Tutorial 3
Pointer and Array

Jiawei Li (Michael)

Office hours: Monday 1:00-3:00pm
Office: PMB426
Email: jiawei.li@Nottingham.edu.cn

Pointer

- Define a pointer

```
int *p;
```

- Initialize a pointer

```
p= &a;
```

- Use a pointer

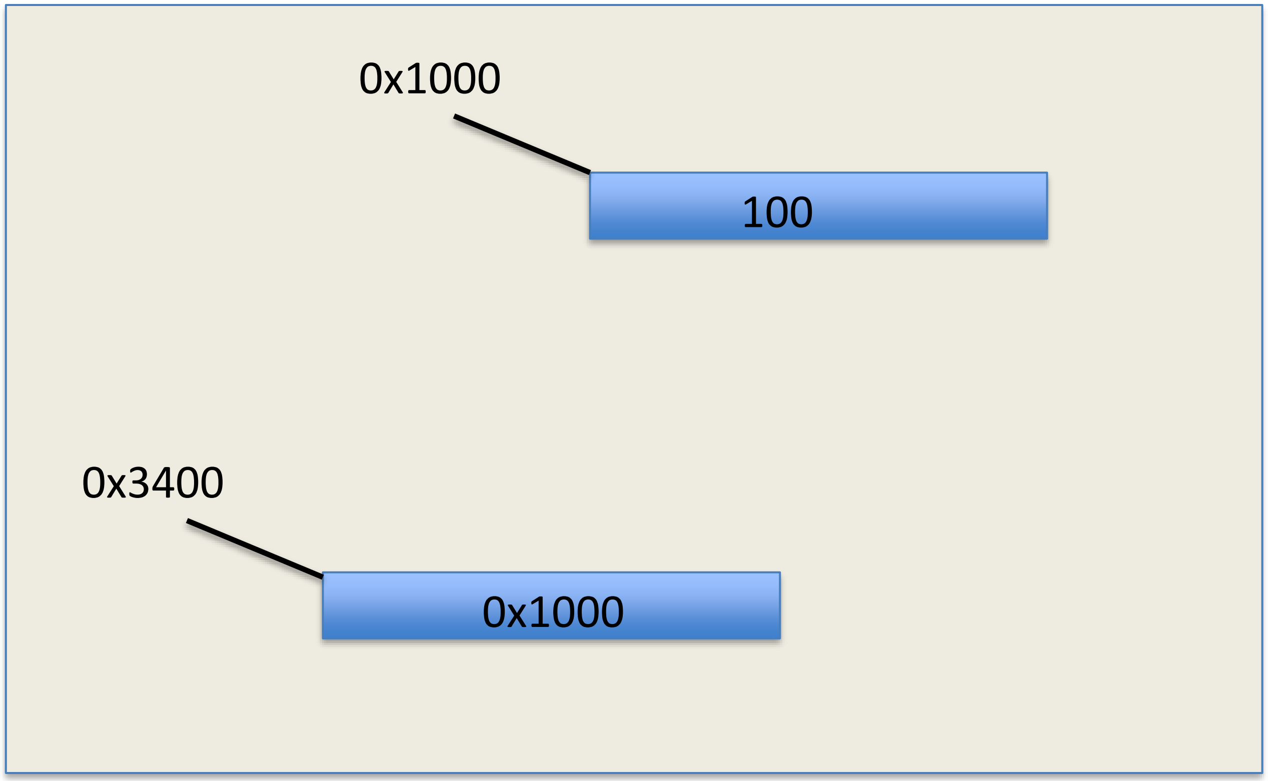
```
*p = 5;   // *p == a
p = &b;  // p -> b
```

Pointer

- A pointer needs to be initialized before using it.
- An initialized pointer is a valid address (not a random value)
- Using an uninitialized pointer may lead to ‘segmentation fault’.

Define a variable

- int a;

