IS1101: Programming and Problem Solving

Pointers - 2

Upul Anuradha

UCSC

Recap…

Type /element	Variable	Pointer Variable
&	Address operator -return address of operand	Address operator -return address of operand
*	-	Indirection/dereferencing operator -return value of whatever operand pointed to
Declaration/Define	int a;	int *aPtr;

Initialization	int a = 5;	int *aPtr = &a;
		int *aPtr = NULL;
		int *aPtr = 0;

Verify address	&a	&aPtr //address of pointer
Verify content	a	aPtr//address of variable a
Value pointed	a	*aPtr//value of variable a

const Qualifier with Pointers

- const qualifier
  - Variable cannot be changed
  - Use const if function does not need to change a variable
  - Attempting to change a const variable produces an error
- const pointers
  - Point to a constant memory location
  - Must be initialized when defined

Pointer to Constant

- Pointer to constant can be declared in following two ways.

  i. `const int *ptr;`

  ii. `int const *ptr;`

- We can change the pointer to point to any other integer variable, but cannot change the value of the object (entity) pointed using pointer ptr.
- The pointer is stored in the read-write area while the object pointed may be in the read-only or read-write area.

Example:

```
3  void main() {
4
5      int i = 10;
6      int j = 20;
7      /* ptr is pointer to constant */
8      const int *ptr = &i;
9
10     printf("ptr: %d\n", *ptr);
11
12     ptr = &j;           /* valid */
13     printf("ptr: %d\n", *ptr);
14
15     *ptr = 100;  /* error: assignment of read-only location */
16
17 }
```

Example:

```
2
3  void main() {
4
5      int i = 10;
6      int j = 20;
7      /* ptr is pointer to constant */
8      const int *ptr = &i;
9
10     printf("ptr: %d\n", *ptr);
11
12     ptr = &j;           /* valid */
13     printf("ptr: %d\n", *ptr);
14
15     *ptr = 100;  /* error: assignment of read-only location */
16
17 }
18
```

Compiler (2) | Resources | Compile Log | Debug | Find Results | Close

Line	Col	File	Message
		D:\WVW Drive\Courses\1st Year - SCS1202 - Program...	**In function 'main':**
15	10	D:\WVW Drive\Courses\1st Year - SCS1202 - Programmi...	[Error] assignment of read-only location '*ptr'

Constant Pointer to Variable

- constant pointer can be declared as follows
  - int *const ptr;
- Above declaration is a constant pointer to an integer variable, means we can change the value of object pointed by pointer, but cannot change the pointer to point another variable.

Example:

```
3  void main() {
4
5      int i = 10;
6      int j = 20;
7
8      /* constant pointer to integer */
9      int *const ptr = &i;
10
11     printf("ptr: %d\n", *ptr);
12
13     *ptr = 100;     /* valid */
14     printf("ptr: %d\n", *ptr);
15
16     ptr = &j;          /* error */
17
18 }
```

Example:

```
3  void main() {
4
5      int i = 10;
6      int j = 20;
7
8      /* constant pointer to integer */
9      int *const ptr = &i;
10
11     printf("ptr: %d\n", *ptr);
12
13     *ptr = 100;     /* valid */
14     printf("ptr: %d\n", *ptr);
15
16     ptr = &j;          /* error */
17
18 }
19
```

Resources | Compile Log | Debug | Find Results | Close

	Message
W Drive\Courses\1st Year - SCS1202 - Program...	In function 'main':
W Drive\Courses\1st Year - SCS1202 - Programmi...	[Error] assignment of read-only variable 'ptr'

Constant Pointer to Constant

- constant pointer to a constant can be declared as follows
  - const int *const ptr;
- Above declaration is a constant pointer to a constant variable which means we cannot change value pointed by the pointer as well as we cannot point the pointer to other variables.

Example:

```
3  void main() {
4
5      int i = 10;
6      int j = 20;
7
8      /* constant pointer to constant integer */
9      const int *const ptr = &i;
10
11     printf("ptr: %d\n", *ptr);
12
13     ptr = &j;      /* error */
14     *ptr = 100;    /* error */
15
16 }
```

Example:

```
3  void main() {
4
5      int i = 10;
6      int j = 20;
7
8      /* constant pointer to constant integer */
9      const int *const ptr = &i;
10
11     printf("ptr: %d\n", *ptr);
12
13     ptr = &j;      /* error */
14     *ptr = 100;    /* error */
15
16 }
17
```

