CS101 Introduction to computing

Array and Pointer

A. Sahu and S. V .Rao

Dept of Comp. Sc. & Engg.

Indian Institute of Technology Guwahati

Outline

- Array Definition, Declaration, Use
- Array Examples
- Pointer
  - Memory access
  - Access using pointer
- Basic Pointer Arithmetic

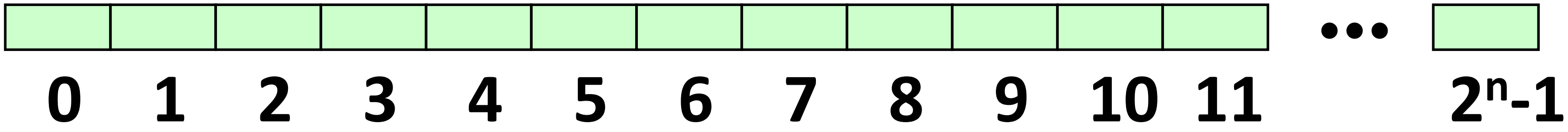
Pointers

- Special case of bounded-size natural numbers
  - Maximum memory limited by processor word-size
  - 2^{32} bytes = 4GB, 2^{64} bytes = 16 exabytes
- A pointer is just another kind of value
  - A basic type in C

```
int *ptr;
```

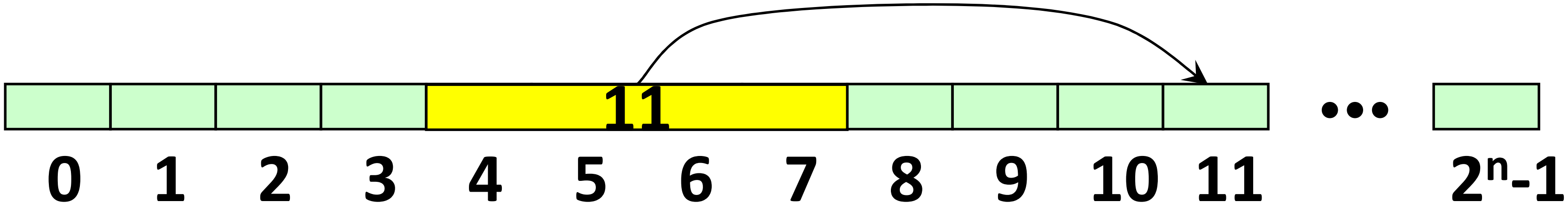
The variable “ptr” stores a pointer to an “int”.

Recall: Memory Organization



- All modern processors have memories organized as sequence of *numbered bytes*
  - Many (but not all) are linear sequences
- Definitions:–
  - *Byte:* an 8-bit memory cell capable of storing a value in range 0 … 255
  - *Address:* number by which a memory cell is identified

Definition – *Pointer*



- A *value* indicating the *number* of (the first byte of) a data object
  - Also called an *Address* or a *Location*
- Usually 2, 4, or 8 bytes, depending upon machine architecture
  - Now a days: Address is **48 bit** and it is **long** type
  - int A; sizeof(int*) ;   printf(“%p”,&A);

Pointer Operations in C

- Creation

  & *variable* Returns variable’s memory address

- Dereference

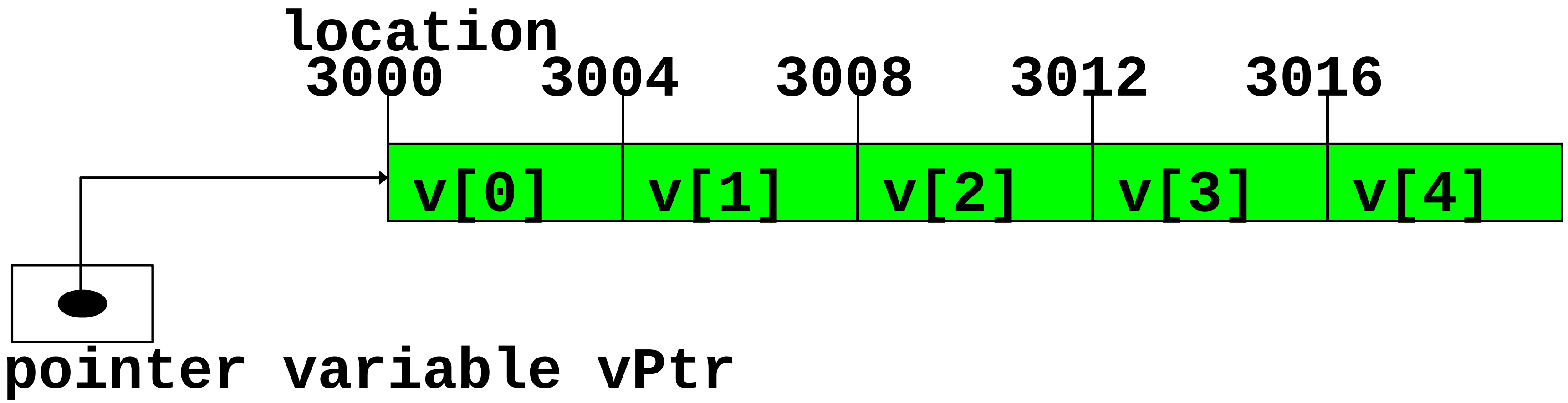
  * *Pointer* Returns contents stored at address

```
int A, B;
int *ptr;
ptr=&A;   // Creation
B=*(ptr); //Dereference
```

Pointer and Array

- 5 element **int** array with 4 byte **int**s
- **vPtr** points to first element **v[0]**
  - 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 **int**s, so it points to address **3008**

```
int  V[5];
int  *vPtr=V;
```



