COMP 3221

Microprocessors and Embedded Systems

Lecture 3: C-Language Review - II

http://www.cse.unsw.edu.au/~cs3221

August, 2003

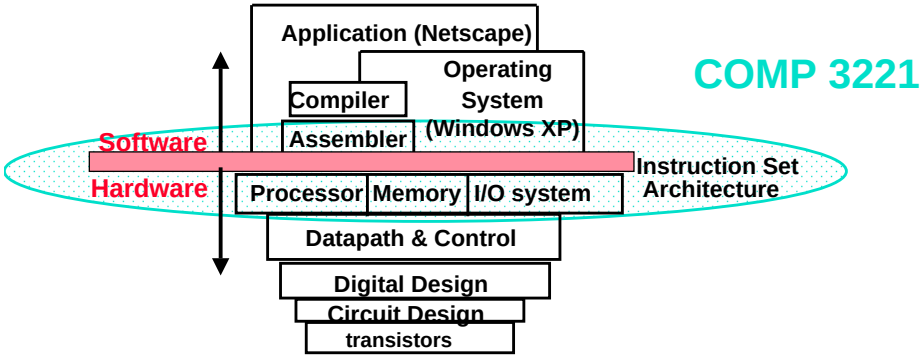
Saeid Nooshabadi

Saeid@unsw.edu.au

Overview

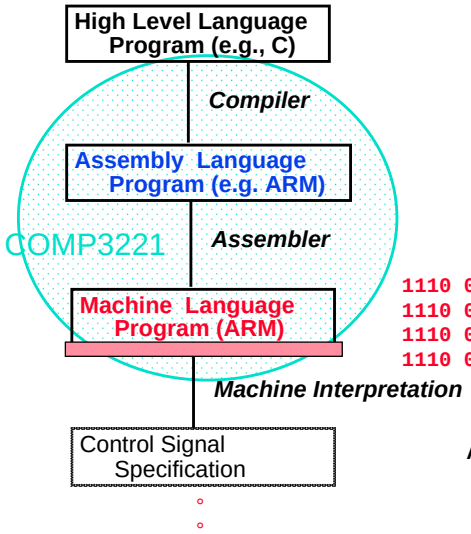
- Common Pointer Mistakes
- Operators
- World's Last C Bug
- Examples

Review: What is Subject about?



- Coordination of many *levels of abstraction*

Review: Programming Levels of Representation




```
temp = v[k];
v[k] = v[k+1];
v[k+1] = temp;
```

```
ldr  r0 , [r2, #0]
ldr  r1 , [r2, #4]
str  r1 , [r2, #0]
str  r0 , [r2, #4]
```

```
1110 0101 1001 0010 0000 0000 0000 0000
1110 0101 1001 0010 0000 0000 0000 0100
1110 0101 1000 0010 0001 0000 0000 0000
1110 0101 1000 0010 0001 0000 0000 0100
```

ALUOP[0:3] <= InstReg[9:11] & MASK

Review: Address v. Value (#1/2)

- Consider memory to be a single huge array:
  - Each cell of the array has an address associated with it.
  - Each cell also stores some value.
- Don't confuse the address referring to a memory location with the value stored in that location.

Review: Pointers in C (#1/6)

- An address refers to a particular memory location. In other words, it *points* to a memory location.
- Pointer: High Level Language (in this case C) way of representing a memory address.
- More specifically, a C variable can contain a pointer to something else. It actually stores the memory address that something else is stored at.

Review: Pointer Arithmetic (4/4)

- So what's valid pointer arithmetic?
  - Add an integer to a pointer.
  - Subtract 2 pointers (in the same array).
  - Compare pointers (<, >, etc.).
  - Compare pointer to NULL (indicates that the pointer points to nothing).
- Everything else is illegal since it makes no sense:
  - adding two pointers, multiplying pointers, etc.

