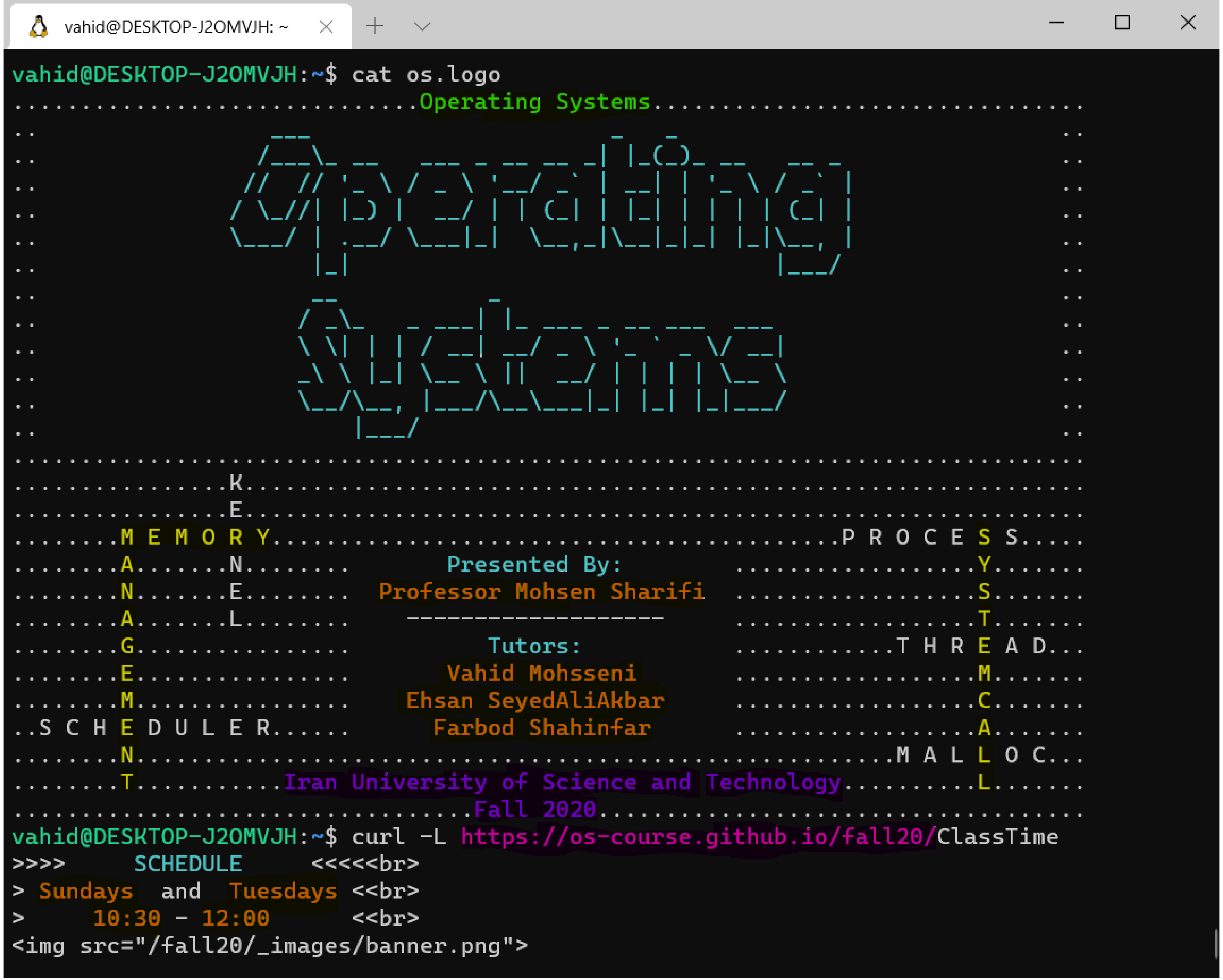
دانشگاه علم و صنعت ایران

Operating Systems

Advanced Topics in C Programming Language

Fall 2020



```
vahid@DESKTOP-J2OMVJH:~$ cat os.logo
..............................Operating Systems................................
..                 ___                      _   _                            ..
..                /___\_ __   ___ _ __ __ _| |_(_)_ __   __ _                ..
..               //  // '_ \ / _ \ '__/ _` | __| | '_ \ / _` |               ..
..              / \_//| |_) |  __/ | | (_| | |_| | | | | (_| |               ..
..              \___/ | .__/ \___|_|  \__,_|\__|_|_| |_|\__, |               ..
..                    |_|                               |___/                ..
..                    __           _                                         ..
..                   / _\_   _ ___| |_ ___ _ __ ___  ___                     ..
..                   \ \| | | / __| __/ _ \ '_ ` _ \/ __|                    ..
..                   _\ \ |_| \__ \ ||  __/ | | | | \__ \                    ..
..                   \__/\__, |___/\__\___|_| |_| |_|___/                    ..
..                       |___/                                               ..
...............................................................................
................K..............................................................
................E..............................................................
........M E M O R Y..........................................P R O C E S S.....
........A.......N........       Presented By:        ..................Y.......
........N.......E........  Professor Mohsen Sharifi  ..................S.......
........A.......L........    -------------------     ..................T.......
........G................          Tutors:           ............T H R E A D...
........E................       Vahid Mohsseni       ..................M.......
........M................    Ehsan SeyedAliAkbar     ..................C.......
..S C H E D U L E R......      Farbod Shahinfar      ..................A.......
........N........................................................M A L L O C...
........T...........Iran University of Science and Technology..........L.......
