Xi'an Jiaotong-Liverpool University
西交利物浦大学

CPT104 Operating System Concepts

Lab 5 (1)

Pointer (2) and Structure

Advanced usage of Pointer

Pointer and Array

```
#include <stdio.h>
int main( )
{
    //! showMemory(start=65520)
   int *p, i;
   int val[5] = { 11, 22, 33, 44, 55};
   p = val;
   for ( i = 0; i<5; i++ )
   {
      printf("val[%d]: value is %d\
 and address is %p\n", i, *(p+i), (p+i));
   }
   return 0;
}
```

- There is a strong relationship between arrays and pointers. Any operation that can be achieved by array subscripting can also be done with pointers.

the starting address of array *val* is assigned to point *p*, equivalent to *p = &val[0];*

Incrementing the pointer so that it points to next element on every increment

p+i increases the address by i **units**, not i **digits**, i.e. **i*sizeof(int)**

Address arithmetic ÓMÚ

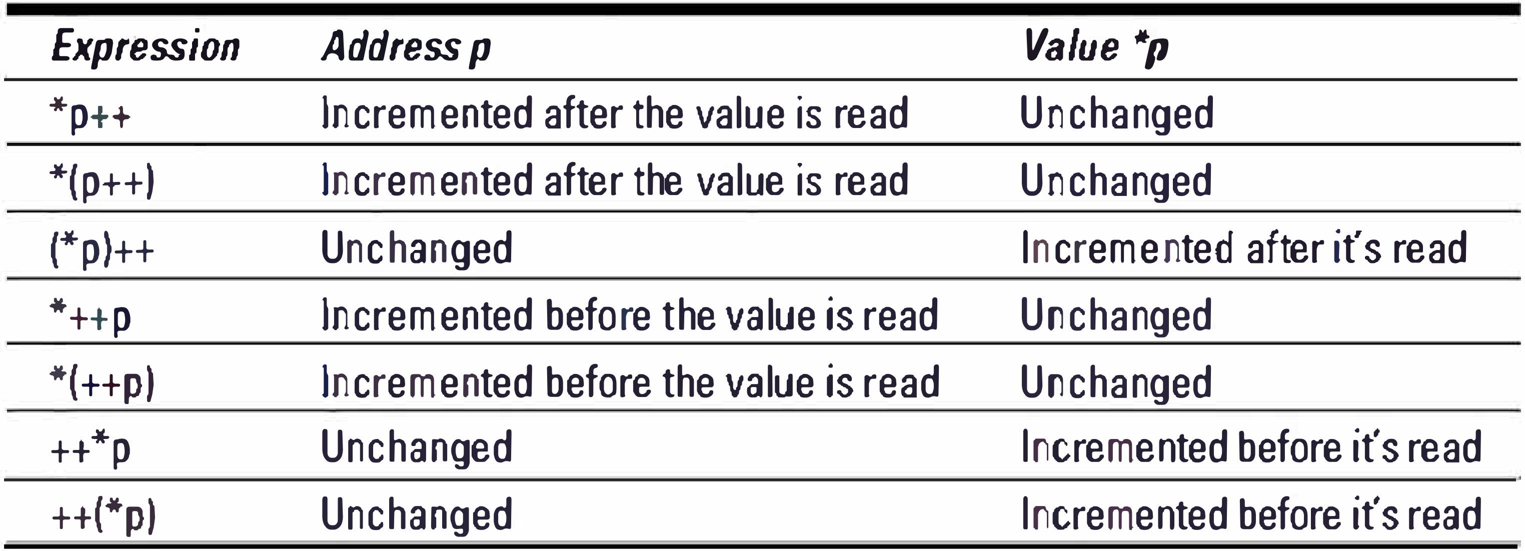
```
#include <stdio.h>
int main( )
{
  //! showMemory(start=65520)
   int *p, i;
   int val[5] = { 11, 22, 33, 44, 55};
   p = &val[0];
   for ( i = 0; i<5; i++ )
   {
       printf("val[%d]: value \
            is %d\n", i, *p++);
   }
   return 0;
}
```

- you can perform arithmetic operations on a pointer just as you can on a numeric value. There are four arithmetic operators that can be used on pointers: ++, --, +, and -

**p++ obtains the content of address p first and then increment p by one.*

Address arithmetic (2)

Pointers In and Out of Parentheses [1]



Expression	*Address p*	*Value *p*
*p++	Incremented after the value is read	Unchanged
*(p++)	Incremented after the value is read	Unchanged
(*p)++	Unchanged	Incremented after it's read
*++p	Incremented before the value is read	Unchanged
*(++p)	Incremented before the value is read	Unchanged
++*p	Unchanged	Incremented before it's read
++(*p)	Unchanged	Incremented before it's read

1. page 273, Beginning Programming wi+h C for Dummies. John Wiley & Sons; 2013 Oc+ 28.

