

Class 9 Pointer-1

Variables & Memory Address

- Each Variable is stored in the memory, and a unique address indicates where it is stored
- The address is a 32bit number in a 32bit computer, and a 64bit number in a 64bit computer
- The address of a variable can be obtained by using operator **&**

```
int a = 5, b = 10;  
  
printf(“%d %d\n”, a, &a);  
printf(“%d %d\n”, b, &b);
```

	.	
	.	
	.	
0x1F3D4A60	5	a
0x1F3D4A64	10	b
	.	
	.	
	.	

Pointer(指针)

A **Pointer** points to
its corresponding **variable**

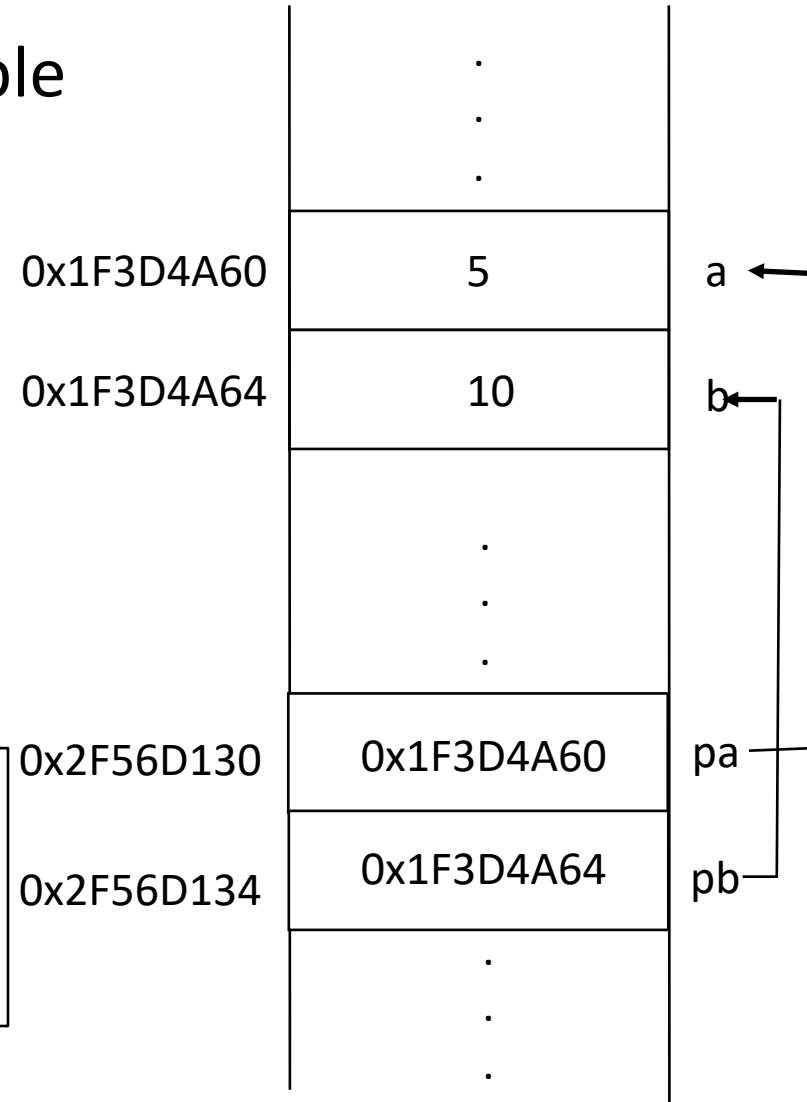
- Pointer is a special type of variable that contains the address of a variable

- Definition of a Pointer Variable

```
type *name;
```

- Examples

```
int a = 5, b = 10;  
  
int *pa = &a; //pa = 0x1F3D4A60  
int *pb = &b; //pb = 0x1F3D4A64
```



Pointer & Dereference Operator

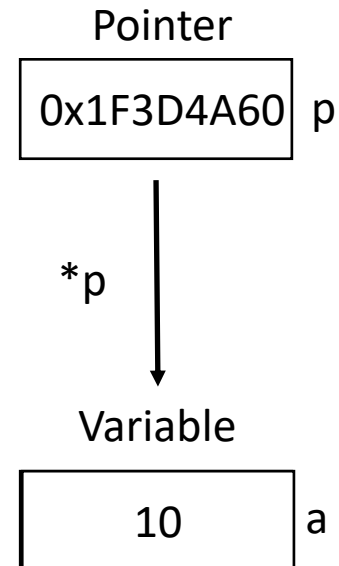
- The unary operator `*` is called a dereference operator (间接访问运算符)

`*pointer_name`

- When the operator `*` is used with a pointer, the data of the corresponding variable is accessed

```
int a;    // variable
int *p;   // pointer
a = 5;    // set the value for a
p = &a;   // make p point to a
printf("%d %d\n", a, *p);

*p = 10;  // change the value of a
printf("%d %d\n", a, *p);
```



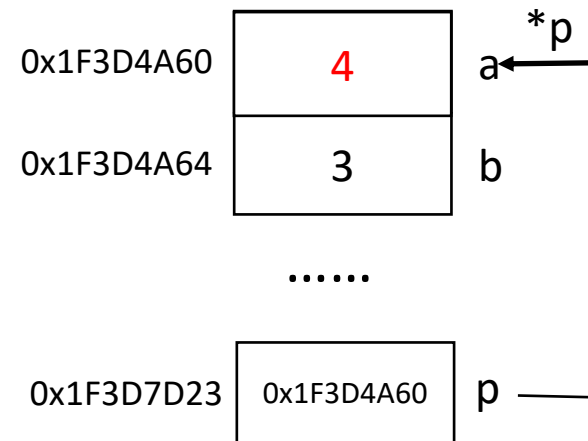
Example



```
int a = 1, b = 3;  
int *p = &a;  
  
*p = 4;  
a = a + 1;  
printf("%d %d\n", a, *p);  
  
*p = *p + 1;  
printf("%d %d\n", a, *p);  
  
p = &b;  
printf("%d %d\n", a, *p);  
  
p = p + 1;  
printf("%d %d\n", a, *p);
```

Output:

5	5
6	6
6	3
6	-82119334



Example

```
int a = 1, b = 3;
int *p = &a;



→

*p = 4;
a = a + 1;
printf("%d %d\n", a, *p);

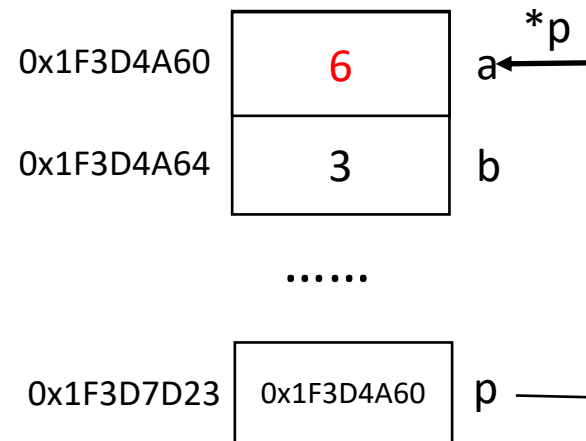
*p = *p + 1;
printf("%d %d\n", a, *p);

p = &b;
printf("%d %d\n", a, *p);

p = p + 1;
printf("%d %d\n", a, *p);
```