..................................Fall 2020....................................
vahid@DESKTOP-J2OMVJH:~$ curl -L https://os-course.github.io/fall20/ClassTime
>>>>     SCHEDULE     <<<<<br>
> Sundays  and  Tuesdays <<br>
>     10:30 - 12:00      <<br>
<img src="/fall20/_images/banner.png">
```

Agenda

- Functions
- Struct and Typedef
- Pointers
- Memory Allocation
- String Processing
- Pointer to Functions
- Header Files
- XV6 Shell

Functions

Functions

```
#include <stdio.h>
#include <stdbool.h>
bool is_even(int value){
    return value % 2 == 0;
}

int main(int argc, char *argv[])
{
   int val;
   scanf(“%d\n”, &val);
   if (is_even(val)) {
       printf(“it is even\n”);
   } else {
       printf(“it is odd\n”);
   }
     return 0;
}
```

Functions

```
#include <stdio.h>
void divide_by_2(int arr[], int size){
    // pass by reference
    for (int I = 0; I < size; I++) {
        arr[I] = arr[I] / 2;
    }
}

int main(int argc, char *argv[])
{
   int val;
   scanf(“%d\n”, &val);
   if (is_even(val)) {
       printf(“it is even\n”);
   } else {
       printf(“it is odd\n”);
   }
     return 0;
}
```

Functions: Trace a function call

```
#include <stdio.h>

void func(int a){
  int b = 10;
  return a / b;
}

int main(int argc, char *argv[])
{
   int val;
   int c;
   scanf(“%d\n”, &val);
   c = func(val);
   printf(“%d\n” c);
   return 0;
}
```

Address	Value
0x00A1	….
0x00A2	….
0x00A3	….
0x00A4	….
0x00A5	….
0x00A6	….
0x00A7	….
0x00A8	…..
0x00A9	….

Functions: Trace a function call

```
#include <stdio.h>

void func(int a){
  int b = 10;
  return a / b;
}

int main(int argc, char *argv[])
{
   int val;
   int c;
   scanf(“%d\n”, &val);
   c = func(val);
   printf(“%d\n” c);
   return 0;
}
```

int val

int c

Address	Value
0x00A1	….
0x00A2	….
0x00A3	….
0x00A4	….
0x00A5	….
0x00A6	….
0x00A7	….
0x00A8	…..
0x00A9	….

Functions: Trace a function call

```c
#include <stdio.h>

void func(int a){
  int b = 10;
  return a / b;
}

int main(int argc, char *argv[])
{
   int val;
   int c;
   scanf(“%d\n”, &val);
   c = func(val);
   printf(“%d\n” c);
   return 0;
}
```

int val

int c

Address	Value
0x00A1	….
0x00A2	….
0x00A3	….
0x00A4	….
0x00A5	….
0x00A6	….
0x00A7	….
0x00A8	…..
0x00A9	….