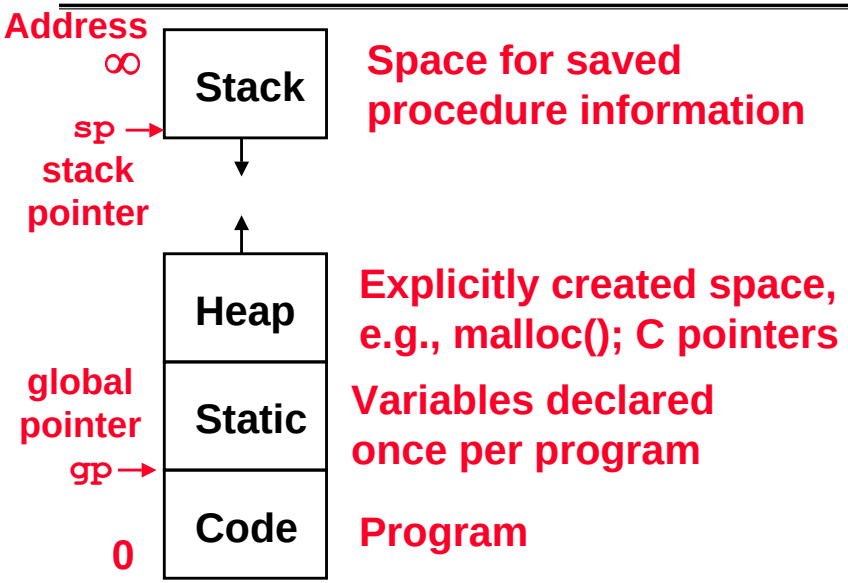
Review: Arrays

- Declaration:

```
int ar[12];
```

  declares a 12-element integer array.
- Accessing elements:
  - `ar[num];` returns the numth element.
  - `ar` is a pointer
  - `ar[0]` is the same as `*ar`
  - `ar[2]` is the same as `*(ar+2)`

Review: C memory allocation



Review: Arguments to Functions

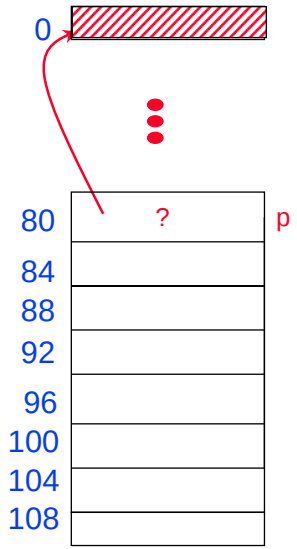
- Arguments can be:
  - passed by value: Make a copy of the original argument (doesn't really affect types such as integers).
  - passed by reference: Pass a pointer, so the called function makes modifications to the original struct.
- Passing by reference can be dangerous, so be careful.

Review: Common Pointer Mistakes (#1/2)

- Declare and write:

```
int *p;
*p = 10; /* WRONG */
```

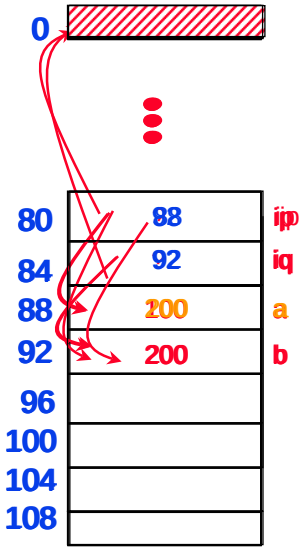
- What address is in p?
  - Answer: NULL; C defines that memory address 0 (same as NULL) is not valid to write to.
- Remember to `malloc` first.



Review: Common Pointer Mistakes (#2/2)

- Copy pointers v. values:

```
int *ip, *iq, a = 100, b = 200;
ip = &a; iq = &b;
*ip = *iq; /* what changed? */
ip = iq;   /* what changed? */
```



World's Last C Bug

- If you remember nothing else, remember this:

```
while (1) {
    status = GetRadarInfo();
    if (status = 1) {
        LaunchMissiles();
    }
}
```

= is used instead of ==

World's Last C Bug Improved!

```
launches = 0;
while (1){
    status = GetRadarInfo();
    if (status = 1){
        LaunchMissiles();
        launches++;
    }
    if (launches > 1){
        apologize();
    }
}
```

Steve Litt: www.troubleshooters.com

Example #1:

- How many bugs in this code?

```
#include <stdio.h>
int main ( ) {
    int numAs; /* counts A's in the input */
    char c;
    while (c = getchar ( ) != EOF) {
/* getchar returns EOF if no more chars to read. */
        if (c = 'A') {
        numAs++;
        }
    }
printf ("%d A's in the input\n", numAs);
return 0;
}
```

Choices:
1 or none,
2,
3,
4,
5 or more

Example #1 (Solution):

- How many bugs in this code?

```
#include <stdio.h>
int main ( ) {
    int numAs; /* counts A's in the input */
    char c;
    numAs = 0;
    while ((c = getchar ( )) != EOF) {
/* getchar returns EOF if no more chars to read. */
        if (c == 'A') {
        numAs++;
        }
    }
printf ("%d A's in the input\n", numAs);
return 0;
}
```

3 Bugs

Bug Symptoms

- A value that's wildly out of range suggests an uninitialized variable or function return value.
- A loop that's executed 0 times or the maximum number of times when it shouldn't be suggests misuse of = in a test, or misparenthesization.