Output:

5	5
6	6
6	3
6	-82119334



Example

```
int a = 1, b = 3;
int *p = &a;

*p = 4;
a = a + 1;
printf("%d %d\n", a, *p);

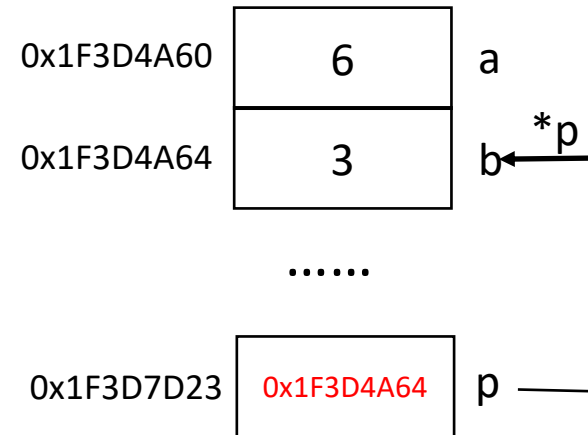
*p = *p + 1;
printf("%d %d\n", a, *p);

➔ p = &b;
printf("%d %d\n", a, *p);

p = p + 1;
printf("%d %d\n", a, *p);
```

Output:

5	5
6	6
6	3
6	-82119334



Example

```
int a = 1, b = 3;
int *p = &a;

*p = 4;
a = a + 1;
printf("%d %d\n", a, *p);

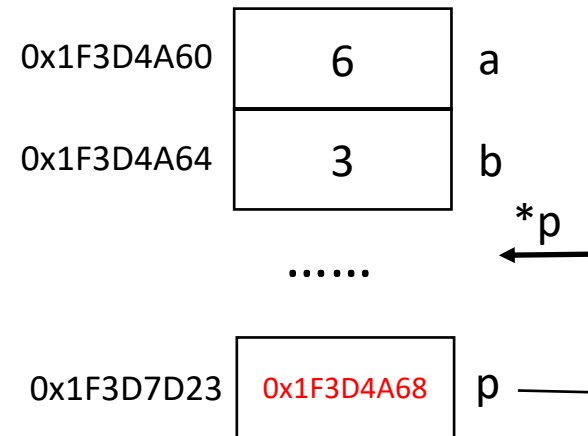
*p = *p + 1;
printf("%d %d\n", a, *p);

p = &b;
printf("%d %d\n", a, *p);

➔ p = p + 1;
printf("%d %d\n", a, *p);
```

Output:

5	5
6	6
6	3
6	-82119334



The pointer p
points to a wired place.
This would cause fatal error.

Notes of Pointer

- A pointer can be assigned to ZERO that means it is a NULL pointer, however DO NOT assign a non-zero number to a pointer

```
int *p;  
p = 0;    //initialized to be a NULL pointer  
p = 30 ;  //DO NOT use like this, though no compiling error
```

- When assign an address to a pointer, DO NOT use * operator before the pointer

```
int *p;  
int i;  
p = &i;    //CORRECT  
*p = &i ;  //ERROR
```

- Initialize a pointer before use it

```
int *p;    // No Initialization  
*p = 200;  // ERROR
```

```
int *p = 0; // a NULL pointer  
if ( p )  
    *p = 200;
```

Notes of Pointer

- The value of a pointer can be changed during a program that means the variables where it points to can be changed

```
int a = 4, b = 5;  
int *p = &a;           // points to a  
printf("%d\n", *p);    // output 4  
p = &b;                 // points to b  
printf("%d\n", *p);    // output 5
```

- A pointer can only point to a variable with the same type.

```
float *p;  
int i;  
p = &i; //ERROR
```

Example

- Exchange the values of two pointers

```
#include <stdio.h>

int main()
{
    int a = 3, b = 4;
    int *pa, *pb, *ptemp;

    pa = &a; pb = &b;
    printf("a = %d, b = %d\n", a, b);
    ➔ printf("*pa = %d, *pb = %d\n", *pa, *pb);

    ptemp = pa;
    pa = pb;
    pb = ptemp;
    printf("a = %d, b = %d\n", a, b);
    printf("*pa = %d, *pb = %d\n", *pa, *pb);
    return 0;
}
```



Output:

```
a = 3,    b = 4
*pa = 3,  *pb = 4
```

Example

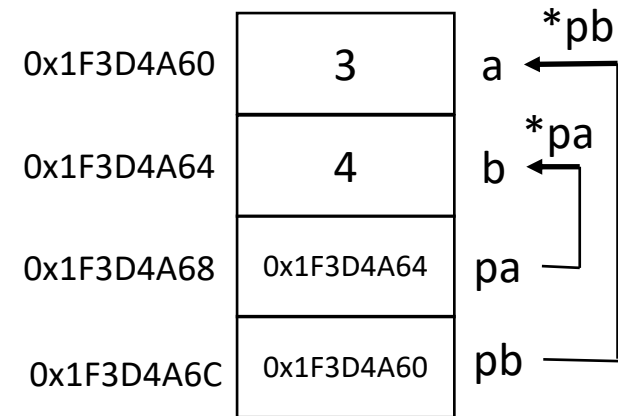
- Exchange the values of two pointers

```
#include <stdio.h>

int main()
{
    int a = 3, b = 4;
    int *pa, *pb, *ptemp;

    pa = &a; pb = &b;
    printf("a = %d, b = %d\n", a, b);
    printf("*pa = %d, *pb = %d\n", *pa, *pb);

    ptemp = pa;
    pa = pb;
    pb = ptemp;
    printf("a = %d, b = %d\n", a, b);
    printf("*pa = %d, *pb = %d\n", *pa, *pb);
    return 0;
}
```



Output:

```
a = 3,    b = 4
*pa = 3,  *pb = 4
a = 3,    b = 4
*pa = 4,  *pb = 3
```

Exercise

- Exchange the values of two variables by using pointers

```
#include <stdio.h>

int main()
{
    int a = 3, b = 4, temp;
    int *pa, *pb;

    pa = &a; pb = &b;
    printf("a = %d, b = %d\n", a, b);
    printf("*pa = %d, *pb = %d\n", *pa, *pb);

    temp = *pa;
    *pa = *pb;
    *pb = temp;
    printf("a = %d, b = %d\n", a, b);
    printf("*pa = %d, *pb = %d\n", *pa, *pb);
    return 0;
}
```

Pointer as Parameters for Functions

- By setting the address of a variable to be the actual parameter of a function, its value can be changed inside the function

```
#include <stdio.h>

void func (int *p)
{
    *p = 10;
}

int main()
{
    int a = 5;
    printf("a = %d\n", a); // output 5
    func( &a );
    printf("a = %d\n", a); // output 10
}
```

Are the values of x and y swapped?

```
#include <stdio.h>

int swap(int, int);
int main()
{
    int x = 3, y = 5;
    swap( x, y );
    printf("x=%d, y=%d\n",x,y);
    return 0;
}

void swap(int x, int y)
{
    int temp;
    temp = x;
    x = y;
    y = temp;
}
```

Output: $x = 3, y = 5$

Program Memory

stack area (栈区)		
0x25D1AB00	3	x
0x25D1AB04	5	y
	.	
	.	
	.	
0x2810AB70	3	x = x
0x2810AB74	5	y = y
0x2810AB78		temp
	.	
	.	
	.	