Resources | Compile Log | Debug | Find Results | Close

	Message
WWW Drive\Courses\1st Year - SCS1202 - Program...	In function 'main':
WWW Drive\Courses\1st Year - SCS1202 - Programmi...	[Error] assignment of read-only variable 'ptr'
WWW Drive\Courses\1st Year - SCS1202 - Programmi...	[Error] assignment of read-only location '*ptr'

Summary:

```
const int x = 10;
int *ptr = &x;
```

- regular pointer to a const int x
- x cannot be changed, but *ptr can

```
int x = 10;
const int *ptr = &x;
```

- const pointer to a regular int x
- x, ptr can be changed, but not *ptr

```
int x = 10;
int *const ptr = &x;
```

- constant pointer to a regular int x
- x can be changed, but not *Ptr (cannot pointed to anything else)

```
const int x = 10;
const int *const Ptr = &x;
```

- const pointer to a const int x
- both x and *Ptr cannot be change

sizeof() Operator

- C provides the special unary operator sizeof to determine the size in bytes of an array (or any other data type) during program compilation.
- When applied to the name of an array the sizeof operator returns the total number of bytes in the array as an integer.
- The number of elements in an array also can be determined with sizeof.

sizeof() Operator

- consider the following array definition:

```
double real[ 22 ];
```

- Variables of type double normally are stored in 8 bytes of memory.
- Thus, array real contains a total of 176 bytes (22x8bytes)
- To determine the number of elements in the array, the following expression can be used:

```
sizeof( real ) / sizeof(double )
```

sizeof(): Example

```
 3  void main() {
 4
 5      char c;
 6      short s;
 7      int i;
 8      long l;
 9      float f;
10      double d;
11      int array[20];
12      int *pArrary = array;
13
14      printf("size of a char: %d\n", sizeof(c));
15      printf("size of a short: %d\n", sizeof(s));
16      printf("size of an int: %d\n", sizeof(i));
17      printf("size of a long: %d\n", sizeof(l));
18      printf("size of a float: %d\n", sizeof(f));
19      printf("size of a double: %d\n", sizeof(d));
20      printf("size of an int array with 20 elements: %d\n", sizeof(array));
21
22  }//main()
```

sizeof(): Example

```
3  void main() {
4
5      char c;
6      short s;
7      int i;
8      long l;
9      float f;
10     double d;
11     int array[20];
12     int *pArrary = array;
13
14     printf("size of a char: %d\n", sizeof(c));
15     printf("size of a short: %d\n", sizeof(s));
16     printf("size of an int: %d\n", sizeof(i));
17     printf("size of a long: %d\n", sizeof(l));
18     printf("size of a float: %d\n", sizeof(f));
19     printf("size of a double: %d\n", sizeof(d));
20     printf("size of an int array with 20 elements: %d\n", sizeof(array));
21
22 }//main()
```

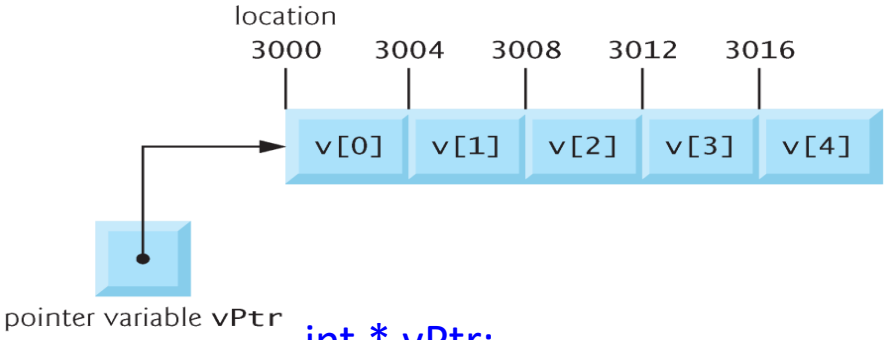
```
size of a char: 1
size of a short: 2
size of an int: 4
size of a long: 4
size of a float: 4
size of a double: 8
size of an int array with 20 elements: 80
```

Arithmetic Operations on Pointers

- 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
  - These operations are meaningless unless them performed on an array

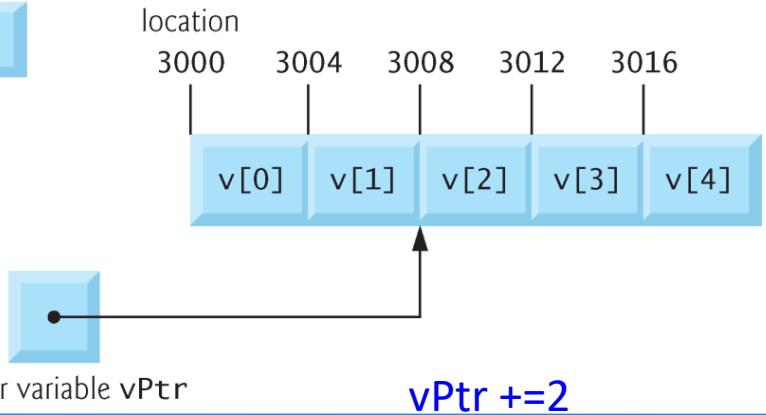
Example:

- 5 elements int array on machine with 4 byte ints
  - vPtr points to first element v[0], say at location 3000 (vPtr = 3000)
  - vPtr += 2; sets vPtr to 3008
  - vPtr points to v[2] (incremented by 2), but the machine has 4 byte ints, so it points to address 3008



int * vPtr;

vPtr = v;



vPtr +=2

Arithmetic Operations on Pointers

- If vPtr had been incremented to 3016, which points to v[4], the statement vPtr -= 4; would set vPtr back to 3000, the beginning of the array.
- If a pointer is being incremented or decremented by one, the increment (++) and decrement (--) operators can be used.
- Either of the statements

  ```
  ++vPtr;  vPtr++;
  ```

  increments the pointer to point to the next location in the array.
- Either of the statements

  ```
  --vPtr;  vPtr--;
  ```

  decrements the pointer to point to the previous element of the array.

Arithmetic Operations on Pointers

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

```
vPtr2  = &v[ 2 ];
vPtr   = &v[ 0 ];
vPtr2 – vPtr;
```

would produce 2.

Arithmetic Operations on Pointers

- Pointer comparison (<, == , >)
  - See which pointer points to the higher numbered array element
  - Also, see if a pointer points to 0

Example

```
3  void main() {
4
5      int *vPtr1, *vPtr2;
6      int array[5] = {10, 20, 30, 40, 50};
7      int temp;
8
9      vPtr1 = array;
10
11     printf("Address of vPtr1:\t%p\n", &vPtr1);
12     printf("Contents of vPtr1:\t%p\n", vPtr1);
13     printf("Address of array[0]:\t%p\n", &array);
14
15     vPtr1 += 2;
16
17     printf("\nAddress of vPtr1 + 2:\t%p\n", &vPtr1);
18     printf("Contents of vPtr1 + 2:\t%p\n", vPtr1);
19
20     vPtr1 += 2;
21
22     printf("\nAddress of vPtr1 + 4:\t%p\n", &vPtr1);
23     printf("Contents of vPtr1 + 4:\t%p\n", vPtr1);
24
25     vPtr2 = &array[2];
26     vPtr1 = &array[0];
27
28     temp = vPtr2 - vPtr1;
29
30     printf("\nContents of temp:\t%d\n", temp);
31
32 }//main()
```

Example

```
Address of vPtr1:         000000000062FE08
Contents of vPtr1:        000000000062FDF0
Address of array[0]:      000000000062FDF0

Address of vPtr1 + 2:     000000000062FE08
Contents of vPtr1 + 2:    000000000062FDF8

Address of vPtr1 + 4:     000000000062FE08
Contents of vPtr1 + 4:    000000000062FE00

Contents of temp:         2
```

```
3  void main() {
4
5      int *vPtr1, *vPtr2;
6      int array[5] = {10, 20, 30, 40, 50};
7      int temp;
8
9      vPtr1 = array;
10
11     printf("Address of vPtr1:\t%p\n", &vPtr1);
12     printf("Contents of vPtr1:\t%p\n", vPtr1);
13     printf("Address of array[0]:\t%p\n", &array);
14
15     vPtr1 += 2;
16
17     printf("\nAddress of vPtr1 + 2:\t%p\n", &vPtr1);
18     printf("Contents of vPtr1 + 2:\t%p\n", vPtr1);
19
20     vPtr1 += 2;
21
22     printf("\nAddress of vPtr1 + 4:\t%p\n", &vPtr1);
23     printf("Contents of vPtr1 + 4:\t%p\n", vPtr1);
24
25     vPtr2 = &array[2];
26     vPtr1 = &array[0];
27
28     temp = vPtr2 - vPtr1;
29
30     printf("\nContents of temp:\t%d\n", temp);
31
32 }//main()
```

Pointers & Arrays

- Arrays and pointers are closely related
  - Array name like a constant pointer
  - Pointers can do array subscripting operations
- Define an array b[5] and a pointer bPtr
  - To set them equal to one another use:
    - bPtr = b;
    - The array name b is actually the address of first element of the array b[5]
    - bPtr = &b[0]; explicitly assigns bPtr to the address of first element of b

