Pointers

ESC108A Elements of Computer Science and Engineering

B. Tech. 2017

Course Leaders:

Roopa G.

Ami Rai E.

Chaitra S.

Objectives

- At the end of this lecture, student will be able to
  - Apply the concept of pointers in C programming language

Contents

- Pointers
- Generic pointers

Pointers

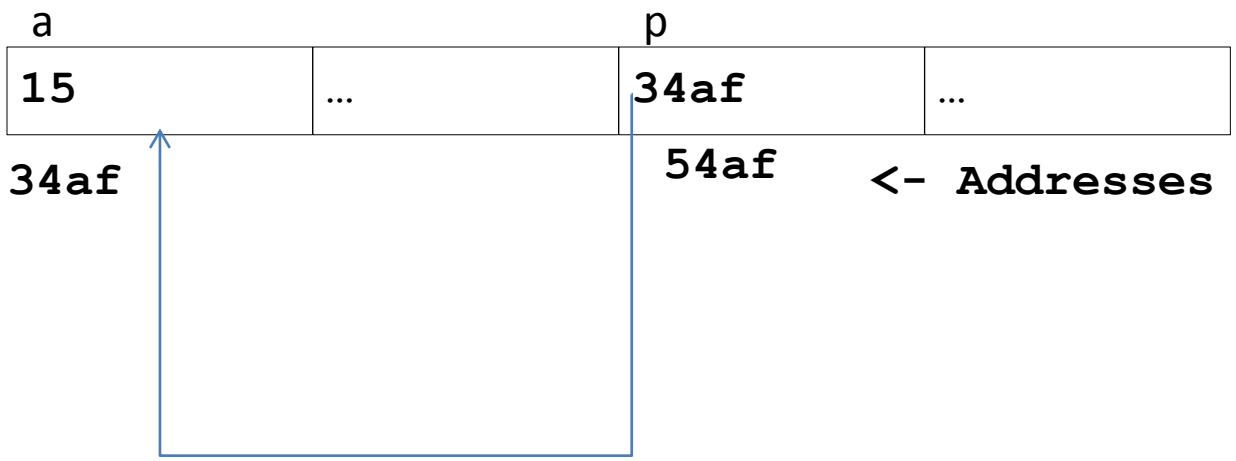
- Pointers are a data type in C
- Variables whose values are memory addresses
  - They hold the address of elements rather than a data value

- Pointers are used to point to
  - Variables of any basic data type
  - An array
  - Functions
  - Structures and union

Pointers and Variables

- Example:

```
int  a = 15;
int *p ;
p = &a;
```

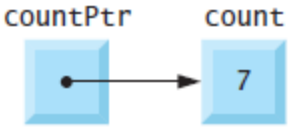


Directly and Indirectly Referencing

- A variable name directly references a value

`count` directly references a variable that contains the value 7

- A pointer indirectly references a value



Pointer `countPtr` indirectly references a variable that contains the value 7

- Referencing a value through a pointer is called indirection

Declaration of Pointer Variables

- A pointer variable is declared by:

  dataType *pointerVarName;

  - The pointer variable *pointerVarName* is used to point to a value of type *dataType*
  - The * before the *pointerVarName* indicates that this is a pointer variable, not a regular variable
  - The * is not a part of the pointer variable name

E.g., int *ptr; //pointer to int

float *ptr2; //pointer to a floating point number

int **p; //pointer to pointer to Integers

Initialisation of Pointer Variables

- Pointers should be initialized either when they're defined or in an assignment statement

- A pointer may be initialized to NULL, 0 or an address
  - A pointer with the value NULL points to nothing
    - NULL is a symbolic constant defined in the <stddef.h> header (and several other headers, such as <stdio.h>)
  - Initializing a pointer to 0 is equivalent to initializing a pointer to NULL, but NULL is preferred