Are the values of x and y swapped?

```
#include <stdio.h>

int swap(int, int);
int main()
{
    int x = 3, y = 5;
    swap( x, y );
    printf("x=%d, y=%d\n",x,y);
    return 0;
}

void swap(int x, int y)
{
    int temp;
    temp = x;
    x = y;
    y = temp;
}
```



Output: x = 3, y = 5

Program Memory

	stack area (栈区)
0x25D1AB00	3
0x25D1AB04	5
	.
	.
	.
0x2810AB70	5
0x2810AB74	3
0x2810AB78	3
	.
	.
	.

The values of two variables can not be changed by a function, if they are passed to the function as parameters

x

y

temp

Using Pointers as Parameters of the Function

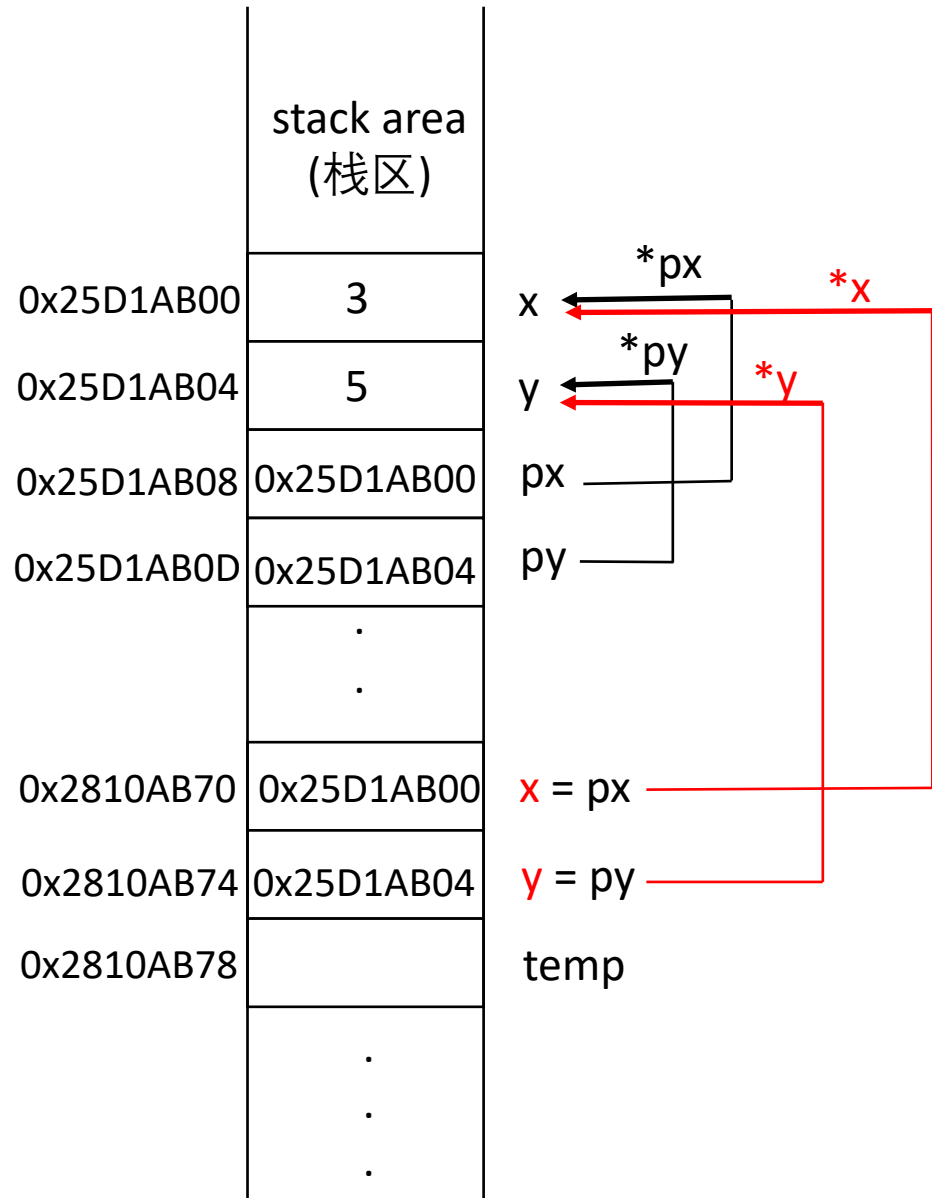
```
#include <stdio.h>

int swap(int *, int *);
int main()
{
    int x = 3, y = 5;
    int *px = &x;
    int *py = &y;
    swap( px, py );
    printf("x=%d, y=%d\n",x,y);
    return 0;
}

void swap(int *x, int *y)
{
    int temp;
    temp = *x;
    *x = *y;
    *y = temp;
}
```