Functions: Trace a function call

```
#include <stdio.h>

void func(int a){
  int b = 10;
  return a / b;
}

int main(int argc, char *argv[])
{
   int val;
   int c;
   scanf(“%d\n”, &val);
   c = func(val);
   printf(“%d\n” c);
   return 0;
}
```

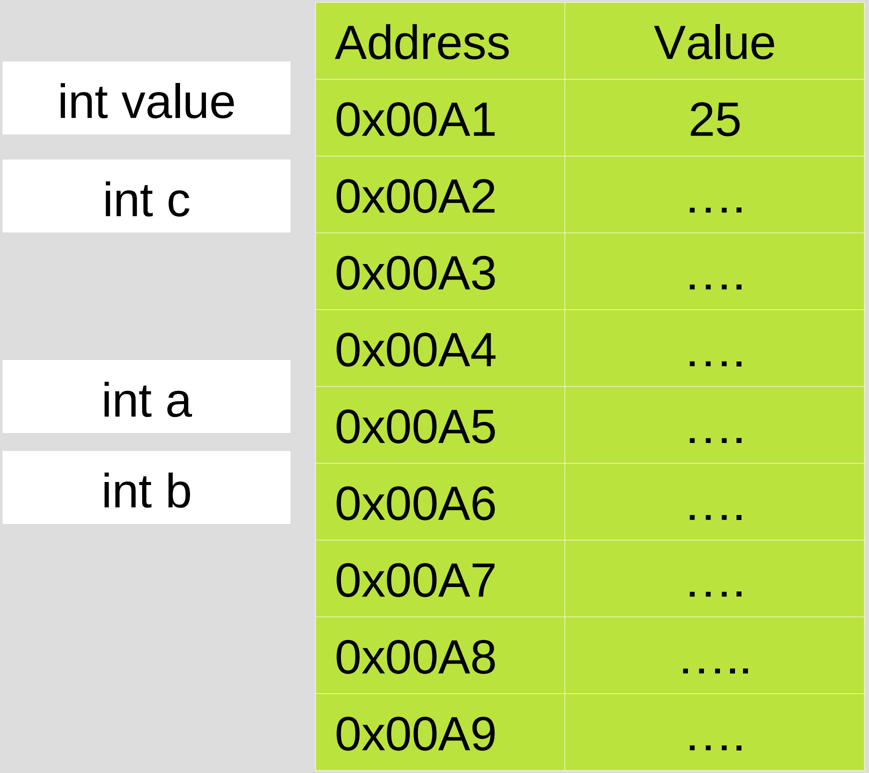
int value

int c

Address	Value
0x00A1	25
0x00A2	….
0x00A3	….
0x00A4	….
0x00A5	….
0x00A6	….
0x00A7	….
0x00A8	…..
0x00A9	….

Functions: Trace a function call

```
#include <stdio.h>

void func(int a){
  int b = 10;
  return a / b;
}

int main(int argc, char *argv[])
{
   int val;
   int c;
   scanf(“%d\n”, &val);
   c = func(val);
   printf(“%d\n” c);
   return 0;
}
```



Address	Value
0x00A1	25
0x00A2	….
0x00A3	….
0x00A4	….
0x00A5	….
0x00A6	….
0x00A7	….
0x00A8	…..
0x00A9	….

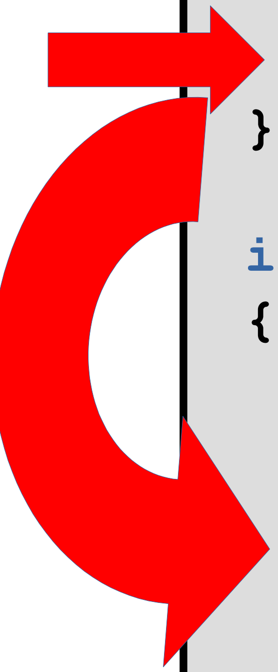
Functions: Trace a function call

```
#include <stdio.h>

void func(int a){
  int b = 10;
  return a / b;
}

int main(int argc, char *argv[])
{
   int val;
   int c;
   scanf(“%d\n”, &val);
   c = func(val);
   printf(“%d\n” c);
   return 0;
}
```

	Address	Value
int value	0x00A1	25
int c	0x00A2	….
	0x00A3	….
	0x00A4	….
int a	0x00A5	25
int b	0x00A6	10
	0x00A7	….
	0x00A8	…..
	0x00A9	….

