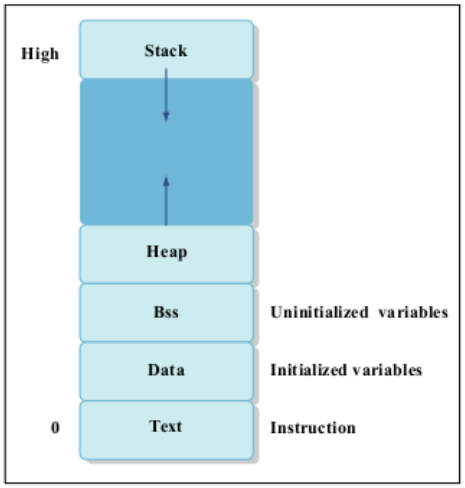
CS262 Lecture 04
Chapter 5 Pointers

Zoran Duric
Department of Computer Science

The Anatomy of C Memory

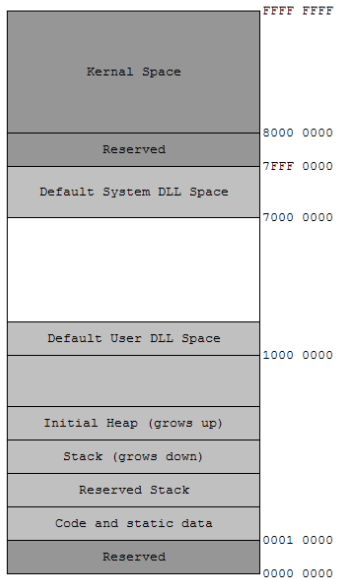
- application's address space
  - read only address
  - read/write address
  - aligned address
    - multi-byte types
  - physical address



- unix like systems

The Anatomy of C Memory

- application's address space
  - read only address
  - read/write address
  - aligned address
    - multi-byte types
  - physical address



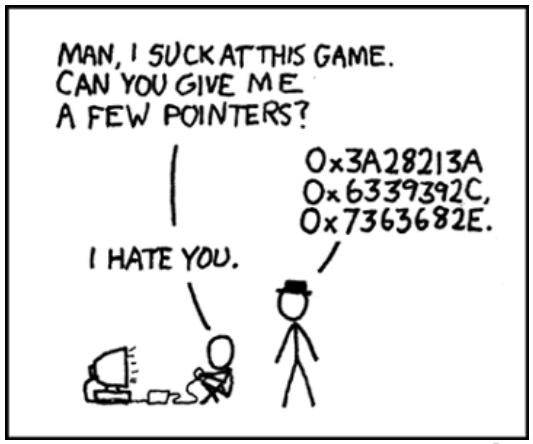
- win32 systems

Variables in Memory Map

```
// fixed address: visible to other files
int global_static;
// fixed address: only visible within file
static int file_static;
// parameters always stacked
int foo(int auto_param)
{
    // fixed address: only visible to function
    static int func_static;
    // stacked: only visible to function
    int auto_i, auto_a[10];
    // array explicitly allocated on heap
    double *auto_d = malloc(sizeof(double)*5);
    // return value in register or stacked
    return auto_i;
}
```

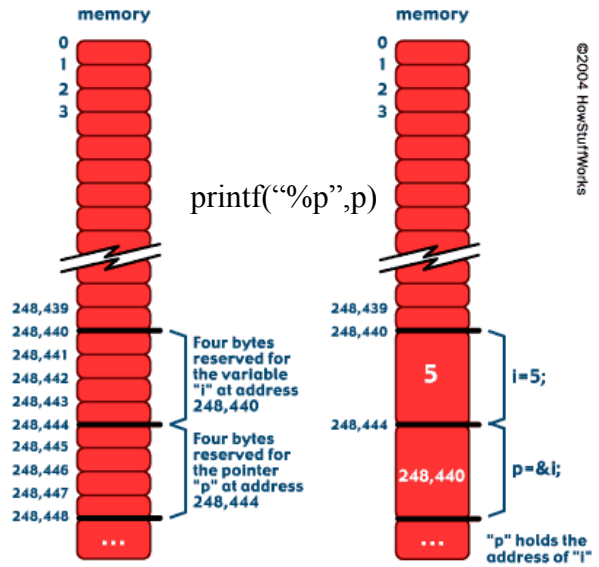
Pointer

- A pointer stores an **address** in application's address space
- read “pointer” as “an address pointing to”
- int i=5;
- int * p= **&** i;