Output: $x = 5, y = 3$

Program Memory




Using Pointers as Parameters of the Function

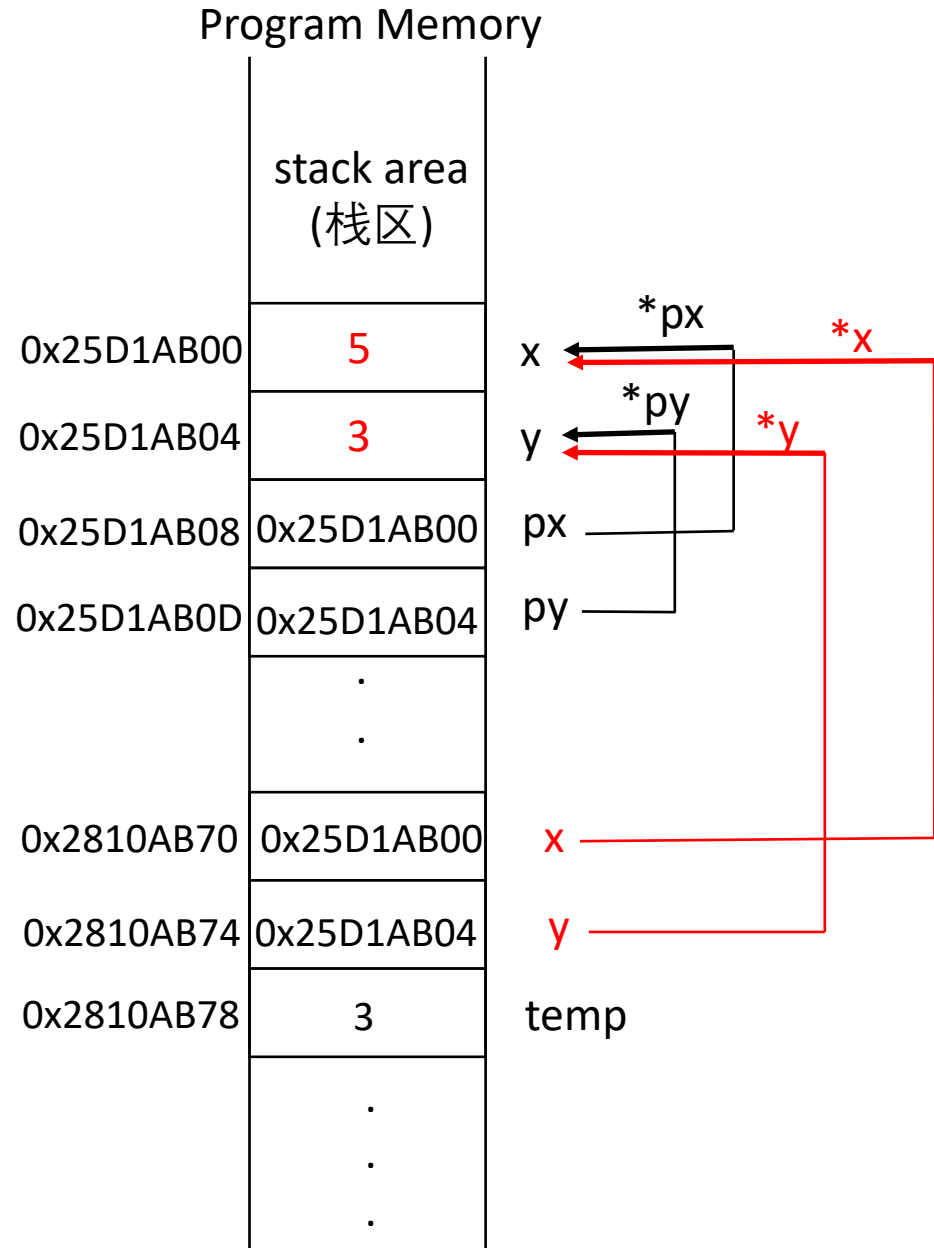
```
#include <stdio.h>

int swap(int *, int *);
int main()
{
    int x = 3, y = 5;
    int *px = &x;
    int *py = &y;
    swap( px, py );
    printf("x=%d, y=%d\n",x,y);
    return 0;
}

void swap(int *x, int *y)
{
    int temp;
    temp = *x;
    *x = *y;
    *y = temp;
}
```



Output: $x = 5, y = 3$



Does the values of Two Variable exchange?

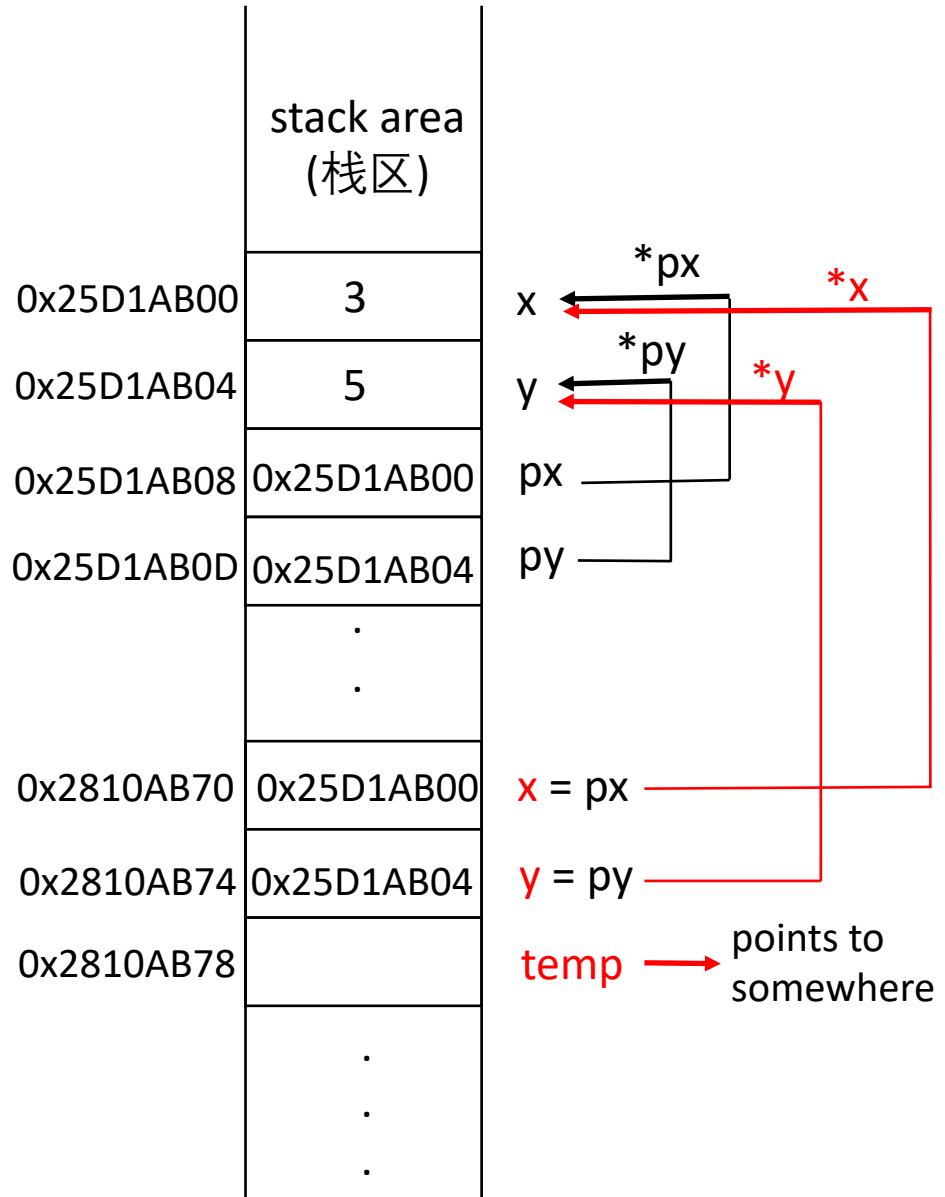
```
#include <stdio.h>

int swap(int *, int *);
int main()
{
    int x = 3, y = 5;
    int *px = &x;
    int *py = &y;
    swap( px, py );
    printf("x=%d, y=%d\n",x,y);
    return 0;
}

void swap(int *x, int *y)
{
    int *temp;
    temp = x;
    x = y;
    y = temp;
}
```

Output: $x = 3, y = 5$

Program Memory




Does the values of Two Variable exchange?