Return register	2

Functions: Trace a function call



```
#include <stdio.h>

void func(int a){
  int b = 10;
  return a / b;
}

int main(int argc, char *argv[])
{
   int val;
   int c;
   scanf(“%d\n”, &val);
   c = func(val);
   printf(“%d\n” c);
   return 0;
}
```

	Address	Value
int value	0x00A1	25
int c	0x00A2	….
	0x00A3	….
	0x00A4	….
int a	0x00A5	25
int b	0x00A6	10
	0x00A7	….
	0x00A8	…..
	0x00A9	….

Return register	2

Functions: Trace a function call

```
#include <stdio.h>

void func(int a){
  int b = 10;
  return a / b;
}

int main(int argc, char *argv[])
{
   int val;
   int c;
   scanf(“%d\n”, &val);
   c = func(val);
   printf(“%d\n” c);
   return 0;
}
```

	Address	Value
int value	0x00A1	25
int c	0x00A2	2
	0x00A3	….
	0x00A4	….
	0x00A5	25
	0x00A6	10
	0x00A7	….
	0x00A8	…..
	0x00A9	….

Return register	2

Struct and Typedef

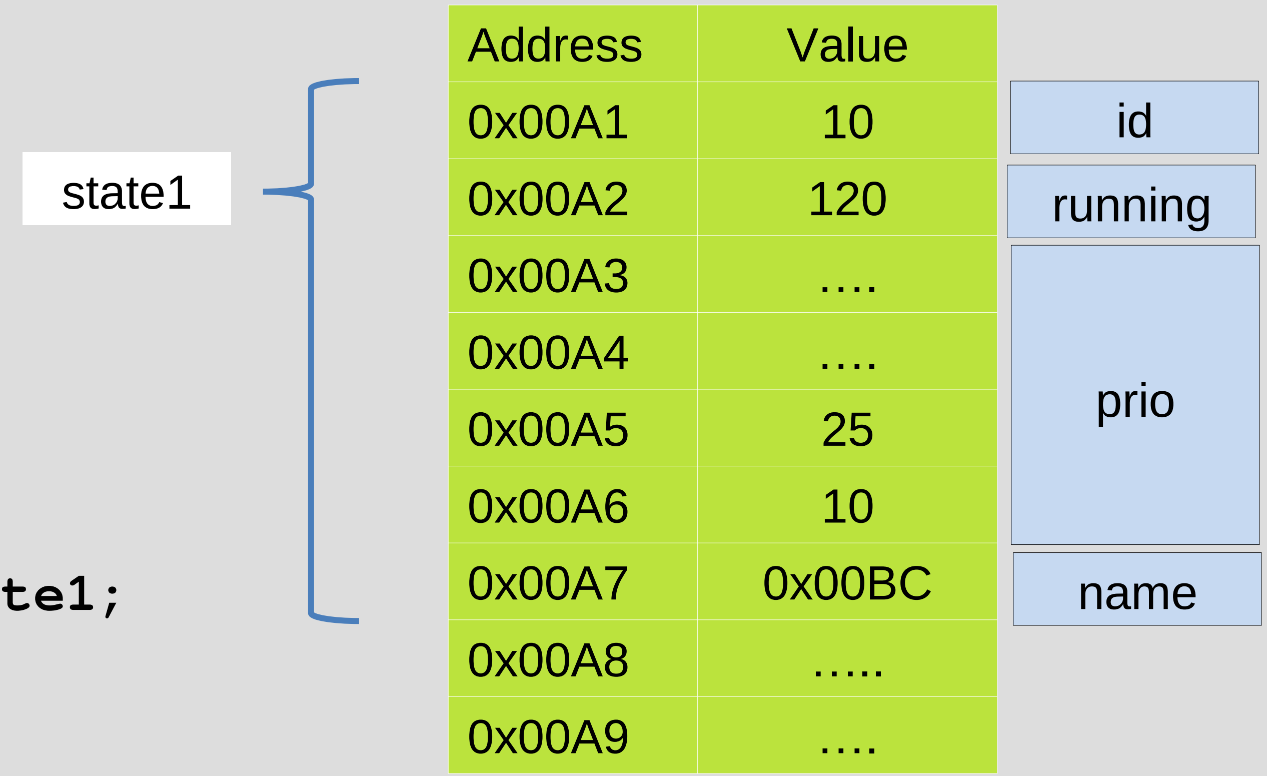
Struct and Typedef

```
#include <stdio.h>

struct point {
    Int x;
    Int y;
};

typedef struct point point_t;

void print_point(struct point p) {
    printf("(%d, %d)\n", p.x, p.y);
}

int main(int argc, char *argv[])
{
     point_t p1 = {.x=5, .y=2};
   print_point(p1);
     return 0;
}
```

