Computer Programming

An-Najah N. University
Computer Engineering Department
Luai Malhis, Ph.D,

Pointers

Definition

- Pointers are variables used to hold (and to refer to) memory addresses of other variables.
- Memory addresses is the location of the first byte of memory allocated for that variable or array. Remember that the amount of memory allocated to a variable is dependent on the data type of the variable.
- You can learn the memory address of a variable or array by using the address operator, (the & symbol), before the variable name.

Definition continue

- A pointer variable is designated at time of declaration by a * before the variable name.
  - int *pntr; // or int* pntr;
- After declaration, a * immediately before a pointer name acts as an indirection operator to refer to the value stored in memory, and consequently, may be used to change what is stored in that memory location by an assignment statement.

```
cout << *pntr;
*pntr += 5; //adds 5 to value pointed to by pntr;
```

Pointer Declaration

- Pointers are declared by specifying the type of location (variable) they point at..
- Syntax:

  type * name; // compiler allocates 4 bytes

  For all data types pointer size is 4 bytes
- For example: // compiler allocates 4 bytes for each of the following pointer declarations.

  int *p; char *tp; double *dp;
- When a pointer is declared it points to null (not valid memory location)

Pointer Initialization 1

- Before using a pointer to store or retrieve data from memory location it must be initialized to a valid memory location.
- Valid locations are either:

  exist: (static) variables of the same type

  new: (dynamic) new locations of same type
- Example: int x;   int *p;

  p = &x; // p now points to static location x, this location is accessed by x or *p

  p = new int; // p points to new location this location is accessed by *p only;
- To delete memory allocated by new use: **delete p**;

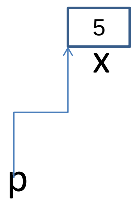
Pointer initialization 2

- Suppose `p' is a pointer, then `*p' is the value in memory location which `p' points to.
- Example:

  int x =5; int *p;

  // make p point to location x

  p = &x; then



The location x can be accesses using *p or x.

cout << *p; // prints 5 on the screen

*p = 10; cout << x ; // prints 10

Pointer Initialization 3

- When a pointer is initialized it must point to a location that can hold data of the same type;
- Example: Given   int *p1; double *p2; char *p3;
  int x;   double y;   char z;
- The following is valid initialization:

  p1 = &x;   p1 = new int;

  p2 = &y;   p2 = new double;

  p3 = &z;   p3 = new char;
- The following is invalid initializations:

  p1 = &z; or p1 = &y; or p1 = new double;

  p2 = &x; or p2 = &z; or p2 = new char

Pointer Initialization 3

- Pointer can only point to one location at a time;
- Example: Given int x =5 ,y = 10;   int *p;

  p = &x; cout << *p;   // prints 5

  p = &y; cout << *p;   // prints 10

  p = new int; cout << *p; //prints random val
- More than one pointer may point to the same location. Example Given: int x; int *p1,*p2;

  p1 = &x;   p2 = &x;   // p1 = p2 = &x;

  *p1 = 10;   then

cout << x;   or cout << *p1;   or cout << *p2;

All prints the same value 10 because they all reference the same location.

Pointer Examples 1

Given int x =5, y =10 ; int *p1, *p2;

what is the output of the following if valid?

```
p1 = x;      // invalid  must use p1 = &x;
y = *p2;    // invalid p2 must point to valid location
p1 = p2 = &x;     cout << *p2;        // 5
p1 =&y;   cout << *p1;    cout << *p2;  //10   5
p1 = &x;    x = y;  cout << *p1;  //  10;
p1 = &x;  p1++;    cout << *p1;  // causes run time
    error because p1 points to invalid location
p2 = &y; *(p2)++; cout << *p2;   // 11
```

Pointer Examples 2

Given: double *dp, d; char c ='A'; What is valid?

```
cin >> dp;                // invalid use of dp
cin >> *dp;               // invalid location of dp
dp =&d;    cin >> *dp;          // valid
d= &dp;   // not valid assignment need cast
dp = new double;    d =*dp;     // valid
*d=dp;       // invalid do not use * with variables
dp = &c;    // invalid  dp and c of different types
dp = new double;  *dp = c;    or c = *dp;   // valid
dp = &d;     *dp = *dp + c;       // valid
```

Pointers and Arrays 1

- Remember that an array name refers to a group of memory addresses, not just a single one.
- The array name holds the address of the first byte of memory allocated to that array.
- Therefore, an array name, without the index, may be considered a pointer to that array.
- Array name is a static pointer and can only point to the location assigned to it.
- Because the array name already acts as a pointer you do not need the address operator, &, to output the first memory address.