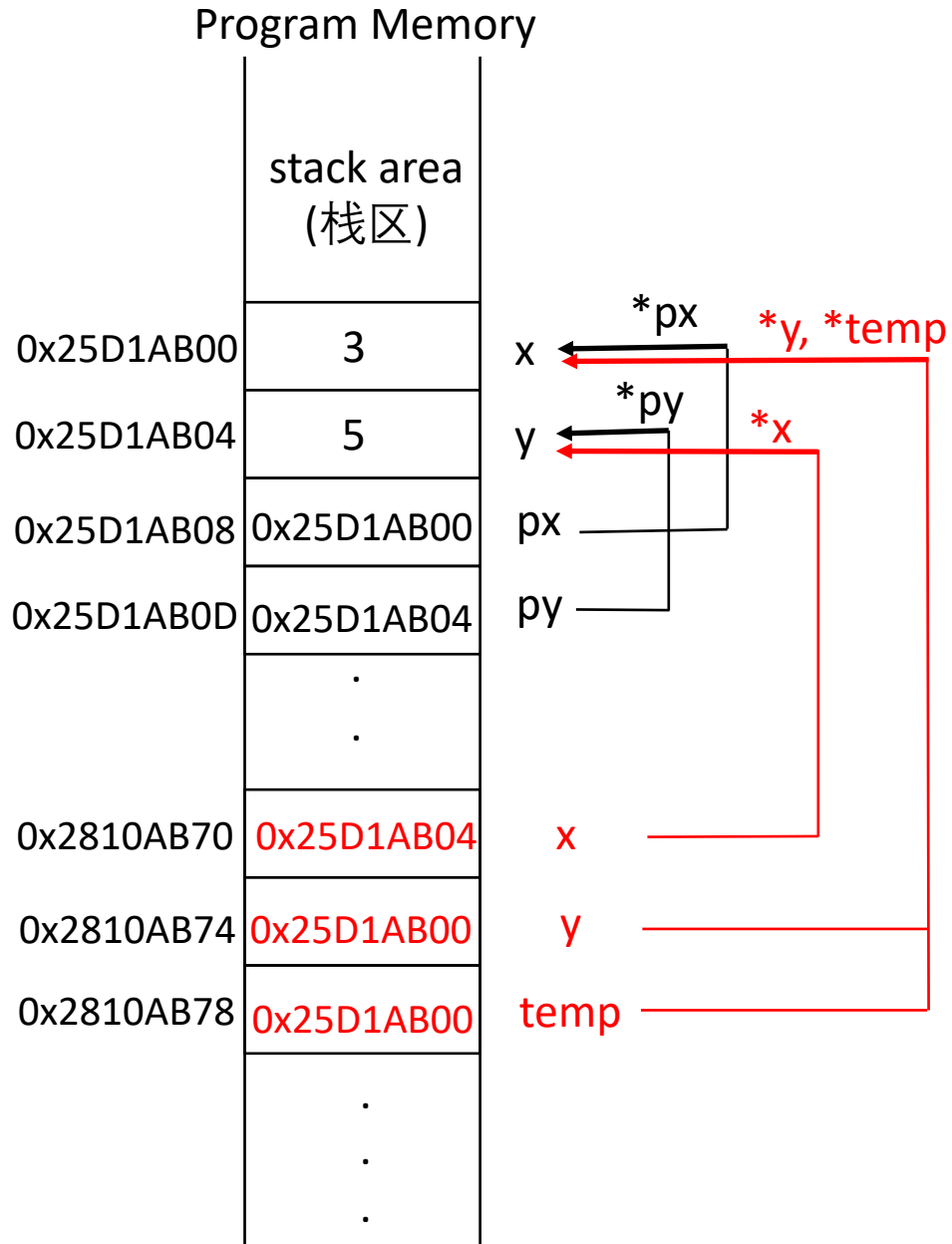
```
#include <stdio.h>

int swap(int *, int *);
int main()
{
    int x = 3, y = 5;
    int *px = &x;
    int *py = &y;
    swap( px, py );
    printf("x=%d, y=%d\n",x,y);
    return 0;
}

void swap(int *x, int *y)
{
    int *temp;
    temp = x;
    x = y;
    y = temp;
}
```



Output: $x = 3, y = 5$



Notes of Using Pointer as Parameters for Functions

- Values of several variables in the calling function (主调函数) can be changed by a called function(被调函数), if pointers are used as formal parameters of the called function(被调函数)
- If the formal parameters of a function are pointers, the actual parameters should be pointers or addresses of variables when the function is called
- The values of pointers or addresses of variables are passed from the calling function (主调函数) to the called function (被调函数)
- Inside the called function (被调函数), the values of variables in the calling function (主调函数) can be changed by using the * operator on formal parameters (pointers)
- The value-passing can never be changed, no matter whether the formal parameters of a called function are pointers or not

Exercise

- Calculate the area and perimeter (周长) of a circle by using a function whose prototype is given as follows.

```
void calAreaAndPerimeter(double radius,  
    double *pArea, double *pPerimeter);
```

```
#include <stdio.h>
#define PI 3.1415926

void calAreaAndPerimeter(double radius,
                        double *pArea, double *pPerimeter)
{
    *pArea = PI * radius * radius;
    *pPerimeter = 2.0 * PI * radius;
}

int main()
{
    double r;
    double area, perimeter;
    scanf("%lf", &r);

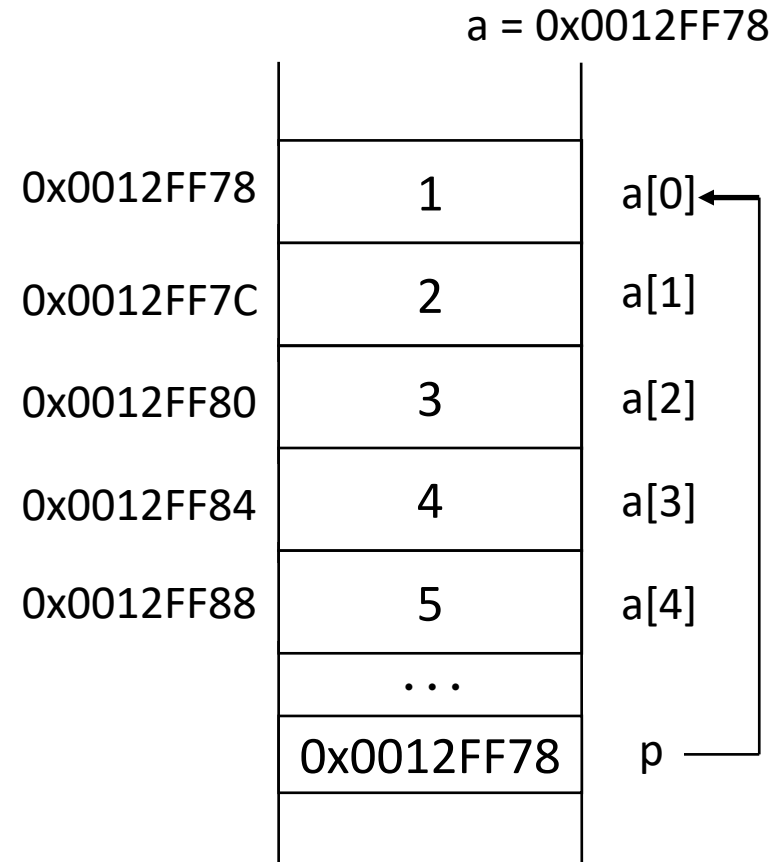
    calAreaAndPerimeter(r, &area, &perimeter);
    printf("radius = %f\n", r);
    printf("area = %f\n", area);
    printf("perimeter = %f\n", perimeter);

    return 0;
}
```