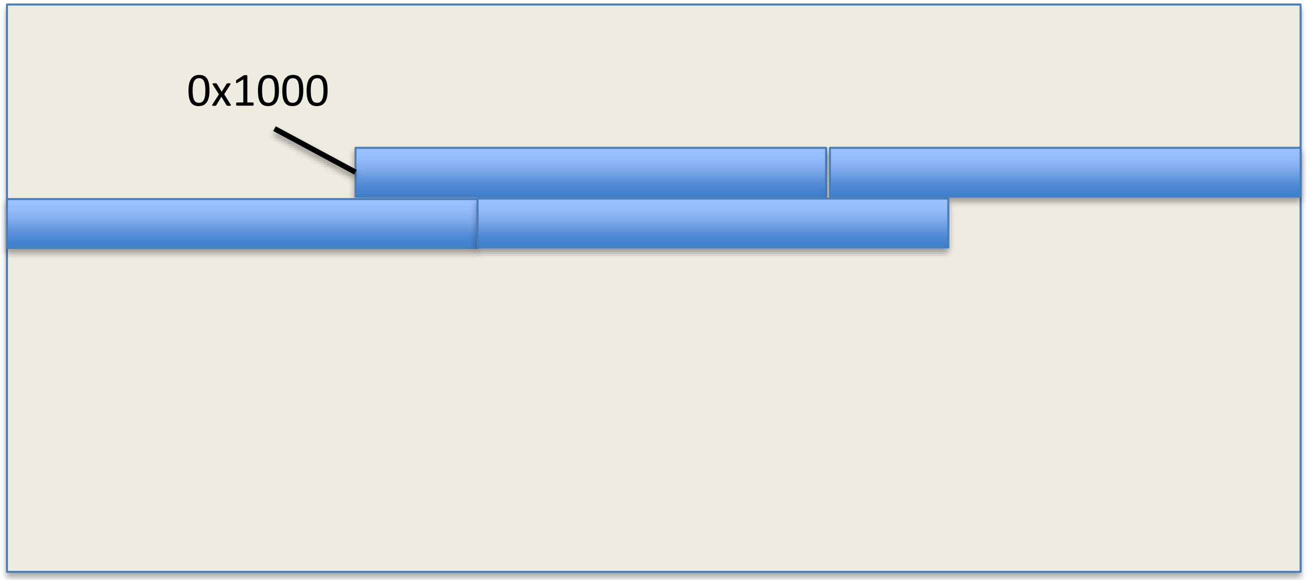


- int *p;
- p = &a;
- *p = 100;

Array

An array is equivalent to a constant pointer. A constant pointer contains a constant address that cannot be changed.

```
int a[4];
// a = &b;
// a==0x1000
// a[0] is an int
// &a[1] ==
// 0x1004
```



Frequent mistakes

- Using uninitialized pointers

```
#include<stdio.h>
int main(int argc, char* argv[ ])
{
        int a;
        a = 100;
        printf(“%d”, a);


        int *ptr = NULL;
        *ptr=100;
        printf(“%d, %p”, *ptr, ptr);
}
```

Frequent mistakes

- Pointer to uninitialized variable

```
#include<stdio.h>
int main(int argc, char* argv[ ])
{
        int *ptr;
        int a;
        ptr = &a;

        printf(“%d, %d, %p\n”, a, *ptr, ptr);

}
```

Exercise 1

```
#include<stdio.h>
int main(int argc, char* argv[ ])
{
        int a = 1;
        int b = 2;

        int *p1 = &a;
        int *p2 = &b;
        printf(“%d, %d”, *p1, *p2);

        p1 = p2;
        *p1 = 3;
        printf(“%d, %d”, a, b);

        printf(“%d, %d”, *p1, *p2);
        return 0;
}
```

Exercise 2

```
#include<stdio.h>
int main(int argc, char* argv[ ])
{
        int a[] = {1, 2, 3};
        printf(“%d, %d, %d\n”, a[0], a[1], a[2]);
        printf(“%p, %p, %p\n”, &a[0], &a[1], &a[2]);

        int b[] = {1, 2, 3};
        printf(“%p, %p\n”, a, b);

        char* s1 = “Paul”;
        char* s2 = “Paul”;
        printf(“%p, %p\n”, s1, s2);


}
```

Exercise for you

Write a program that allows the user to enter several numbers into an array and print out positive numbers. The program should start by asking the user to enter how many numbers they are going to input, using the prompt "Enter number of values: ". This input must be an integer greater than zero. If it is not greater than zero then the program should print the error message "Invalid array length!" and exit. If the number is valid, the program should create an int array of that length.

Next, the program should prompt the user to enter a number for each position in the array, starting from position 0, using the prompt "Enter position XX value: " (where XX is the zero-based index position in the array that the number will be stored in). The value must be an even int value. If the number is not valid, the program should repeatedly prompt the user to enter the number for the same position again, using the message above, until it is valid.

Once all the numbers have been entered, the program should print all the positive values (excluding zero) in the array, each in a line, and exit. User input should be on the same line as the prompt. Output messages should be printed on a line by themselves. When the program exits the command prompt should not be on the same line as the output message. You can assume the user will only enter a single whole number at each prompt (ie, no text, no floating-point numbers, no multiple numbers, no empty input, etc).