

EECS 10: Computational Methods in Electrical and Computer Engineering

Lecture 22

Rainer Dömer

doemer@uci.edu

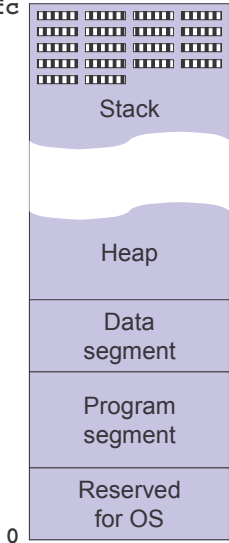
The Henry Samueli School of Engineering
Electrical Engineering and Computer Science
University of California, Irvine

Lecture 22: Overview

- Data Structures
 - Memory organization
 - Objects in memory
 - Pointers
 - Pointer definition
 - Pointer operators
 - Pointer dereferencing

Memory Organization

- Memory Segmentation
 - typical (virtual) memory layout on processor with 4-byte words and 1 GB of memory
 - Stack
 - grows and shrinks dynamically
 - function call hierarchy
 - stack frames with local variables
 - Heap
 - “free” storage
 - dynamic allocation by the user
 - Data segment
 - global (and static) variables
 - Program segment
 - stores binary program code
 - Reserved area for operating system



bfff fffc

Stack

Heap

Data segment

Program segment

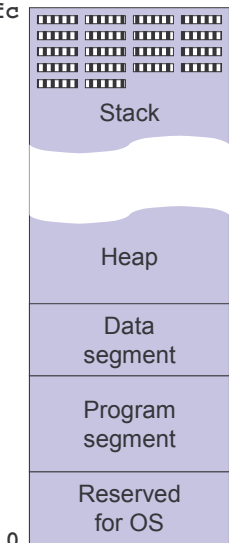
Reserved for OS

0

EECS10: Computational Methods in ECE, Lecture 22 (c) 2008 R. Doemer 3

Memory Organization

- Memory Segmentation
 - typical (virtual) memory layout on processor with 4-byte words and 1 GB of memory
- Memory errors
 - *Out of memory*
 - Stack and heap collide
 - *Segmentation fault*
 - access outside allocated segments
 - e.g. access to segment reserved for OS
 - *Bus error*
 - mis-aligned word access
 - e.g. word access to an address that is not divisible by 4



bfff fffc

Stack

Heap

Data segment

Program segment

Reserved for OS

0

EECS10: Computational Methods in ECE, Lecture 22 (c) 2008 R. Doemer 4

Objects in Memory

- Data in memory is organized as a set of objects
- Every object has ...
 - ... a *type* (e.g. `int`, `double`, `char[5]`)
 - type is known to the compiler at compile time
 - ... a *value* (e.g. `42`, `3.1415`, `"text"`)
 - value is used for computation of expressions
 - ... a *size* (number of bytes in the memory)
 - in C, the `sizeof` operator returns the size of a variable or type
 - ... a *location* (address in the memory)
 - in C, the “address-of” operator (`&`) returns the address of an object
- Variables ...
 - ... serve as identifiers for objects
 - ... are bound to objects
 - ... give objects a name

Objects in Memory

- Example: Variable values, addresses, and sizes

```
int x = 42;
int y = 13;
char s[] = "Hello World!";

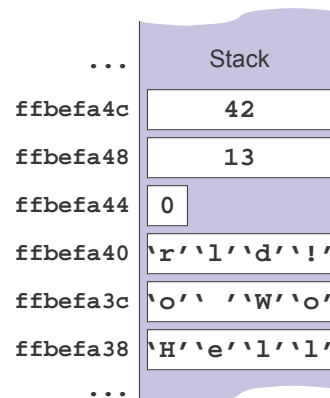
printf("Value   of x   is %d.\n", x);
printf("Address of x   is %p.\n", &x);
printf("Size    of x   is %u.\n", sizeof(x));
printf("Value   of y   is %d.\n", y);
printf("Address of y   is %p.\n", &y);
printf("Size    of y   is %u.\n", sizeof(y));
printf("Value   of s   is %s.\n", s);
printf("Address of s   is %p.\n", &s);
printf("Size    of s   is %u.\n", sizeof(s));
printf("Value   of s[1] is %c.\n", s[1]);
printf("Address of s[1] is %p.\n", &s[1]);
printf("Size    of s[1] is %u.\n", sizeof(s[1]));
```

