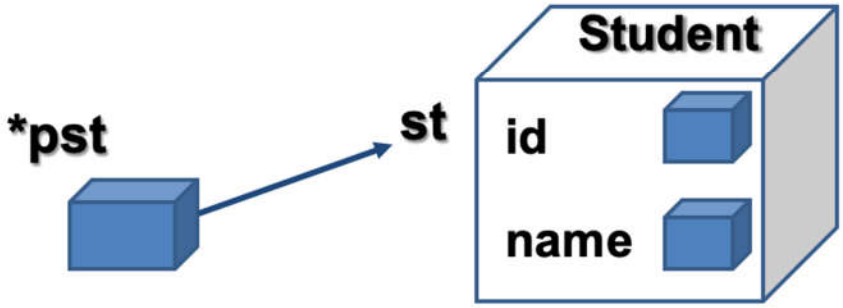
- In C language, array names can be thought of as a constant pointer type
- The address of the array matches the first element address of the array
- Use unary operators (++, --) to move between elements in the array pointed to by the pointer

Pointer and one-dimensional array

```c
#include <stdio.h>

void input_array(int *p, int size);
void output_array(int *p, int size);
int add_array(int *p, int size);

int main(){
    int scores[20], n;

    printf("Enter the number of Element in scores array: ");
    scanf("%d", &n);

    input_array(scores,n);
    output_array(scores,n);
    printf("\nTotal value of the elements: %d", add_array(scores,n));
    return 0;
}

void input_array(int *p, int size){

void output_array(int *p, int size){

int add_array(int *p, int size){
```

Pointer variables type Struct

```c
#include <stdio.h>
struct Student{
    char id[10];
    char name[40];
};

int main(){
    struct Student st = {"ST001", "Pham Ngoc Tho"};
    struct Student *pst;
    pst = &st;

    printf("ID: #%s | Name: %s\n", st.id, st.name);
    printf("ID: #%s | Name: %s\n", pst->id, pst->name);
    return 0;
}
```



B. Bài tập

Ex1.
Write a short C program that declares and initializes (to any value you like) a double, an int, and a char. Next declare and initialize a pointer to each of the three variables. Your program should then print the address of, and value stored in, and the memory size (in bytes) of each of the six variables.

Use the “0x%x” formatting specifier to print addresses in hexadecimal. You should see addresses that look something like this: "0xbfe55918". The initial characters "0x" tell you that hexadecimal notation is being used; the remainder of the digits give the address itself.

Use “%f” to print a floating value. Use the memory size allocated for each variable.
sizeof operator to determine the

Sample output:

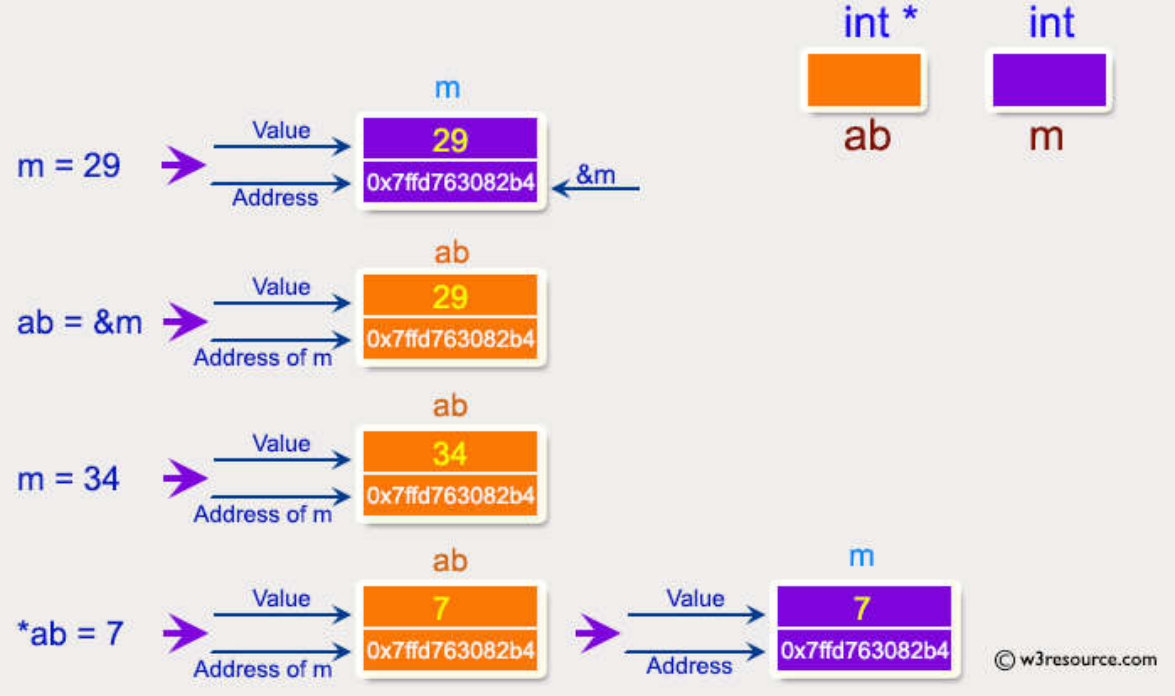
```
The address of char     ___ is 0x________
The address of int      ___ is 0x________
The address of double   ___ is 0x________
The address of char*    ___ is 0x________
The address of int*     ___ is 0x________
The address of double*  ___ is 0x________

The value of char       ___ is   ________
The value of int        ___ is   ________
The value of double     ___ is   ________
The value of char*      ___ is 0x________
The value of int*       ___ is 0x________
The value of double*    ___ is 0x________

The size of char        is   ________ bytes
The size of int         is   ________ bytes
The size of double      is   ________ bytes
The size of char*       is   ________ bytes
The size of int*        is   ________ bytes
The size of double*     is   ________ bytes
```

EX2.

