EASTERN WASHINGTON UNIVERSITY
start something big

Pointer 1

Computer Science Department
Eastern Washington University
Yun Tian (Tony) Ph.D.

Recall

- C Arrays
- Two types
  - 'static' Array
  - Dynamic Array
  - Their differences
- How to use 'static' array in C program

Today

- Concept of address
- Concept of pointers
- How to define/declare pointers
- How to use pointers?

Motivation

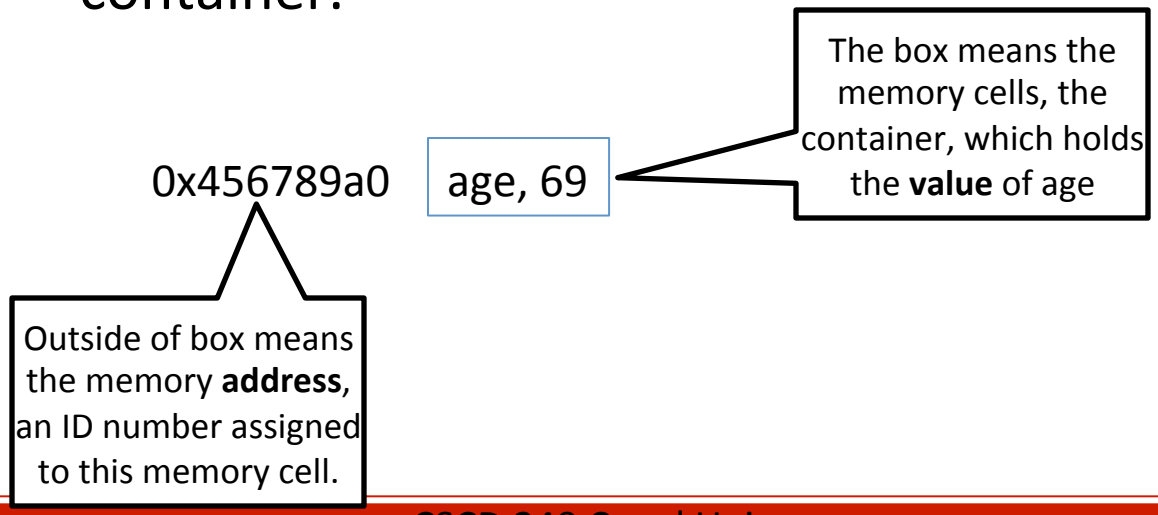
- Some C programming tasks are performed more easily with pointers.
- Some tasks cannot be performed without using pointers.
  - Dynamic memory allocation.
  - Linked Structures like linked list and Tree
- So it becomes necessary to learn pointers.

Concept of Address

- Every variable needs a memory location(as container),
  - We use the variable name to refer to the value stored in that container.
- Every memory location has its address defined,
  - We can obtain the address by using ampersand (&) operator.
  - int age = 69;
    - Age is a variable, we use the variable name to refer to its value.