Returning a pointer from a function

```c
#include <stdio.h>
int* larger(int*, int*);
int main()
{
     int a = 15;
     int b = 92;
     int *p;
     p = larger(&a, &b);
     printf("%d is larger",*p);
     return 0;
}
int* larger(int *x, int *y)
{
    if(*x > *y)
        return x;
    else
        return y;
}
```

- A pointer function return a memory location (address) as a value.

Declare pointer function larger which returns the address of larger variable

Call pointer function by returning an address.

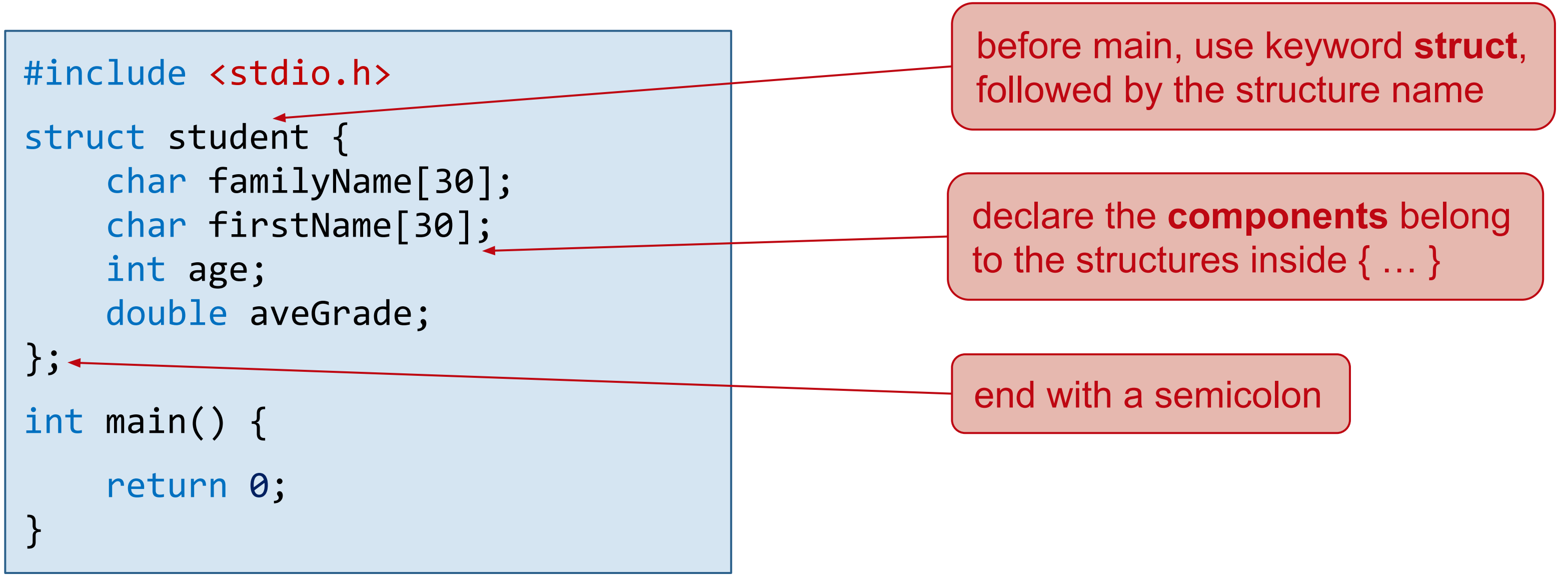
Return address of variable defined in main function.

- **Do not return address of local variable.**
- Alternative way is to define and refer to ***static variable*** in a sub-function.

Structure

Create Our Own Data Types

- Now, we want to learn to create *our own data types* using structures
  - Structures allow us to store multiple information in just ***one*** variable, for example: a **student** with family and first name, age, and average grades



```c
#include <stdio.h>

struct student {
    char familyName[30];
    char firstName[30];
    int age;
    double aveGrade;
};

int main() {

    return 0;
}
```