Example #2

- **What output is produced by this code?**

```
void addOne (int x) {
    x = x + 1;
}
int main ( ) {
    int y = 3;
    addOne (y);
    printf (%d, y);
    return 0;
}
```

Choices:
3,
4,
unknown,

Example #2 (Solution)

- **What output is produced by this code?**

```
void addOne (int x) {
    x = x + 1;
}
int main ( ) {
    int y = 3;
    addOne (y);
    printf (%d, y);
    return 0;
}
```

3 is Produced

passed by value: Make a copy of the original argument.

The original won't change

Example #3

- What choice for the blank ensures that 7 = 7 is printed?

```
#include <stdio.h>
int *f (int varNotToSet, int varToSet) {
 int n = 7;
 varToSet = 7;
 return _____ ;
  /* &n, &varNotToSet, or &varToSet could go here */
}
int main ( ) {
 int *ptr;
 int k1 = 7, k2 = 0;
 ptr = f(k1, k2);
 printf ("7 = ");
 printf ("%d\n", *ptr);
 return 0;
}
```

Choices:
`&n, &varNotToSet, &varToSet`
1
2,
3,
none

Example #3 (Solution)

Answer: none

- Variables and function parameters are allocated on the system stack
- When a function exits, its allocated space disappears.

Example #4

- What choice for the blank ensures that `values[0]=x`

```
int main ( ){
     int values[20];
     int x;
     ...
     assign ( _____ , x);
     ...
}

void assign ( int *y , int x) {
*y = x;
}
```

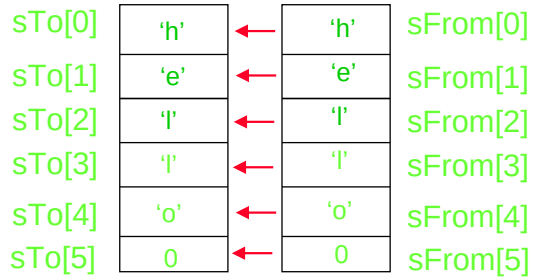
Answer:

1. **values**
2. **&values[0]**

Example #5

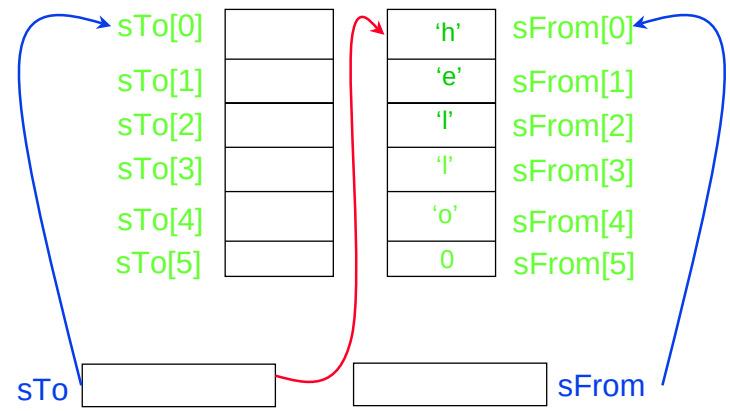
- How to copy one array to another array

```
int main ( ){
     char sFrom[6], sTo[6];
     copy (sTo[6], sFrom[6]);
 }
void copy (char sTo[], char sFrom[]) {
----
----
}
```



Example #5 (Soultion #1/4)

```
void copy (char sTo[], char sFrom[]) {
 sTo = Sfrom;
}
```