Address Operator

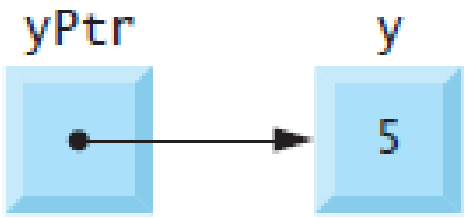
- & or address operator
- A unary operator that returns the address of its operand
- Assuming the definitions

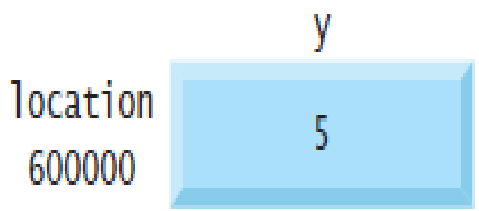
```
int y = 5;
int *yPtr;
```

- the statement

```
yPtr = &y;
```

//assigns the address of the variable y to pointer variable yPtr, Variable yPtr is then said to “point to” y





Indirection Operator

- The unary *(asterisk) operator
- Indirection operator or dereferencing operator
  - To access the value of the variable using the pointer
- Returns the value of the object to which its operand (i.e., a pointer) points

- Example, the statement

```
printf( "%d", *yPtr); //prints the value of variable y, namely 5
```

- The & and * operators are complements of one another

Pointer - Example

```
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 \n The value of aPtr is %p",&a,aptr );
//%p outputs the memory location as a hexadecimal integer
printf( "\n\nThe value of a is %d \n The 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 );
```

Pointer Arithmetic

- Pointers are valid operands in arithmetic expressions, assignment expressions and comparison expressions

- What's ptr + 1?
  - ➔The next memory location!
- What's ptr - 1?
  - ➔The previous memory location!
- What's ptr * 2 and ptr / 2?
  - ➔Invalid operations!!!

Pointer Expressions - Program

```
main ( ){
int a, b, *p1,* p2, x, y, z;
a = 12;
b = 4;
p1 = &a;
p2 = &b;
x = *p1 * *p2 – 6;
y = 4* - *p2 / *p1 + 10;
printf("Address of a = %u\n", p1);
printf("Address of b = %u\n", p2);
printf("\n");
```

```
printf("a = %d, b = %d\n", a, b);
printf("x = %d, y = %d\n", x, y);
*p2 = *p2 + 3;
*p1 = *p2 - 5;
z = *p1 * *p2 – 6;
printf("\n a = %d, b = %d," , a , b);
printf("\n z = %d\n" , z);
}
```

Void Pointers in C - Introduction

- Suppose we have to declare integer pointer, character pointer and float pointer then we need to declare 3 pointer variables
- Instead of declaring different types of pointer variable, it is feasible to declare single pointer variable which can act as integer pointer, character pointer and float pointer

Void Pointers in C - Basics

- In C, General Purpose Pointer is called as Generic Pointer or Void Pointer
- It does not have any data type associated with it
- It can store address of any type of variable
- The compiler has no idea what type of object a void Pointer really points to?

Void Pointers - Declaration

- Declaration of Void Pointer :

```
void * pointer_name;
```

- Void Pointer Example :

```
void *ptr;    // ptr is declared as Void pointer
char cnum;
int inum;
float fnum;
ptr = &cnum;  // ptr has address of character data
ptr = &inum;  // ptr has address of integer data
ptr = &fnum;  // ptr has address of float data
```

Void Pointers - Example

```
main(){
        int i;
        char c;
        void *the_data;

        i = 6;
        c = 'a';

        the_data = &i;
        printf("the_data points to the integer value %d\n", *(int*) the_data);

        the_data = &c;
        printf("the_data now points to the character %c\n", *(char*) the_data);
}
```

Summary

- Pointers store addresses and have same size independent of the data type they point to
- Pointer values changes based on the type that they point to on increment, decrement or difference
- Void Pointer does not have any data type associated with it

Further Reading

Kernighan, B. W. and Richie, D. (1992) *The C Programming Language.* 2nd ed., New Delhi:PHI.