before main, use keyword **struct**, followed by the structure name

declare the **components** belong to the structures inside { … }

end with a semicolon

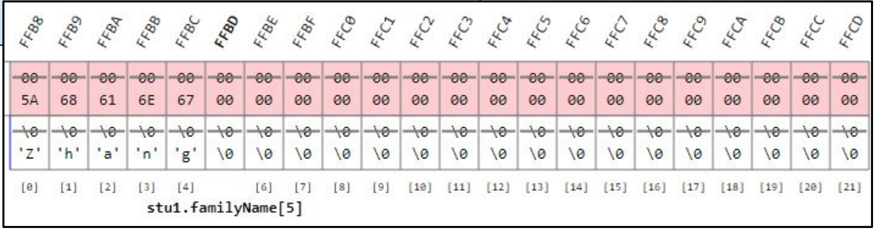
Instantiating Structures

- Create a new instance of struct student:

```c
#include <stdio.h>

struct student {
    char familyName[30];
    char firstName[30];
    int age;
    double aveGrade;
};

int main() {
    //! showMemory(start=65469)
    struct student stu1 = {"Zhang", "Wei", 24, 3.5};

    return 0;
}
```

declare with **struct** followed by the **name** of the structures

initialize with values in { … } in the same order



FFB8	FFB9	FFBA	FFBB	FFBC	**FFBD**	FFBE	FFBF	FFC0	FFC1	FFC2	FFC3	FFC4	FFC5	FFC6	FFC7	FFC8	FFC9	FFCA	FFCB	FFCC	FFCD
~~00~~ 5A	~~00~~ 68	~~00~~ 61	~~00~~ 6E	~~00~~ 67	~~00~~ 00	~~00~~ 00	~~00~~ 00	~~00~~ 00	~~00~~ 00	~~00~~ 00	~~00~~ 00	~~00~~ 00	~~00~~ 00	~~00~~ 00	~~00~~ 00	~~00~~ 00	~~00~~ 00	~~00~~ 00	~~00~~ 00	~~00~~ 00	~~00~~ 00
~~\0~~ 'Z'	~~\0~~ 'h'	~~\0~~ 'a'	~~\0~~ 'n'	~~\0~~ 'g'	~~\0~~ \0	~~\0~~ \0	~~\0~~ \0	~~\0~~ \0	~~\0~~ \0	~~\0~~ \0	~~\0~~ \0	~~\0~~ \0	~~\0~~ \0	~~\0~~ \0	~~\0~~ \0	~~\0~~ \0	~~\0~~ \0	~~\0~~ \0	~~\0~~ \0	~~\0~~ \0	~~\0~~ \0
[0]	[1]	[2]	[3]	[4]		[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]	[15]	[16]	[17]	[18]	[19]	[20]	[21]

stu1.familyName[5]

Accessing Structures

- Print and access the instance of struct student:

```c
#include <stdio.h>

struct student {
    char familyName[30];
    char firstName[30];
    int age;
    double aveGrade;
};

int main() {
    //! showMemory(start=65469)
    struct student stu1 = {"Zhang", "Wei", 24, 3.5};
    printf("Name: %s %s\n", stu1.familyName, stu1.firstName);
    printf("Age: %d\n", stu1.age);
    printf("Average grade: %.2lf\n", stu1.aveGrade);

    return 0;
}
```

use the dot operator

Modifying Structures

- Assign values to the components of an instance of struct student from user input:

```c
#include <stdio.h>

struct student {
    char familyName[30];
    char firstName[30];
    int age;
    double aveGrade;
};

int main() {
    //! showMemory(start=65469)
    struct student stu1;
    scanf("%s", stu1.familyName);
    scanf("%s", stu1.firstName);
    scanf("%d", &stu1.age);
    scanf("%lf", &stu1.aveGrade);

    return 0;
}
```

declare an instance

use the dot operator

need & for reading int

Pass Structures to Functions

- Create a function taking a student structure and print it

```c
#include <stdio.h>
struct student {
    char familyName[30];
    char firstName[30];
    int age;
    double aveGrade;
};
void printStudent(struct student);

int main() {
    //! showMemory(start=65469)
    struct student stu1 = {"Zhang", "Wei", 24, 3.5};
    printStudent(stu1);

    return 0;
}
void printStudent(struct student stu) {
    printf("Name: %s %s\n", stu.familyName, stu.firstName);
    printf("Age: %d\n", stu.age);
    printf("Average grade: %.2lf\n", stu.aveGrade);
}
```

the function prototype
after the structure definition,
before the main function

calling the function

passing a structure

check in memory visualization,
we are ***passing by value***
(the values are copied)

Thank you for your attention !

- In this lab, you have learned:
  - advanced usage of pointer
    - Array and pointer
    - Address Arithmetic
    - Returning a pointer
  - Structure
    - Definition
    - Pass Structures to Functions
    - Arrays of Structures
- **For more information:**

✓ refer to book chapter 5, 5.3-5.6, chapter 6, 6.1-6.4