1D Array & Pointer

- The name of a 1D array is a constant that is the memory address of the 1st element of this array

```
int a[5] = {1, 2, 3, 4, 5};  
int *p;  
  
printf("%x\n", a); // 0x0012FF78  
  
p = &a[0];  
printf("%x\n", p); // 0x0012FF78  
  
p = a;  
printf("%x\n", p); // 0x0012FF78
```



Access Array Elements by using a Pointer

- If a pointer points to the 1st element of an array, all elements can be accessed by the following two methods

- Index-based Method

```
int a[5] = {1, 2, 3, 4, 5};
int *p = a, i;
for ( i = 0 ; i < 5 ; i++ )
{
    printf(“%d %d\n”, a[i], p[i]);
}
```

- Pointer-based Method

```
int a[5] = {1, 2, 3, 4, 5};
int *p = a, i;
for ( i = 0 ; i < 5 ; i++ )
{
    printf(“%d %d\n”, *(a+i), *(p+i));
}
```

```
int a[5] = {1, 2, 3, 4, 5};
int *p = a;
```

1	a[0], p[0], *a, *p
2	a[1], p[1], *(a+1), *(p+1)
3	a[2], p[2], *(a+2), *(p+2)
4	a[3], p[3], *(a+3), *(p+3)
5	a[4], p[4], *(a+4), *(p+4)
.	
.	
.	

Examples

```
int a[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};  
int *p = &a[0];  
  
printf("%d\n", *p); // 1  
  
printf("%d\n", p[4]); // 5  
  
p++;  
printf("%d\n", *p); // 2  
  
p += 4;  
printf("%d\n", *p); // 6  
  
p--;  
printf("%d\n", *p); //5  
  
p -= 5;  
printf("%d\n", *p); //ERROR
```

```

#include <stdio.h>

int main()
{
    int a[5], i;
    int *p = a;

    for (i = 0 ; i < 5 ; i++ )
    {
        *p = i;
        p++;
    }

    p = a;
    for ( i = 0 ; i < 5 ; i++ )
    {
        printf("%d ", *p)
        p++;
    }
    printf("\n");

    return 0;
}

```

```

#include <stdio.h>

int main()
{
    int a[5], i = 0;
    int *p = a;

    for ( ; p < a + 5; i++, p++ )
        *p = i;

    for ( p = a ; p < a + 5; p++ )
        printf("%d ", *p)
    printf("\n");

    return 0;
}

```

```

int a[5];
a++; //ERROR
a--; //ERROR

```

Because a is a constant,
a++ or a-- is not allowed

Array as Function Parameters

- Array Names as the Function Parameters (Passing Address)
 - The array name is constant address of that array
 - When the array name is used as actual parameter, only the address is passed into the function
 - The memory on the stack area is allocated to store the address, not the whole array. By using this address, the value of each array element can be changed

```

#include <stdio.h>

void fun(int b[])
{
    int i;
    for ( i = 0 ; i < 3 ; i++ )
        printf("%d ", ++b[i]);
    printf("\n");
}

int main ( )
{
    int a[3] = {1, 2, 3}, i;
    fun( a );

    for( i = 0 ; i < 3 ; i++ )
        printf("%d ", a[i]);

    printf("\n");

    return 1;
}

```

Program Memory

	stack area (栈区)	
		a = 0xAB00
0xAB00	1	a[0]
0xAB04	2	a[1]
0xAB08	3	a[2]
	.	
	.	
	.	
0xAB70		
	.	
	.	
	.	

```

#include <stdio.h>

void fun(int b[])
{
    int i;
    for ( i = 0 ; i < 3 ; i++ )
        printf("%d ", ++b[i]);
    printf("\n");
}

int main ( )
{
    int a[3] = {1, 2, 3}, i;

    fun( a );

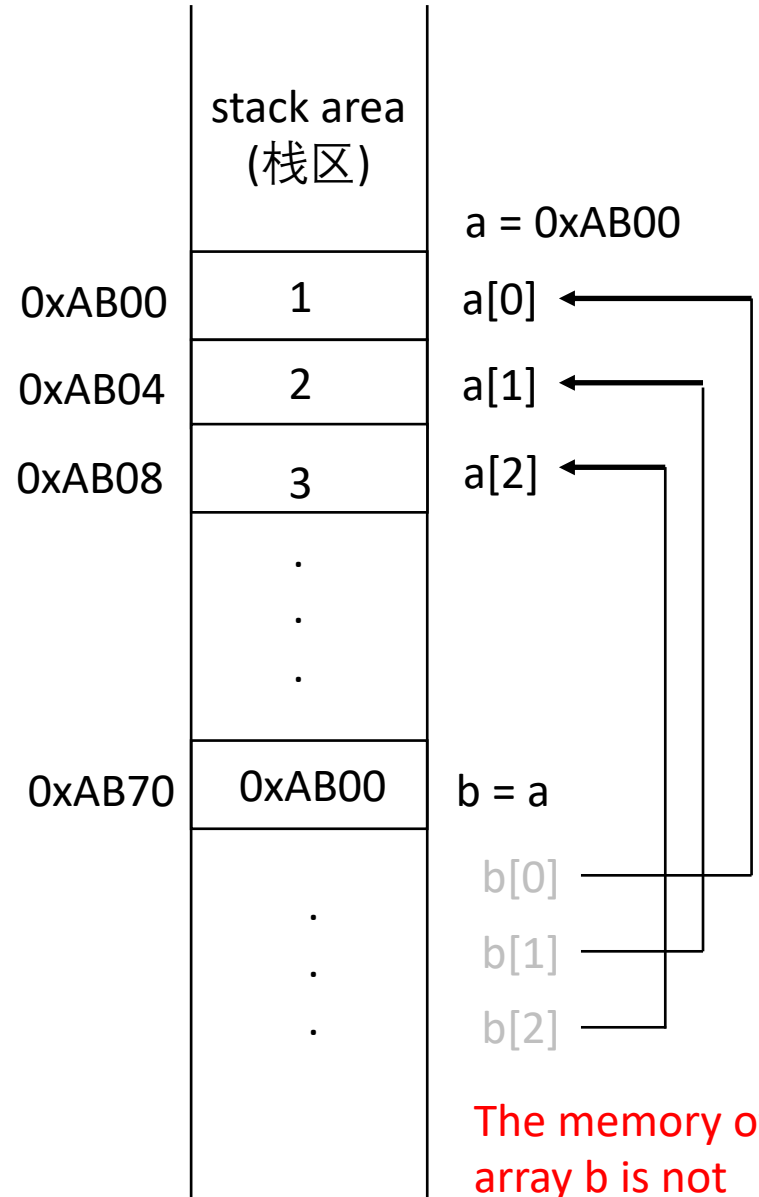
    for( i = 0 ; i < 3 ; i++ )
        printf("%d ", a[i]);

    printf("\n");

    return 1;
}