Demo-Ptr-vs Array

```
main() {
   int V[5]={2,1,4,6,3};
   int *vPtr=V;  /* V=vPtr;//notLegal*/
   printf("V=%p, &V[0]=%p\n",
                 V, &V[0]);
   printf("V[0]=%d   *vPtr=%d\n",
                V[0], *vPtr);
   printf("V[2]=%d   *(vPtr+2)=%d\n",
                V[2], *(vPtr+2));
   prntf(“%d %d %d %d”,
         V[2], 2[V], vPtr[2], 2[VPtr]);
}
```

`V[2],2[V],vPtr[2] and 2[VPtr] are same`

*(V+2), *(2+V), *(vPtr+2), *(2+vPtr)

Pointer Arithmetic

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

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

    `vPtr2 - vPtr` would produce 2
- Pointer comparison (**<**, **==** , **>**)
  - See which pointer points to the higher numbered array element
  - Also, see if a pointer points to **`0`**

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

Array and Pointer

- Arrays and pointers closely related
  - **Array name like a constant pointer**
  - Pointers can do array subscripting operations

```
  int   b[5]={2,8,9,5,3};
  int  *bPtr;
        bPtr=&b[1];
```

b[0]	2	b	bPtr-1
b[1]	8	b+1	bPtr
b[2]	9	b+2	bPtr+1
b[3]	5	b+3	bPtr+2
b[4]	3	b+4	bPtr+3

Array and Pointer

- Arrays and pointers closely related
  - Array name like a constant pointer
  - Pointers can do array subscripting operations

```
double   b[5]={2,8,9,5,3};
double   *bPtr;
      bPtr=&b[1];
```

b[0]	2	b	bPtr-1
b[1]	8	b+1	bPtr
b[2]	9	b+2	bPtr+1
b[3]	5	b+3	bPtr+2
b[4]	3	b+4	bPtr+3

Array and Pointer

```
int  b[5];
int *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 address of first element of **b**

Array and Pointer

```
int  b[5];
int *bPtr;
```

- Element **`b[3]`**:
  - Can be accessed by **`*( bPtr + 3 )`**
  - Where **n** is the offset. Called pointer/offset notation

Array and Pointer

```
 int  b[5];
 int *bPtr;
```

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

Array and Pointer

- `int A[10];`
`int *p;`
  - *Type* of `A` is `int *`
  - `p = A; //`legal assignment
  - `A = p;` **// not legal assignment**
  - `*p` refers to `A[0]`
    `*(p + n)` refers to `A[n]`
  - `p = &A[5];` is the same as `p = A+5;`

Array Name is pointer but const
Ptr: == >
`int * const A;`

Array and Pointer

```
 int  A[5],i,S=0;
 int *APtr;

for(i=0;i<5;i++){
   S=S+A[i];
 }
```

```
 int  A[5],i,S=0;
 int *APtr;

for(i=0;i<5;i++){
   S=S+*(A+i);
 }
```

Array and Pointer

```
 int  A[5],i,S=0;
 int *APtr;

for(i=0;i<5;i++){
   S=S+A[i];
 }
```

```
 int  A[5],i,S=0;
 int *APtr;
Aptr=A;
for(i=0;i<5;i++){
   S=S+*(APtr);
   APtr++;
 }
```

Increment address (value of Aptr) by 4 each time

Array and Pointer

```
  int   i;
  char A[5],S=0;
  char *APtr;

for(i=0;i<5;i++){
    S=S+A[i];
 }
```

```
  int   i;
  char A[5],S=0;
  char *APtr;
Aptr=A;
for(i=0;i<5;i++){
    S=S+*(APtr);
    APtr++;
  }
```

Increment address (value of Aptr) by 1 each time

Array and Pointer

```
 int  i;
 long A[5],S=0;
 long *APtr;

for(i=0;i<5;i++){
   S=S+A[i];
 }
```

```
 int  i;
 long A[5],S=0;
 long *APtr;
Aptr=A;
for(i=0;i<5;i++){
   S=S+*(APtr);
   APtr++;
 }
```

Increment address (value of Aptr) by 8 each time

Pointer Arithmetic

```
int *p, *q;
q = p + 1;
```

- Construct a pointer to the next *integer* after ***p** and assign it to **q**

```
double *p, *r;
int n;
r = p + n;
```

- Construct a pointer to a *double* that is **n** *doubles* beyond ***p**, and assign it to **r**
- **n** may be negative

Pointer Arithmetic (continued)

```
long int *p, *q;
p++; q--;
```

- Increment **p** to point to the next **long int**; decrement **q** to point to the previous **long int**

```
float *p, *q;
int n;
n = p – q;
```

- **n** is the number of floats between ***p** and ***q**; i.e., what would be added to **q** to get **p**

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 meaningless unless performed on an array

Arrays and Pointers

- `double A[10];` *vs.* `double *A;`
- *Only* difference:–
  - **`double A[10]`** sets aside *ten* units of memory, each large enough to hold a **`double`**
  - **`double *A`** sets aside *one* pointer-sized unit of memory
    - You are expected to come up with the memory elsewhere!
  - Note:– all pointer variables are the same size in any given machine architecture
    - Regardless of what types they point to

Array-Array Assignment

- *C* does *not* assign arrays to each other
- *E.g,*

  -
    ```
    double A[10], B[10];

    A=B;//Not a valid Statement
    ```
    - assigns the pointer value **B** to the pointer value **A**
    - Contents of array **A** are untouched

Thanks