Pointers and Arrays 2

- The indirection operator, *, may be used with the static array name to store a value in the first array element.

```
*arrayname = 15;       //stores 15 in arrayname[0]
```

- Values can be stored in subsequent array elements by adding numbers to the array name and employing the indirection operator

```
*(arrayname + 1) =25; //stores 25 in arrayname[1]
*(arrayname + n) =67; //stores 67 in arrayname[n]
arrayname = arrayname +5;  // invalid can not make
arrayname point to any location other than first
*arrayname = *arrayname + 5;   // valid add 5 to
   arrayname[0]
```

Pointer Arithmetic

Integer values may be added to or subtracted from a pointer to move it to different locations

Example: Given int A[5]; int *p;

```
p = A;  // p = &A[0];   // p points to first elem of A
p = A; then  p = p+2;  // now p points to A[2];
p = &A[2];   p--;         // now p points to A[1];
p = A; p = p+5; // p points outside the array A
p = A; p +=2;  p[1] = 18;  // stores 18 in A[3];
p = A+5;  p[-1] = 20;  // stores 20 in A[4];
```

Important Note: With A index is absolute from A.

With p index is relative to pointer location

Pointer Arithmetic 2

- One pointer may also be subtracted from another pointer.
- Example Given int A[10] = {1,4,6,10,12,20,22};

  int *p1 = &A[2]; int *p2 = &A[5];

  cout << p2 – p1; **prints 3** // number of elements between p1 and p2;

  cout << *p2 - *p2; **prints 14** // the difference between the values pointed to between p2, p1
- Pointers can not be added p1 + p2 has no meaning in C. // it is an int number

  p1 = p1 + *p2; // move p1 by the value *p2(+)

  p1 -= *p2; // move p1 by the value *p2 (-)

Other Pointer Operations

- Pointers can be initialized at time of declaration.

  float *fp= &fvar; // fvar must exist 1st
- Pointers, and memory addresses, may also be compared using the relational operators.

  p1 < p2 ; p1 != p2; p1 ==p2
- Static pointers can not be incremented
- Int A[10]; A= A+5; or A++; or A--; A[-1]; are invalid operations
- Since A is a static pointer it must always point to the same location (first element of the array) and can not be moved.

Static Arrays and Pointers 1

int A[10] = {12,4,7,10, 13, 16};

int *p1 = A; Then

A[0], *A, p1[0] and *p1 all used to refer to the first element of the array (12 in this case).

int *p2 = A + 2; now p2 points to A[2]. Then

A[1] and *(A+1) , p2[-1], and *(p2-1) all refer to the second element of the array (4 in this case)

Int *p3 = &A[4]; p3 = p3-2; Then

A[2], *(A+2), p3[0], and *p3 all refer to the third element in the array (7 in this case).

Static Arrays and pointers 2

Given int A[10] = {12,4,7,10, 13, 16}; int *p = A;

Then we can print array elements as follows:

for (int i =0; i < 10; i++) cout << A[i];

for (int i =0; i < 10; i++) cout << *(A+i);

for (int i =0; i < 10; i++) cout << p[i];

for (int i =0; i < 10; i++) cout << *(p+i); //not *p+i;

for (int i =0; i < 10; i++) {cout << *p; p++;}

Note that:

for (int i =0; i < 10; i++) {cout << *A; A++;} is invalid

Dynamic Arrays

- So far we have allocated arrays statically.
  Example int AI[20]. In which case array size is fixed at compile time to 20 and can not change.
- However, C allows us to declare a pointer and make it point to consecutive memory locations (array) at run time.
- Example int *ap; Then we write:
  ap = new int[30]; where we allocate array called ap and we can use ap to refere to any element in the array like ap[i]; i< 30; or *ap, or *(ap+i)

Dynamic Arrays 2

- Array size may be entered by the user at run time.
- Example: double *dp; int size;
  cout << “please enter array size:”;
  cin >> size;
  dp = new double[size];
- You can access array elements using static method or dynamic method, dp[i] or *dp; dp++; …etc
- To delete memory allocated to the array we use the statement delete [] dp;

Dynamic Arrays 3

- Be carful: In dynamic arrays if array pointer is made to point accidentally outside the array then the pointer can not be made to point again to the array.
- Example given int *p = new int[10]; int x;
  p = p+15; p now points outside the array and can not make it point to the array again.
  p = &x; p now points to memory location x and can not make it point to the array again.
  p = new int; p now points to new memory and can not be made to point the array again.