```

Program Memory



The memory of
array b is not
allocated on stack

```

#include <stdio.h>

void fun(int b[])
{
    int i;
    for ( i = 0 ; i < 3 ; i++ )
        printf("%d ", ++b[i]);
    printf("\n");
}

int main ( )
{
    int a[3] = {1, 2, 3}, i;

    fun( a );

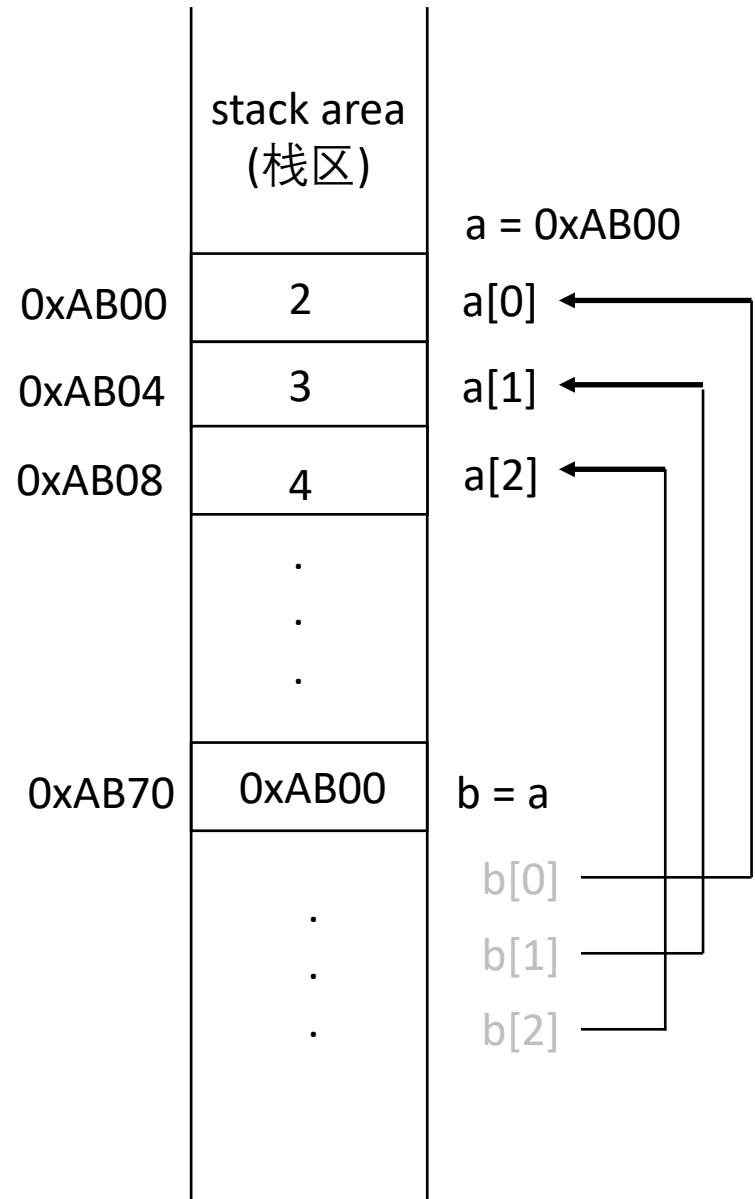
    for( i = 0 ; i < 3 ; i++ )
        printf("%d ", a[i]);

    printf("\n");

    return 1;
}

```

Program Memory



```

#include <stdio.h>

void fun(int b[])
{
    int i;
    for ( i = 0 ; i < 3 ; i++ )
        printf("%d ", ++b[i]);
    printf("\n");
}

int main ( )
{
    int a[3] = {1, 2, 3}, i;

    fun( a );

    for( i = 0 ; i < 3 ; i++ )
        printf("%d ", a[i]);

    printf("\n");

    return 1;
}

```

Program Memory

Program Memory		
	stack area (栈区)	
		a = 0xAB00
0xAB00	2	a[0]
0xAB04	3	a[1]
0xAB08	4	a[2]
	.	
	.	
	.	
0xAB70		
	.	
	.	
	.	


```
#include <stdio.h>

void fun(int b[])
{
    int i;
    for ( i = 0 ; i < 3 ; i++ )
        printf("%d ", ++b[i]);
    printf("\n");
}

int main ( )
{
    int a[3] = {1, 2, 3}, i;
    fun( a );
    for( i = 0 ; i < 3 ; i++ )
        printf("%d ", a[i]);
    printf("\n");
    return 1;
}
```

Output:

2 3 4

2 3 4

Four Methods to Change Array Elements by a Function

- Both actual and formal parameters are array names

```
void func(int b[]){...};  
int main(){  
    int a[3];  
    func(a);  
    ...  
}
```

- Actual parameter is a pointer, while formal parameter is an array name

```
void func(int b[]){...};  
  
int main(){  
    int a[3]; int *pa = a;  
    func(pa);  
    ...  
}
```

- Actual parameter is array name, while formal parameter is a pointer
- Both actual and formal parameters are pointers

```
void func(int *pb){...};  
int main(){  
    int a[3];  
    func(a);  
    ...  
}
```

```
void func(int *pb){...};  
int main(){  
    int a[3]; int *pa = a;  
    func(pa);  
    ...  
}
```