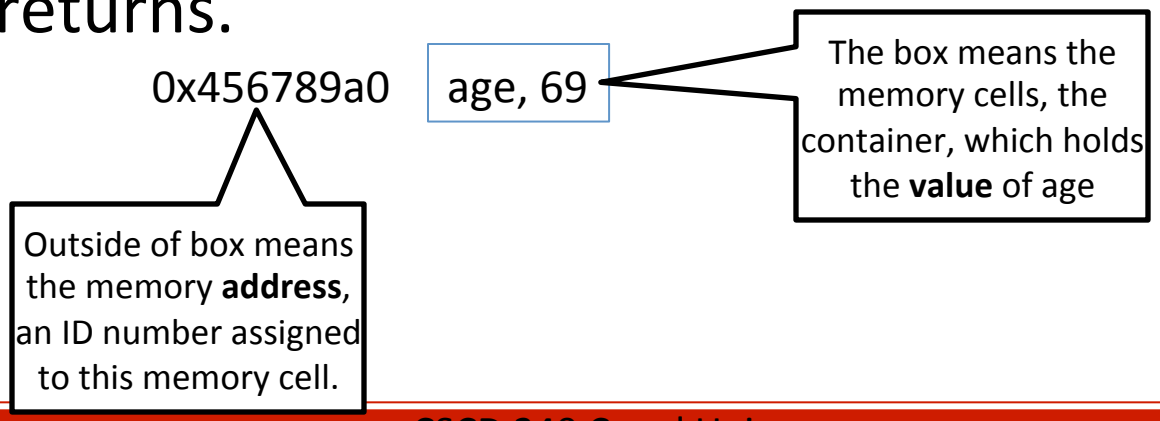
Concept of Address

- int age = 69;
  - age is a variable, we use the variable name to refer to its value,
    - That is, the variable name also refer to the value in container.



Memory Allocation

- int age = 69; in main() or in another function
  - The memory space (the box or the container below) for variable **age** is allocated automatically when system calls that function.
  - It is deallocated automatically once the function returns.



Concept of Address

```
#include <stdio.h>
int main ()
{
   int  var1;
   char arr[4];

   printf("Address of var1 variable: %x\n", &var1  );
   printf(”Value of arr: %x\n", arr  );
   printf("Address of arr[0] variable: %x\n", &arr[0]  );
   printf("Address of arr[1] variable: %x\n", &arr[1]  );
   printf("Address of arr[2] variable: %x\n", &arr[2]  );
   printf("Address of arr[3] variable: %x\n", &arr[3]  );
   return 0;
} //Demo pointerArray.c
```

Concept of Address

- Output of the program

```
Address of var1 variable: 5b7b8be8
Value of arr: 5b7b8be4
Address of arr[0] variable: 5b7b8be4
Address of arr[1] variable: 5b7b8be5
Address of arr[2] variable: 5b7b8be6
Address of arr[3] variable: 5b7b8be7
```

5b7b8be4	5b7b8be5	5b7b8be6	5b7b8be7
arr[0]	arr[1]	arr[2]	arr[3]

We observe that &arr[0] and arr return the same value.
That means the array name **arr could be used as the array base address.**

Concept of Address

```
#include <stdio.h>
int main ()
{
   int  var1;
   int arr[4];

   printf("Address of var1 variable: %x\n", &var1  );
   printf(”Value of arr: %x\n", arr  );
   printf("Address of arr[0] variable: %x\n", &arr[0]  );
   printf("Address of arr[1] variable: %x\n", &arr[1]  );
   printf("Address of arr[2] variable: %x\n", &arr[2]  );
   printf("Address of arr[3] variable: %x\n", &arr[3]  );
   return 0;
}
```

Concept of Address

- Output of the program

```
Address of var1 variable: 5cc9ebe0
Value of arr: 5cc9ebd0
Address of arr[0] variable: 5cc9ebd0
Address of arr[1] variable: 5cc9ebd4
Address of arr[2] variable: 5cc9ebd8
Address of arr[3] variable: 5cc9ebdc
```

5cc9ebd0	5cc9ebd4	5cc9ebd8	5cc9ebdc
arr[0]	arr[1]	arr[2]	arr[3]

We observe that &arr[0] and arr return the same value.
That means the array name **arr could be used as the array base address.**

Concept of Address

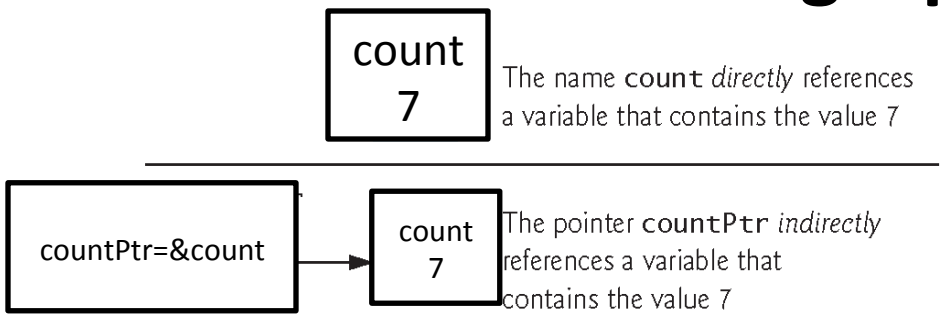
- What are the new output after we change the array type from char array to integer array?
- What changes have you observed ?