Pointers & Arrays

- Consider Element b[3]
  - It can be accessed by *(bPtr + 3)
  - Where * is the offset
    - Called pointer/offset notation
  - Can be accessed by bPtr[3]
    - Called pointer/subscript notation
  - bPtr[3] same as b[3]
  - Can be accessed by performing pointer arithmetic on the array itself
    - *(b + 3)

Pointers & Arrays

- Following program shows the four methods we have discussed for referring to array…
  1. using a counter with the array name
  2. pointer/offset with the array name as a pointer
  3. using a counter with the pointer name
  4. pointer/offset with the pointer

Pointers & Arrays

```
3  void main() {
4
5      int b[5] = {10, 20, 30, 40, 50}; // initialize int array b
6      int *bPtr; // an int pointer
7      int i, offset; //counters
8
9      bPtr = b; // set the pointer to the array b
10
11     printf("array b is printing using a counter...\n");
12     for(i=0; i<5; i++)
13         printf("b[%d] = %d\n", i, b[i]);
14
15     printf("\nprinting using pointer/offset notation whare pointer is the array name...\n");
16     for(offset=0; offset<5; offset++)
17         printf("b[%d] = %d\n", offset, *(b + offset));
18
19     printf("\nprinting using the pointer and a counter...\n");
20     for(i=0; i<5; i++)
21         printf("b[%d] = %d\n", i, bPtr[i]);
22
23     printf("\nprinting using pointer/offset notation..\n");
24     for(offset=0; offset<5; offset++)
25         printf("b[%d] = %d\n", offset, *(bPtr + offset));
26
27 }//main()
```

Answer

```
array b is printing using a counter...
b[0] = 10
b[1] = 20
b[2] = 30
b[3] = 40
b[4] = 50

printing using pointer/offset notation whare pointer is the array name...
b[0] = 10
b[1] = 20
b[2] = 30
b[3] = 40
b[4] = 50

printing using the pointer and a counter...
b[0] = 10
b[1] = 20
b[2] = 30
b[3] = 40
b[4] = 50

printing using pointer/offset notation..
b[0] = 10
b[1] = 20
b[2] = 30
b[3] = 40
b[4] = 50
```

Example

```
3  void main() {
4
5      int b[10] = {10, 20, 30, 40, 50}; // initialize int array b
6      int *bPtr; // an int pointer
7      int i; //counter
8
9      bPtr = b; // set the pointer to the array b
10
11     printf("address of bPtr: %p \t contents of bPtr: %p\n", &bPtr, bPtr);
12
13      printf("address of b: %p \t contents of b[0]: %d %d %d\n", &b, b[0], *bPtr, *b);
14
15     printf("\nI am accessing element b[3]!!!\nLet see how many ways i can do it.\n");
16
17     printf("b[3] = %d\n", b[3]);
18     printf("(bPtr + 3) = %d\n", *(bPtr+3));
19     printf("(b + 3) = %d\n", *(b+3));
20     printf("bPtr[3] = %d\n", bPtr[3]);
21
22     printf("\nprinting all the elements...\n");
23     for(i=0; i<10; i++)
24         printf("b[%d] = %d\n", i, *(bPtr + i));
25
26 }//main()
```

Output

```
address of bPtr: 000000000062FDE8        contents of bPtr: 000000000062FDF0
address of b: 000000000062FDF0   contents of b[0]: 10 10 10

I am accessing element b[3]!!!
Let see how many ways i can do it.
b[3] = 40
(bPtr + 3) = 40
(b + 3) = 40
bPtr[3] = 40

printing all the elements...
b[0] = 10
b[1] = 20
b[2] = 30
b[3] = 40
b[4] = 50
b[5] = 0
b[6] = 0
b[7] = 0
b[8] = 0
b[9] = 0
```

Arrays of Pointers

- Arrays may contain pointers.
- A common use of an array of pointers is to form an array of strings, referred to simply as a string array.
- Each entry in the array is a string, but in C a string is essentially a pointer to its first character.
- So each entry in an array of strings is actually a pointer to the first character of a string.
- Example:

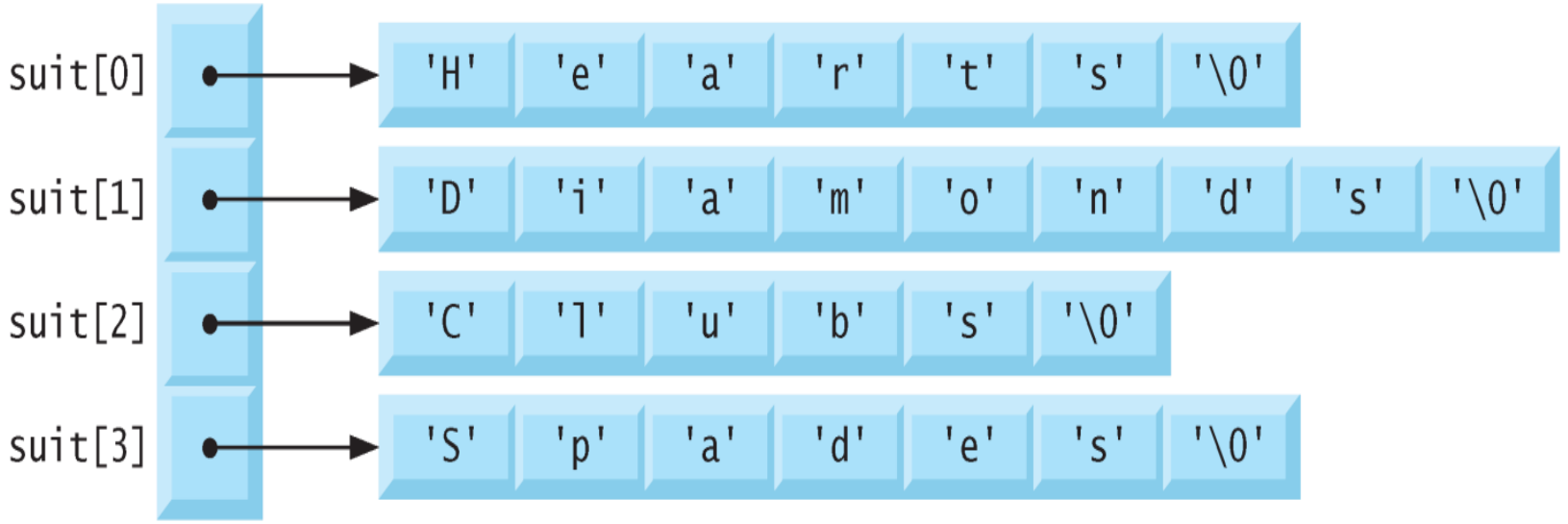
```
//2-D array:
const char suit[ 4 ][ 10 ] = { "Hearts", "Diamonds", "Clubs", "Spades" };

//Array of pointers
const char *suit[ 4 ] = { "Hearts", "Diamonds", "Clubs", "Spades" };
```

Arrays of Pointers

- The suit[4] portion of the definition indicates an array of 4 elements.
- The char * portion of the declaration indicates that each element of array suit is of type "pointer to char."
- Qualifier const indicates that the strings pointed to by each element pointer will not be modified.
- The four strings are 7, 9, 6 and 7 characters long, respectively.
- Although it appears as though these strings are being placed in the suit array, only pointers are actually stored in the array.
- Each pointer points to the first character of its corresponding string.
- Thus, even though the suit array is fixed in size, it provides access to character strings of any length.
- This flexibility is one example of C's powerful data-structuring capabilities.

Arrays of Pointers



- suit array has a fixed size, but strings can be of any size

Example

```c
#include <stdio.h>
#define SIZE 5

void main() {

    char *studentName[10]; // initialize the string array
    int i; //counter

    for(i=0; i<SIZE; i++) {
        printf("Enter student %d name: ", i+1);
        scanf("%s", studentName+i);
        printf("You entered: %s\n", studentName+i);
    }

    printf("\nname of students:\n");
    for(i=0; i<SIZE; i++)
        printf("%d. %s\n", i+1, studentName+i);

}//main()

```

Example

```c
#include <stdio.h>
#define SIZE 5

void main() {

    char *studentName[10]; // initialize the string array
    int i; //counter

    for(i=0; i<SIZE; i++) {
        printf("Enter student %d name: ", i+1);
        scanf("%s", studentName+i);
        printf("You entered: %s\n", studentName+i);
    }

    printf("\nname of students:\n");
    for(i=0; i<SIZE; i++)
        printf("%d. %s\n", i+1, studentName+i);

}//main()
```

```
Enter student 1 name: Kamal
You entered: Kamal
Enter student 2 name: Sunil
You entered: Sunil
Enter student 3 name: Hiruni
You entered: Hiruni
Enter student 4 name: Nimali
You entered: Nimali
Enter student 5 name: Amal
You entered: Amal

name of students:
1. Kamal
2. Sunil
3. Hiruni
4. Nimali
5. Amal
```