CSE101-Lec#22

Pointers in C

Created By:
Amanpreet Kaur &
Sanjeev Kumar
SME (CSE) LPU

Outline

- Introduction
- Pointer Variable Definitions and Initialization
- Pointer Operators
- Pointer expressions and arithmetic

Introduction

- Pointers
  - Powerful, but difficult to master
  - Simulate call-by-reference
  - Close relationship with arrays and strings

Let’s look to something interesting

Which one to go and grab first?

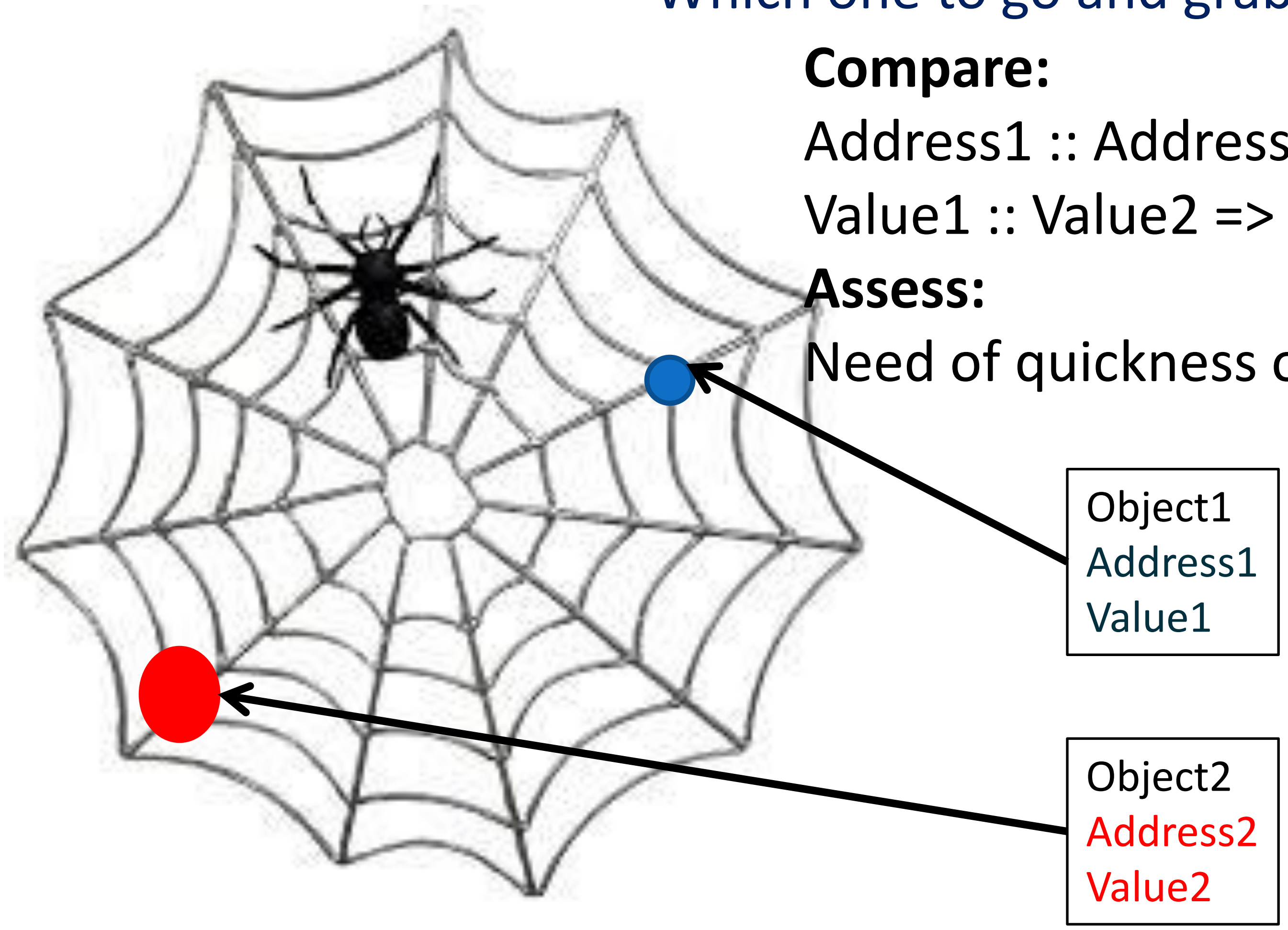
Compare:

Address1 :: Address2 => Nearness

Value1 :: Value2 => Quantity

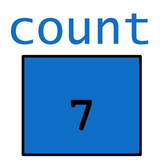
Assess:

Need of quickness or more food

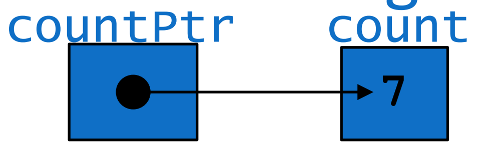


Pointer Variable Definitions and Initialization

- Pointer variables
  - Contain memory addresses as their values
  - Normal variables contain a specific value (direct reference)