Pointer

aligned memory address



Declare a Pointer

- examples
  - int *i, j; // i is a pointer but j is just int
  - int i, *j; // j is a pointer but i is not
  - int *i, *j; // both i and j are pointers
  - int ** i = &j;   // what is this?
  - void * x=i;

Dereferencing

- int i=5;
- int * p=&i;
- p=10; // this means p has address 10
- *p=10; // this changes the value at address p

Dereferencing

- see swap.c

const

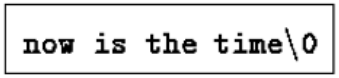
1. const int * p; //a pointer to const int
   - *p=0; //wrong
   - p=&i; p=&j; //OK
2. int const * p; //same as above
3. int * const p; //a const pointer to int
   - *p=0; //OK
   - p=&i; //wrong

Pointers and Array

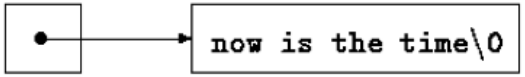
- char A[]=″GMU″; //A[0]='g' is allowed
- char * A=″GMU″; //ok, but A[0]='g' will crash
- char * p=A; **//array name is the pointer of its firs element**
- p=&A[0]; **//same as above**
- p=&A; **//same as above**

Pointers and Array

- char a_message[] = "now is the time"; /* an array */



- char * p_message = "now is the time"; /* a pointer*/



Multi-Dimensional Array

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

Pointers and Array

- /* strcpy: copy t to s; pointer version */
- void strcpy(char *s, char *t)
- {
  - while ((*s = *t) != '\0')
  - {
    - s++; t++;
  - }
- }

Pointers and Array

- void strcpy(char *s, char *t) { while(*s++ = (*t)++) ; }

Multi-Dimensional Array

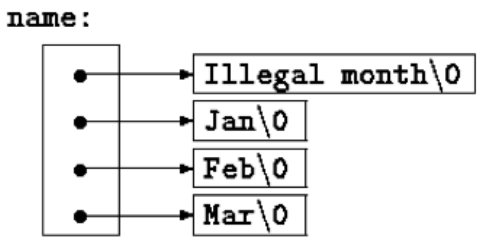
- char * B[]={“Hello”,”World”}; **//array of char ***
- char C[2][3]; **//array of char with 6 elements**
- (char *) * p=B; **//OK**
- char ** p=C; **//wrong**
- char * p=&C[0][0]; **//OK**
- char * p=C; **//same as above**

- void bar(char * foo[]){...}
- bar(B); **//OK**
- bar(C); **//wrong!, void bar(char foo[][3]) or bar(char *);**
- char * D[]=B; //wrong, char * D[]={“A”,”B”,”C”} or char ** D=B

Multi-Dimensional Array

- char * name [] = {“Illegal month”, “Jan”, “Feb”, Mar”};

name[0][1]=’c’;



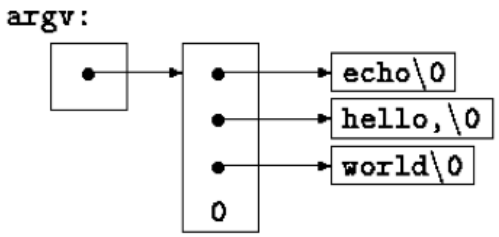
- char name[][15]= {“Illegal month”, “Jan”, “Feb”, Mar”};

aname:

Illegal month\0 Jan\0	Feb\0	Mar\0
0 15	30	45

Multi-Dimensional Array

- int void main(int argc, char ** argv)
  - >echo hello world (run command echo)



Multi-Dimensional Array

- Practice: write a program to do this:

Pointer and Array

- **There are differences between A (array) and p (pointer)**
  - you can't assign values to A but can do so to p
    - A=p; //wrong
  - sizeof(A) gives you the size of the entire array
  - sizeof(p) gives you the size of a pointer
  - p++ is allowed, but A++ is not

Pointer and Array

- char A[]=”GMU”;
- char * p=&A[1];
- printf(“%c”,p[-1]);
- what has been printed? G

Operation Cost

1. Integer arithmetic
2. Pointer access
3. Simple conditionals and loops
4. Static and automatic variable access
5. Array access
6. Floating-point with hardware support
7. Switch statements
8. Function calls
9. Floating-point emulation in software
10. Malloc() and free()
11. Library functions (sin, log, printf, etc.)
12. Operating system calls (open, brk, etc.)