Concept of Pointer

- A pointer is a **variable** whose value is the address of another variable.
  - Normally, a regular variable directly contains a specific value.
    - Use variable name to retrieve its value.
  - A pointer, on the other hand, contains an **address** of a variable that contains a specific value.

Concept of Pointer

- In this sense, a variable name **directly** references a value, and a pointer **indirectly** references a value.
- Referencing a value through a pointer is called indirection or **dereferencing a pointer.**



Directly and indirectly referencing a variable.

Define Pointer Variables

- Like any variable or constant, you must define a pointer before you can use it to store any variable address.
- To define/declare a pointer variable,
- type * ptrName;
  - **type** is the pointer's base type; it must be a valid C data type
  - **ptrName** is the name of the pointer variable.
  - The asterisk * you used to declare a pointer.

Define Pointer Variables

E.g.

int *ip;

- pointer to an integer, variable **ip** should hold an address of another integer variable.

double *dp;

- pointer to a double

float *fp;

- pointer to a float

char *ch

- pointer to a character

Initialize Pointer

- int age=10; →memory allocated automatically
- int * iptr; →in main or in the same function as previous statement.
- iptr = &age;
- After this assignment, iptr points to variable **age.**

Initialize Pointer

- In C programming, we usually don't care about the specific value that a pointer variable holds(in binary format), but we really care about which variable it currently points to.

- Next class, we will learn this,
  - int arr[4] = { 1, 2, 4, 9};
  - int *iptr = arr;

Use Pointer

- Typical way to use a pointer variable,
  - (a) we define a pointer variable
  - (b) assign the address of a variable to the pointer
  - (c) finally access the value at the address contained in the pointer variable.
    - We say '**access the value that the pointer points to**'.

Use Pointer

- Step (c) is done by using unary operator * that returns the value of the variable located at the address specified by its operand.
  - Dereferencing a pointer
- E.g.

```
int a = 10;
int *ptr = &a;
*ptr = 100; //this changes the value of variable a
int total=3 + *ptr;
```

*ptr, return the value of variable **a**, **because ptr points to a, or because ptr holds the address of variable a.**

Use Pointer

- ***ptr, is the to dereference pointer ptr.**
  - **That is, it returns the value at the memory address stored in variable ptr.**
  - **More simply, *ptr, returns the value that ptr variable points to.**

  - If you draw this picture above, actually *ptr operation, will follow the arrow from ptr box, and return the value in the box to which the arrow points.

Memory Allocation

- int * iptr; in main or in a function,
- The memory space for holding **a memory address** is allocated automatically for variable iptr.
- When the function returns, that memory space (the box for iptr itself) is deallocated automatically. **But, the memory that iptr points to is not included in this deallocation operation.**

Memory Allocation

- int * iptr; in main or in a function,
  - The box on the right, iptr is a memory cell, used to hold a memory address of another int variable.
  - The variable **iptr** like regular variables in C, has a address associated with it, (shown outside of box)

Before Initialized

- int * iptr; in main or in a function,
- After we create the pointer, before we initialize it, it could contains arbitrary(junk) memory address.
  - In other word, iptr right now points to arbitrary memory location.

- If you dereference it before initialized, you get **Segmentation Fault**.

Demo

- pointer1.c

Summary

- Concept of address
- Concept of pointers
- How to define/declare pointers
- How to use pointers?

Next Class

- Pointer arithmetic