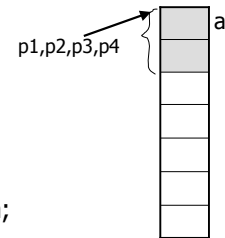


## Pointer Arithmetic and Arrays

Alon Efrat  
Computer Science Department  
University of Arizona

## Pointer Review

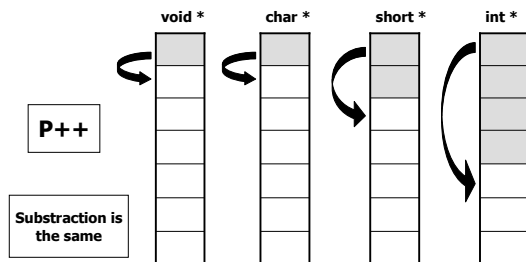
- void \*p0 ;
- char \*p1 ;
- short \*p2 ;
- int \*p3 ;
- int a;
- p1=p2=p3=p4 =&a;



2

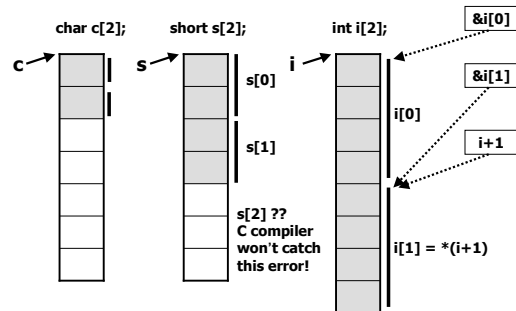
## Pointer Arithmetic

- Depend on base type of the pointer



3

## Array Declaration



4

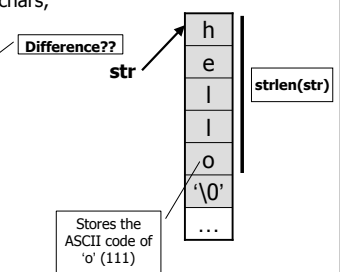
## Array Initialization

- int a[3];
  - The size is required by compiler to decide the amount of memory to allocate
  - For local array, all elements' default initial values are undefined values
  - For global array, all elements' default initial values are 0's
- int a[] = {10, 20, 30};
  - When initialization is given, size can be omitted

5

## Char Array - String

- A string - an array of chars, ending with '\0';
- char str[] = "hello";
- char \*str = "hello";
  - /\* The '\0' is added automatically.
- char str2[6];
- strcpy(str2, "hello");
- int len = strlen(str);
- printf("%s\n", str);
- Printf("%c\n", str[2]);



6

## Example: strlen()

```

/* strlen: returns length of string s
(not including the '\0' ) */
int strlen(char *s)
{
    int n;
    for (n=0; *s != '\0'; s++)
        n++;
    return n;
}

```

7

## Array Argument

- As formal argument, `char *s` and `char s[]` are equivalent
- `int foo(char *s)`, `foo` can regard `s` as:
  - A pointer, or
  - An char array (string)
  - How does `foo` know the size of array `s`?
- Passing partial array: `foo(&str[2], &str[10]);`  
(work on `str[2]`, `str[3]... str[10]`), or
- Passing the length of the array, or
- Work on all elements till the `'\0'` at the end of the string.

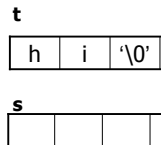
8

## Example: strcpy()

```

/* strcpy(): array version */
void strcpy(char *s, char *t){
    int i=0;
    while ((s[i]=t[i]) != '\0') i++;
}
/* strcpy(): pointer version */
void strcpy(char *s, char *t){
    while ((*s=*t) != '\0'){ s++; t++;}
}
/* strcpy(): pointer version 2 */
void strcpy(char *s, char *t)
{
    while ((*s++ = *t++) != '\0');
}

```



9

## A quick look at printf/scanf

```

char s[]="Hello world\n";
Printf( "%s", s);

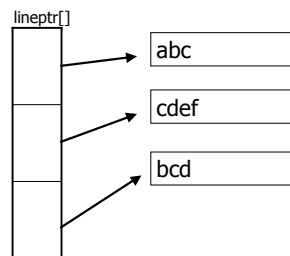
char buff[100];
Scanf("%s", s); /* Note - no & */

```

10

## Pointer Array

- `char *lineptr[3]=`
- {
- "abc", "cdef", "bcd"
- }



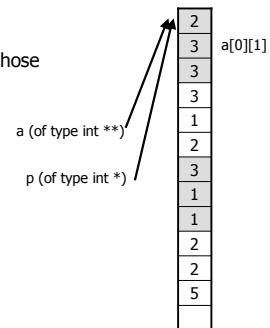
11

## Multi-dimensional Arrays

- Array of arrays
  - One dimensional array whose element is also array
- ```

int a[4][3] = {
    {2, 3, 3},
    {3, 1, 2},
    {3, 1, 1},
    {2, 2, 5}
};
int *p = (int *)a;

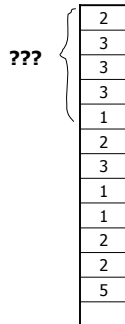
```
- Size of various pointers



12

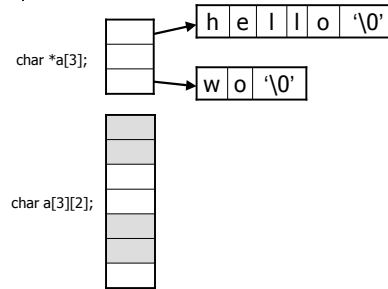
## Multi-dimensional Array as Arguments

- int foo(int a[4][3]); -- OK
  - int foo(int a[][3]); -- OK
  - int foo(int a[][]); -- Error
- Offset formula for m\*n-matrix
  - $a[i][j] = *(a + i + j*m)$



13

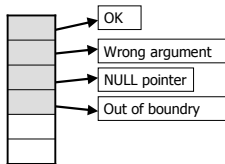
## 2-D Array .vs. Pointer Array



14

## String Array

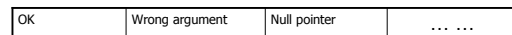
- char \*msg[] = {"OK", "Wrong argument", "NULL pointer", "Out of boundry"}
- printf("%s", msg[1]);



15

## String Array (cont.)

- char msg[][20] = {"OK", "Wrong argument", "NULL pointer", "Out of boundry"}
- printf("%s", msg[1]);



16

## argv and argc

- Assume you write from the command line
  - cat file1 file2 file3
- argv, argc are initialized by the OS
- void main(int argc, char \* argv)
- {
  - argc = 3
  - argv[0] = "cat", argv[1] = "file1",
  - argv[2] = "file2", argv[3] = file[3]
- }
- Local variables to main. Argv[0] is the name of the program. Argc is the number of arguments in the command line.

17

## Acknowledgement

- John H. Hartman, *Classnotes for Csc352-Spring03*, CS Dept., University of Arizona, 2003
- Brian W. Kernighan, Dennis M. Ritchie, *The C Programming Language (2nd Ed.)*, Prentice Hall, 1988

18