Example 1

- Given int x; int *p1, *p2; make p point to x;
  and make p2 point to new location.
  solution: <u>p1 = &x</u>;   <u>p2 = new int;</u>
- Given: double *dp, d; Which one is valid

a. <u>dp =&d;</u>   b. d = *dp;   c. d=&dp;

- Given int s1[5] = {3,5,6};   int s2[5];   int *p;
  which of the following is valid and why?

a. s2 = s1;   b.   p= &s1[6];
c.   s2 = *p;   <u>d.   p = s1+1;</u>

Examples 2

- Given int x[]={0,2,4,6,8,10}; int *y;
  y = &x[2];   *y += 1;   *(y+2) +=2;
  what is the new array content<u>: {0,2,5,6,10,10};</u>
- Given float x[10]={2.5,3.5,4.5,20.0,0.0,6.5};
  float *p= x + (int) *x; float sum=0.0;   int i =0;
  while(*p && i < 10){ sum+=*p++; i++};
  cout<<sum;   what is printed?   <u>24.5</u>
- Given int x[6] = {1,5,3,4,0}; int *p= x+1;
  for(int i= *p; i>=0; i -=*x){cout<<"hello";}
  how many time hello is printed <u>5</u>

Examples 3

- Given an Array A of 100 int values, and int *p;
  Write code to:

Print array content using static pointer A

```
  for (int i = 0; i < 100; i++)
        cout << A[i];   // cout << *(A+i);
```

Print array content using pointer p;

```
 for(int i =0, p = A; i < 100; i++)
     cout << p[i];   //cout << *(p+i);
   // {cout < *p;   p++;}  This is not possible with A
```

Examples 4

Given an Array A of 100 int values, and int *p;

- Use p to replace all even values in A with 0.
  ```
  for(int i =0, p = A; i < 100; i++)
  { if (*p%2 == 0) *p = 0; p++;}
  ```
- Use pointer p to print all elements in the array in reverse location 99 ,98,…0
  ```
  for(int i =0, p = &A[99]; i < 100; i++)
  {cout << *p << endl; p--;}
  ```
- Use p to sum all elements in A until first 0.
  ```
  for(int sum =0, p = A; *p; p++) sum = *p;
  ```

Examples 5

Given an Array A of 100 int values, and int *p;

Declare two pointers p1 and p2

(1) Make p1 point to the first negative value in A.

(2) Make p2 point to the last even value in A.

(3) Find the count elems. between p1 and p2 **inclusive**.

(4) Find the sum of values between p1 and p2.

Solution: int *p1, *p2;

(1) p1 = A; while(1) { if (*p1>=0) p1++; else break;}

(2) p2 = A+99; while(*p2%2) p2--;

(3) int elemcount = p2 – p1+1;

(4) For (int sum=0, p = p1; p <=p2;p++) sum += *p;

Examples 6

Given an Array A of 100 int values, and int *p;

Declare two pointers p1 and p2 make p1 and p2 point to array locations index 20 and index 50 respectively, then (1) use pointer p to print all odd values between p1 and p2 **exclusively.** Then (2) copy all values between p1 and p2 into new dynamically allocated array.

(1) int * p1 = &A[20]; int *p2 = &A[50];

for (p = p1+1; p !=p2; p++) if (*p%2) cout << *p;

(2) int *t = new int [p2 – p1 -1];

for (p = p1+1; p !=p2;) *t++ = *p++;

Example 7

- Declare a dynamic array of N double values where N is entered by the user then read the N double values and store them in the array. Multiply each value in the array by 2. Then print the new content array in reverse.

Solution:

int N; cin >> N; int *p = new double [N];

for (int *t =p, int i =0; i < N; i++, t++) cin >> *t;

for (int *t =p, int i =0; i < N; i++, t++) *t+=2;

for (int *t =p+N; t >= p;) cout << *t-- << endl;

Important points

- Pointer declaration: type * name;
- Pointer must point to a location before to be used.
- Pointer type and location type must be the same.
- Pointers can be subtracted. But can not be added.
- Static arrays (pointer) point to the first element.
- Static pointers can not change location it points to.
- Dynamic pointers can point to any valid location.
- Dynamic pointer can not only point to two or more locations at the same time.
- Two or more pointers can point to same location.