  - Pointer is a **variable that contains address of a another variable t**hat has a specific value (indirect reference)
  - Indirection – referencing a pointer value



Pointer Variable Definitions and Initialization

- Pointer definitions
  - `*` used with pointer variables

    ```
    int *myPtr;
    ```

  - Defines a pointer to an `int` (pointer of type `int *`)
  - Multiple pointers require using a `*` before each variable definition

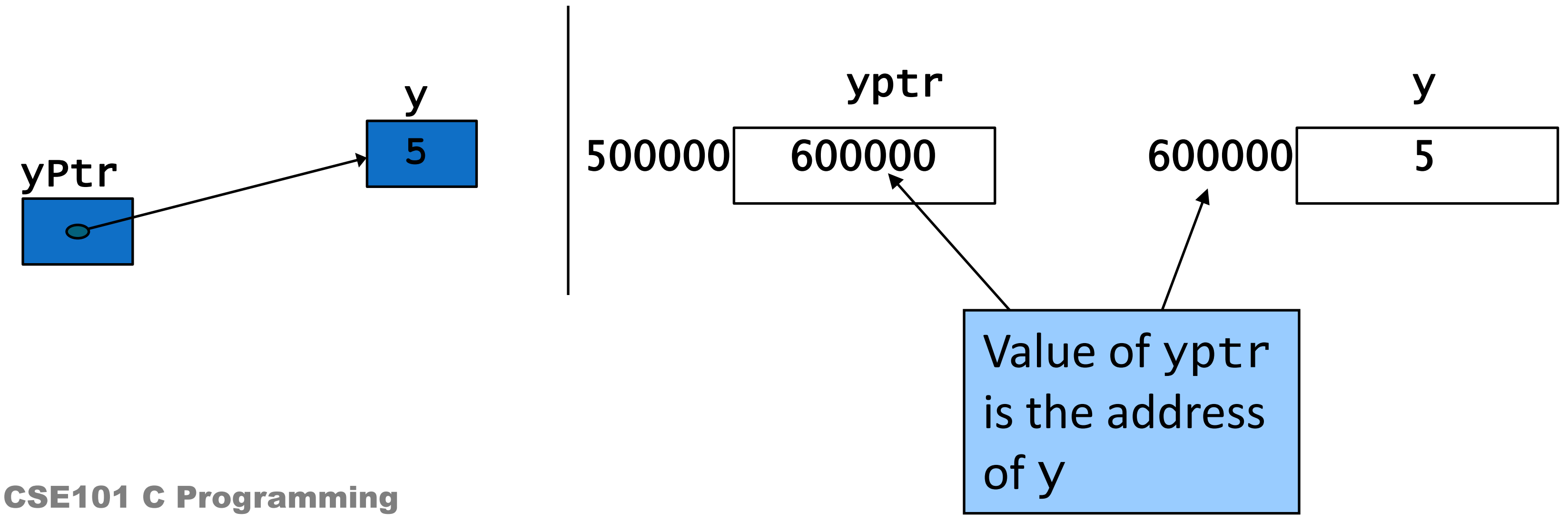
    ```
    int *myPtr1, *myPtr2;
    ```

  - Can define pointers to any data type
  - Initialize pointers to `NULL` or an address

Pointer Operators

- & (address operator)
  - Returns address of operand

```
int y = 5;
int *yPtr;
yPtr = &y;      /* yPtr gets address of y */
yPtr  “points to” y
```



Pointer Operators

- `*` (indirection/dereferencing operator)
  - Returns the value of the variable that it points to.
  - `*yptr` returns value of `y` (because `yptr` points to `y`)
  - `*` can be used for assignment

```
        *yptr = 7;  /* changes y to 7 */
```

Example Code

```c
#include <stdio.h>

int main()
{
   int a;         /* a is an integer */
   int *aPtr;     /* aPtr is a pointer to an integer */

   a = 7;
   aPtr = &a;     /* aPtr set to address of a */

   printf( "The address of a is %p"
           "\nThe value of aPtr is %p", &a, aPtr );

   printf( "\n\nThe value of a is %d"
           "\nThe value of *aPtr is %d", a, *aPtr );

   printf( "\n\nShowing that * and & are complements of "
           "each other\n&*aPtr = %p"
           "\n*&aPtr = %p\n", &*aPtr, *&aPtr );

   return 0; /* indicates successful termination */

} /* end main */
```