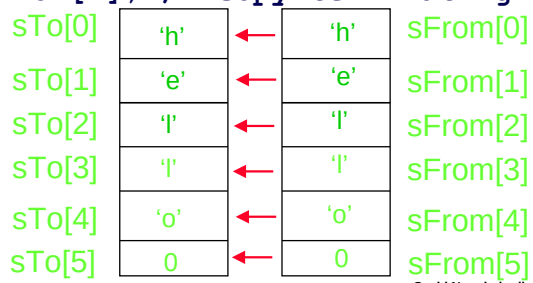


Similarly you don't compare two string using ==

Example #5 (Soultion #2/4)

°Straight Forward Array Version

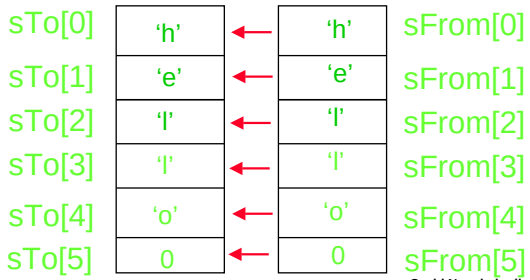
```
void copy (char sTo[], char sFrom[]) {
     int k = 0;
     while (sFrom[k] != '\0') {
          sTo[k] = sFrom[k];
          k++;
     }
     sTo[k] = sFrom[k]; /* copy terminating 0
*/
}
```



Example #5 (Soultion #3/4)

°Array Version (Taking advantage of value returned by assignment operator

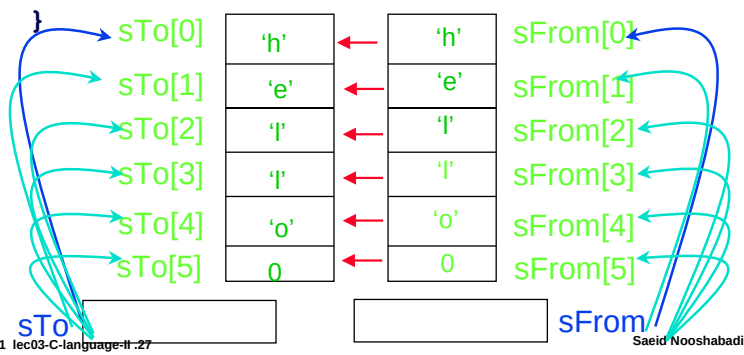
```
void copy (char sTo[], char sFrom[]) {
     int k = 0;
     while ((sTo[k] = sFrom[k]) != '\0'){
          k++;
     }
}
```



Example #5 (Soultion #3/4)

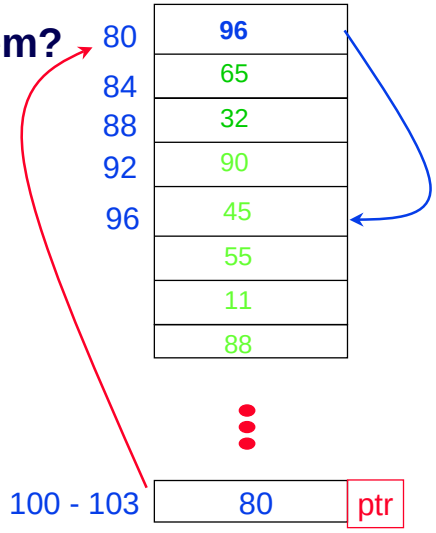
°Pointer Version

```
void copy (char sTo[], char sFrom[]) {
     while ((*sTo = *sFrom) != '\0') {
     /* pointer arithmetic (K&R 5.4) */
          sFrom++;
          sTo++;
     }
}
```



Pointers to pointers (**ptr)

°**Where do we use them?**

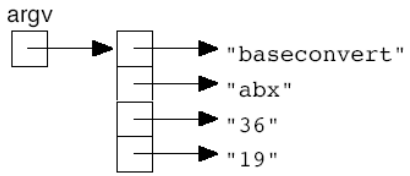


Recall: C Syntax; Arguments to `main`

- To get the main function to accept arguments, use this:

```
int main (int argc, char *argv[])
```

- What does this mean?
  - `argc` will contain the number of strings on the command line (the executable counts as one, plus one for each argument).
  - `argv` is a pointer to an array containing the rest of the arguments as strings



Things to Remember

- Common Pointer problems
  - Declare and write
  - Copy pointers v. values
- Bug Symptom
  - A value that's wildly out of range suggests an uninitialized variable or function return value.
  - A loop that's executed 0 times or the maximum number of times when it shouldn't be suggests misuse of = in a test, or misparenthesization.