Write a program in C to demonstrate how to handle the pointers in the program

---

Pictorial Presentation:



Expected output:

```
Address of m : 0x7ffcc3ad291c
Value of m : 29

Now ab is assigned with the address of m.
Address of pointer ab : 0x7ffcc3ad291c
Content of pointer ab : 29

The value of m assigned to 34 now.
Address of pointer ab : 0x7ffcc3ad291c
Content of pointer ab : 34

The pointer variable ab is assigned with the value 7 now.
Address of m : 0x7ffcc3ad291c
Value of m : 7
```

Ex3.

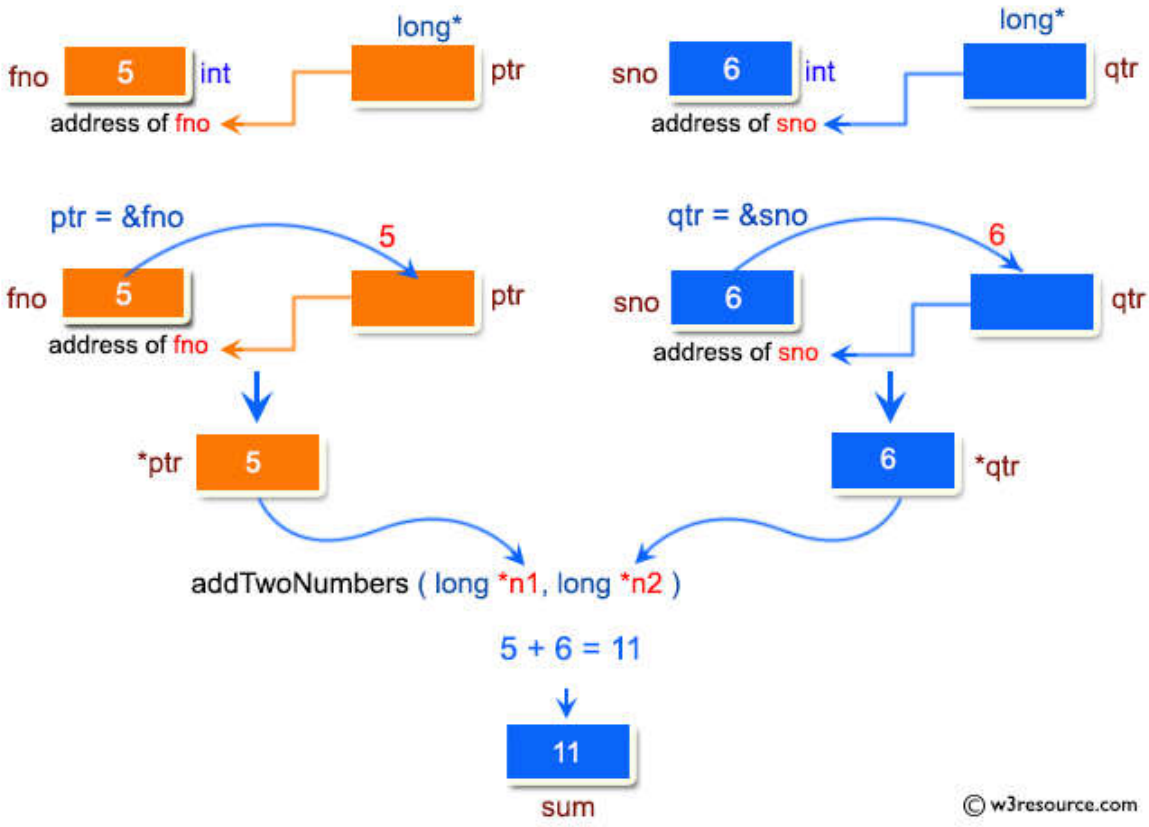
swap_nums seems to work, but not swap_pointers. Fix it.

```c
#include <stdio.h>

void swap_nums(int *x, int *y)
{
  int tmp;

  tmp = *x;
  *x = *y;
  *y = tmp;
}

void swap_pointers(char *x, char *y)
{
  char *tmp;

  tmp = x;
  x = y;
  y = tmp;
}

int main()
{
  int a,b;
  char *s1,*s2;

  a = 3; b=4;
  swap_nums(&a,&b);
  printf("a is %d\n", a);
  printf("b is %d\n", b);

  s1 = "I should print second";
  s2 = "I should print first";
  swap_pointers(s1,s2);
  printf("s1 is %s\n", s1);
  printf("s2 is %s\n", s2);

  return 0;
}
```

Ex4.

Write a program in C to add numbers using call by reference

Pictorial Presentation:



Sample output:

```
Pointer : Add two numbers using call by reference:
------------------------------------------------------
 Input the first number : 5
 Input the second  number : 6
 The sum of 5 and 6  is 11
```

EX5.

Write a program, use the pointer to enter a sequence of integers consist N element. Ouput to screen:

- Maximum value of the element in array
- Minimum value of the element in array
- Sum the elements in array