This program demonstrates the use of the pointer operators: & and *

Output

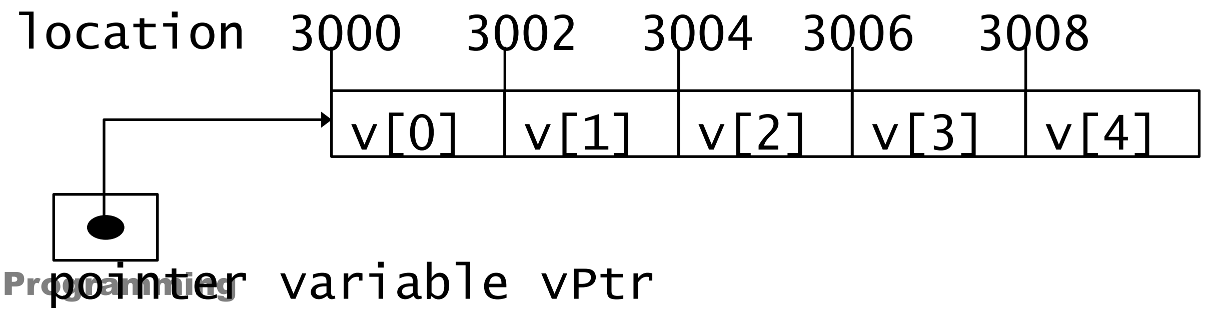
```
The address of a is 0012FF7C
The value of aPtr is 0012FF7C

The value of a is 7
The value of *aPtr is 7

Showing that * and & are complements of each other.
&*aPtr = 0012FF7C
*&aPtr = 0012FF7C
```

Pointer Expressions and Pointer Arithmetic

- Arithmetic operations can be performed on pointers
  - Increment/decrement pointer (++ or --)
  - Add an integer to a pointer(+ or += , - or -=)
  - Pointers may be subtracted from each other
- Operations are meaningless unless performed on an array.

Pointer Expressions and Pointer Arithmetic

- An array `int v[5]` has been defined on machine with 2 byte integers.
  - `int *vPtr;`
  - `vPtr = v;`
  - `vPtr` points to first element `v[0]`
    - at location `3000 (vPtr = 3000)`
  - `vPtr += 2;` sets `vPtr` to `3004`
    - `vPtr` points to `v[2]` (incremented by 2), but the machine has 2 byte `int`s, so it points to address `3004`



Pointer Expressions and Pointer Arithmetic

- Subtracting pointers
  - Returns number of elements from one to the other. If

```
vPtr2 = v[2]; //address 3004
vPtr = v[0]; //address 3000

x = vPtr2 - vPtr
```

assign to x the number of array elements from vPtr to vPtr2, in this case 2.

- Pointer comparison (<, == , >)
  - See which pointer points to the higher-numbered element of the same array.
  - Pointer comparison is used to determine whether pointer is **`NULL`**

Pointer Expressions and Pointer Arithmetic

- Pointers of the same type can be assigned to each other
  - If not the same type, a cast operator must be used
  - Exception: pointer to `void` (type `void *`)
    - Generic pointer, represents any type
    - No casting needed to convert a pointer to `void` pointer
    - `void` pointers cannot be dereferenced.

Pointer Expressions and Pointer Arithmetic

- Increment/decrement
  - Increments the pointer to point next location in array.
    - `++ vPtr; or`
    - `vPtr++;`

```
        vPtr = 3000;
        ++vPtr; //points to 3002
```

  - Decrements the pointer to point the previous element.
    - `--vPtr; or`
    - `vPtr--;`

```
        vPtr = 3002;
        --vPtr; //points to 3000
```

Types of pointers

- Void pointer
- Wild pointer
- Const pointer

Void pointer

- Is a pointer that can hold the address of variables of different data types at different times also called generic pointer.
- The syntax for declaring a void pointer is

```
void *pointer_name;
```

- Here, the keyword **`void`** represents that the pointer can point to value of any data type.
- But before accessing the value through generic pointer by dereferencing it, it must be properly **typecasted**.
- To Print value stored in pointer variable:

```
*(data_type*) pointer_name;
```

Limitations of Void pointers:

- Void pointers cannot be directly dereferences. They need to be appropriately typecasted.
- Pointer arithmetic cannot be performed on void pointers.

Program to show use of void pointer.

```
#include<stdio.h>

void main()
{
int a=10;
char c='R';
void *ptr;
ptr=&a; // assigns address of int variable a to ptr

printf("\n value pointed to by generic pointer is
%d", (*(int *)ptr));

ptr=&c; // assigns address of char variable a to ptr

printf("\n value pointed to by generic pointer is
%d", (*(char *)ptr));

getch();
}
```

Wild pointer

- Pointer which are not initialized during its definition holding some junk value(a valid address) are Wild pointer.
- Example of wild pointer:

```
int *ptr;
```

- Every pointer when it is not initialized is defined as a wild pointer.
- As pointer get initialized, start pointing to some variable its defined as pointer, not a wild one.

Constant Pointers

- A constant pointer, **`ptr`**, is a pointer that is initialized with an address, and cannot point to anything else.
- But we can use **`ptr`** to change the contents of variable pointing to
- Example

```
int value = 22;
int * const ptr = &value;
```

Constant Pointer

- Constant pointer means the pointer is constant.
- Constant pointer is NOT pointer to constant.
- For eg:

  ```
  int * const ptr2
  ```

  indicates that ptr2 is a pointer which is constant. This means that ptr2 cannot be made to point to another integer.
- However the integer pointed by ptr2 can be changed.

Program on constant pointer

```
#include<stdio.h>
void main()
 {
int i = 100,k;

int * const ptr = &i;
*ptr = 200; // value of i is changed.
ptr = &k; //won't compile .
getch();
}
```

Next Lecture

What if we want a function to return more than one values ...??

Pointer as a parameter

cse101@lpu.co.in