Objects in Memory

- Example: Variable values, addresses, and sizes

```
int x = 42;
int y = 13;
char s[] = "Hello World!";
...
```

```
Value of x is 42.
Address of x is ffbe4c.
Size of x is 4.
Value of y is 13.
Address of y is ffbe48.
Size of y is 4.
Value of s is Hello World!.
Address of s is ffbe38.
Size of s is 13.
Value of s[1] is e.
Address of s[1] is ffbe39.
Size of s[1] is 1.
```



EECS10: Computational Methods in ECE, Lecture 22

(c) 2008 R. Doemer

7

Pointers

- *Pointers* are variables whose values are *addresses*
 - The “address-of” operator (&) returns a pointer!
- Pointer Definition
 - The unary * operator indicates a pointer type in a definition

```
int x = 42; /* regular integer variable */
int *p; /* pointer to an integer */
```

- Pointer initialization or assignment
 - A pointer may be set to the “address-of” another variable
 - A pointer may be set to 0 (points to no object)
 - A pointer may be set to NULL (points to “NULL” object)

```
p = &x; /* p points to x */
```

```
p = 0; /* p points to no object */
```

```
#include <stdio.h> /* defines NULL as 0 */
p = NULL; /* p points to no object */
```

EECS10: Computational Methods in ECE, Lecture 22

(c) 2008 R. Doemer

8

Pointers

- Pointer Dereferencing

- The unary * operator dereferences a pointer to the value it points to (“content-of” operator)

```
#include <stdio.h>
int x = 42; /* regular integer variable */
int *p = NULL; /* pointer to an integer */
```



Pointers

- Pointer Dereferencing

- The unary * operator dereferences a pointer to the value it points to (“content-of” operator)

```
#include <stdio.h>
int x = 42; /* regular integer variable */
int *p = NULL; /* pointer to an integer */
p = &x; /* make p point to x */
```



Pointers

- Pointer Dereferencing
 - The unary * operator dereferences a pointer to the value it points to (“content-of” operator)

```
#include <stdio.h>
int x = 42; /* regular integer variable */
int *p = NULL; /* pointer to an integer */

p = &x; /* make p point to x */
printf("x is %d, content of p is %d\n", x, *p);
```

```
x is 42, content of p is 42
```



Pointers

- Pointer Dereferencing
 - The unary * operator dereferences a pointer to the value it points to (“content-of” operator)

```
#include <stdio.h>
int x = 42; /* regular integer variable */
int *p = NULL; /* pointer to an integer */

p = &x; /* make p point to x */
printf("x is %d, content of p is %d\n", x, *p);
*p = 2 * *p; /* multiply content of p by 2 */
printf("x is %d, content of p is %d\n", x, *p);
```

```
x is 42, content of p is 42
x is 84, content of p is 84
```



Pointers

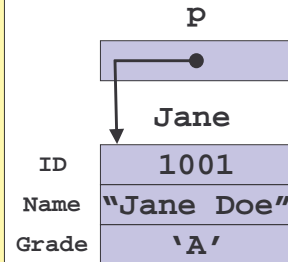
- Pointer Dereferencing
 - The `->` operator dereferences a pointer to a structure to the content of a structure member

```
struct Student
{
    int ID;
    char Name[40];
    char Grade;
};

struct Student Jane =
{1001, "Jane Doe", 'A'};

struct Student *p = &Jane;

void PrintStudent(void)
{
    printf("ID:    %d\n", p->ID);
    printf("Name:  %s\n", p->Name);
    printf("Grade: %c\n", p->Grade);
}
```



```
ID:    1001
Name:  Jane Doe
Grade: A
```