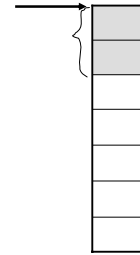


# Pointer Arithmetic and Arrays

Stanley Yao  
Computer Science Department  
University of Arizona

## Pointer Review

- Starting address
- Base type
  - void \*
  - \* operator
- Size of the value
- Size of the pointer
- Size of the value

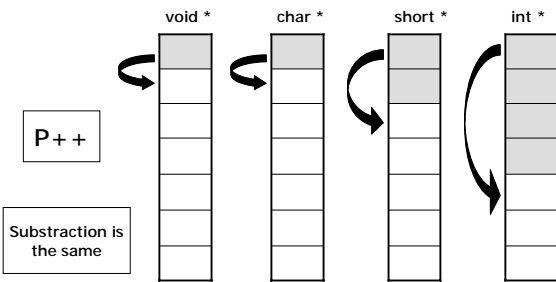


Csc352-Summer03, Stanley Yao

2

## Pointer Arithmetic

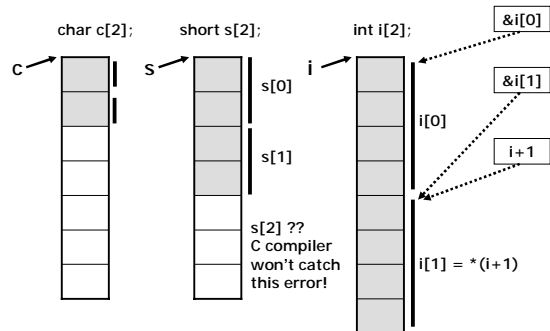
- Depend on base type of the pointer



Csc352-Summer03, Stanley Yao

3

## Array Declaration



Csc352-Summer03, Stanley Yao

4

## Array Initialization

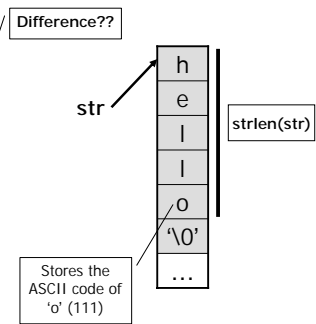
- int a[3];
  - The size is required by compiler to decide the amount of memory to allocate
  - For local array, all elements' default initial values are undefined values
  - For global array, all elements' default initial values are 0's
- int a[] = {10, 20, 30};
  - When initialization is given, size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

Csc352-Summer03, Stanley Yao

5

## Char Array - String

- char str[] = "hello";
- char \*str = "hello";
- char str[6];
- strcpy(str, "hello");
- int len = strlen(str);
- printf("%s\n", str);
- Printf("%c\n", str[2]);



Csc352-Summer03, Stanley Yao

6

## Example: strlen()

```

/* strlen: return length of string s */
int strlen(char *s)
{
    int n;
    for (n=0; *s != '\0'; s++)
        n++;
    return n;
}

```

Csc352-Summer03, Stanley Yao

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])`;
  - Passing subarray of `str` starting from `str[2]`
  - In `foo`, `s[-1]` is actually `str[1]`

Csc352-Summer03, Stanley Yao

8

## Array Argument (cont.)

- In `int foo(char *s)`, the size of array `s` is unknown
  - `s` is only a pointer
  - Pass another parameter indicating size if needed
  - Use `s[0]`
  - etc.

Csc352-Summer03, Stanley Yao

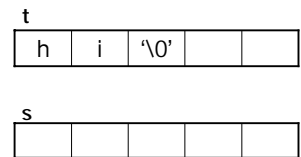
9

## 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');
}

```

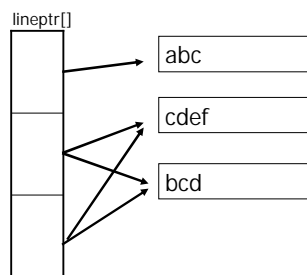


Csc352-Summer03, Stanley Yao

10

## Pointer Array

- `char *lineptr[3];`
- `lineptr[0] = "abc";`
- `lineptr[1] = "cdef";`
- `lineptr[2] = "bcd";`



Csc352-Summer03, Stanley Yao

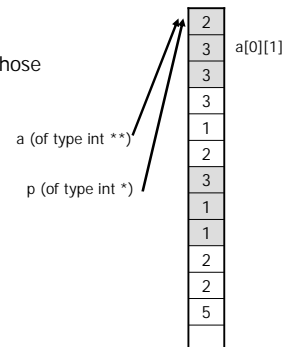
11

## 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;

```

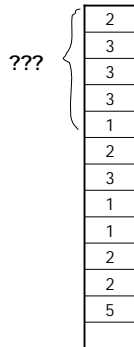


Csc352-Summer03, Stanley Yao

12

## Multi-dimensional Array as Arguments

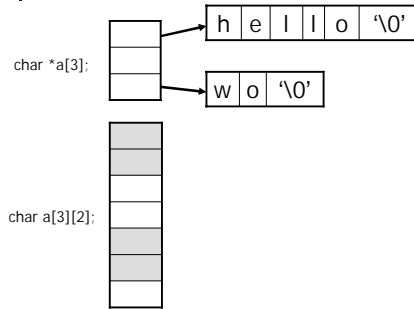
- int foo(int a[4][3]); -- OK
- int foo(int a[][3]); -- OK
- int foo(int a[][]); -- Error
- Which is a[1][0]???
- Only the first dimension can be omitted



Csc352-Summer03, Stanley Yao

13

## 2-D Array .vs. Pointer Array

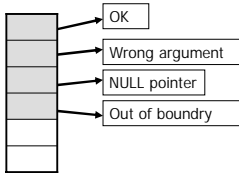


Csc352-Summer03, Stanley Yao

14

## String Array

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

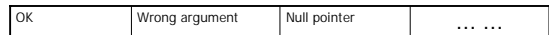


Csc352-Summer03, Stanley Yao

15

## String Array (cont.)

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



Csc352-Summer03, Stanley Yao

16

## Acknowledgement

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

Csc352-Summer03, Stanley Yao

17