Exercise

- Write a function and find all elements that can be divided by 3 in an array. The array and the prototype of the function are given as follows.

```
int a[10] = {5, 15, 25, 36, 47, 58, 69, 78, 56, 90};  
  
// pSource - the original array  
// n - the number of elements in the original array  
// pTar - the target array where results are saved  
// returned value - the number of elements that are found  
int select(int *pSrc, int n, int *pTar);
```

```
#include <stdio.h>

int select(int *pSrc, int n, int *pTar)
{
    int i, num = 0;
    for ( i = 0 ; i < n ; i++ )
        if ( pSrc[i] % 3 == 0 )
            pTar[num++] = pSrc[i];

    return num;
}

int main()
{
    int a[10] = {5, 15, 25, 36, 47, 58, 69, 78, 56, 90};
    int b[10];
    int i, num;
    num = select(a, 10, b);
    for ( i = 0 ; i < num ; i++ )
        printf("%d\n", b[i]);

    return 0;
}
```

Function Returns a Pointer

```
type * function_name (parameter list)
{
    Function Body;
}
```

Examples:

```
int *func1(int x, int y)
{
    .....
}

float *func2(float *p, int n)
{
    .....
}
```

Example

- Select the maximal element from an array and return its address

```
#include <stdio.h>

int *max(int *pArray, int n)
{
    int i, *p;
    p = &pArray[0];
    for ( i = 1 ; i < n ; i++ )
        if ( *p < pArray[i] )
            p = &pArray[i];
    return p;
}

int main()
{
    int a[6] = {5, 14, 18, 98, 37, 23};
    int *pMax = max(a, 6);
    printf("Max Element = %d\n", *pMax);
    return 0;
}
```

- DO NOT make a function return a pointer that points to released memory
 - DO NOT return the address of a local variable inside the function

```
#include <stdio.h>

char *lost()
{
    char c1 = 'A';
    return &c1;
}

int main()
{
    char *p;
    p = lost();
    printf("%c\n", *p);
    return 0;
}
```

The memory of c1 is allocated when the function begins, and it is released when the function ends

ERROR!

The pointer p points to the released memory, this memory is not accessible.

- String Constant (字符串常量)
 - A sequence of characters quoted by **double quote marks ("")**
 - Examples: "C Language", "Hello World!", "C"
 - Contains 0 or several characters
 - Null String : ""
 - No string variable in C language
 - No fixed size
- Character Constant (字符常量)
 - A **single** character quoted by **single quote masks (")**
 - Example : 'a', 'C'
 - Contains only a single character
 - Null character : '\0'
 - It is able to assign a character constant to a character variable
 - 1 Byte

Memory of a String

- A String has a sequential memory
- In the end of a string, there is a null character ('\0') that indicates the tail of the string
- The actual size of a string is N+1, where N is the number of non-null characters.

The memory of the string "C Language"

C		L	a	n	g	u	a	g	e	\0
---	--	---	---	---	---	---	---	---	---	----

The memory of the string "C"

C	\0
---	----

The memory of the character 'C'

C

String & Character Array

- A string can be assigned to a character array

```
char a[] = {"C Language"};  
OR  
char a[] = "C Language";
```

- The sizes of the following two arrays are different

```
// size of the array a is 5  
char a[] = {'H','e','l','l','o'};
```

```
// size of the array b is 6, there is  
// a null character '\0' at the tail of b  
char b[] = "Hello";
```

Input and Output of a String

- Output a String

- Using formatted character “%s” and printf() function
 - “%s” – Print as a string, all characters are printed until the non-character ‘\0’

```
char a[] = "Hello World!";  
printf("%s", a);
```

- Using formatted character “%c” and printf() function
 - “%c” – Print as a character array

```
char a[] = "Hello World!";  
int i;  
for ( i = 0 ; i < 12 ; i++ )  
    printf("%c", a[i]);
```

Input and Output of a String

- Input a String
 - Using formatted character “%s” and scanf() function
 - During the inputting of a string, no space character exist in the middle of the string

```
char a[11];  
//The last character should be spared out for  
//saving '\0', therefore inputted characters  
//should be less than 10  
scanf("%s", a);
```

If “C Language” is inputted, the obtained string only includes two characters that are ‘C’ and ‘\0’

- Using formatted character “%c” and scanf() function

```
char a[11];  
int i;  
//We can input 11 characters  
for ( i = 0 ; i < 11 ; i++ )  
    scanf("%c", &a[i]);
```

String & Pointer

- Use a pointer to access elements of a string

```
char c[] = "C Language";  
char *p = c;
```

C		L	a	n	g	u	a	g	e	\0
---	--	---	---	---	---	---	---	---	---	----

index-based
Method

p[0] p[1] p[2] p[3] p[4] p[5] p[6] p[7] p[8] p[9] p[10]

pointer-based
Method

*p *(p+1) *(p+2) *(p+3) *(p+4) *(p+5) *(p+6) *(p+7) *(p+8) *(p+9) *(p+10)

```
#include <stdio.h>
```

```
int main()
```

```
{
```

```
    char c[] = "C Language";
```

```
    int i;
```

```
    char *p = c;
```

```
    for ( i = 0 ; i < 10 ; i++ )
```

```
        printf("%c", p[i]);
```

```
    printf("\n");
```

```
    for ( i = 0 ; i < 10 ; i++, p++ )
```

```
        printf("%c", *p);
```

```
    printf("\n");
```

```
    p = c;
```

```
    printf("%s\n", p); // output: C Language
```

```
    p += 4;
```

```
    printf("%s\n", p); // output: nguage
```

```
    return 0;
```

```
}
```

print a string that starts
from the pointer p to '\0'

Pointer & Array for a String Constant

- If a string constant is assigned to a pointer, we can use the pointer to obtain the values of each character. However, we can not change any character.

```
char *p = "C Language";  
char c1 = p[3]; //OK  
p[3] = 'A';      //ERROR  
p = "Hello";     //OK
```

- If a string constant is assigned to an array, we can use the array name to obtain and change each character. However, we can not make another string constant to be assigned to this array

```
char c[] = "C Language";  
char c1 = c[3]; //OK  
c[3] = 'A';     //OK  
c = "Hello";    //ERROR
```

Functions of Strings

- Output a String

```
int puts(const char *pc);
```

- Input a String

```
char *gets(char *pc);
```

```
#include <stdio.h>

int main()
{
    char c[256]; // array to save a sting
    gets(c);    // input the string
    puts(c);    // output the string
    return 0;
}
```


Functions of Strings

#include <string.h> is required before the following functions are called

- Calculate the length of a String
 - Return the length of the string

```
int strlen(const char *pc);
```

- Concatenate two strings to be a united string
 - Concatenated string is saved in pc1

```
char *strcat(char *pc1, const char *pc2);
```

- Copy a string
 - Copied string is saved in pc1

```
char *strcpy(char *pc1, const char *pc2);
```

- Compare two strings
 - If two strings are identical, return 0

```
int *strcmp(const char *pc1, const char *pc2);
```

- Get Substring(子字符串)

- Returns a pointer to the first occurrence of *str2* in *str1*

```
char *strstr( char *str1, const char *str2);
```