- You can define a structure to store values in a certain way.
- You can define a name for the struct.
- This is may be good for code readability and creating abstractions.

Struct and Typedef

```
struct obj_state {
    uint8_t id;
    uint8_t running;
    float prio;
    char *name[10];
};

int main()
{
    struct obj_state state1;
    return 0;
}
```

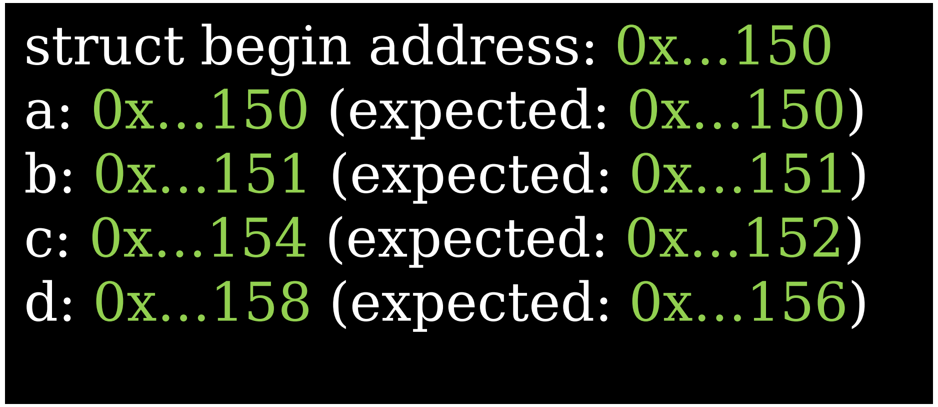


state1

Address	Value
0x00A1	10
0x00A2	120
0x00A3	….
0x00A4	….
0x00A5	25
0x00A6	10
0x00A7	0x00BC
0x00A8	…..
0x00A9	….

Struct and Typedef

- Fields of a struct may not be contiguous because compiler may add padding for performance purposes.



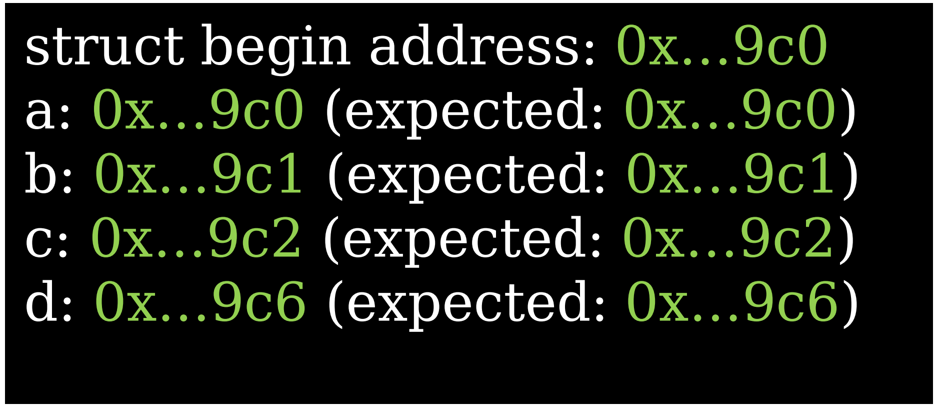
```
struct begin address: 0x…150
a: 0x…150 (expected: 0x…150)
b: 0x…151 (expected: 0x…151)
c: 0x…154 (expected: 0x…152)
d: 0x…158 (expected: 0x…156)
```

```
struct {
    char a,
    char b,
    int c,
    char d
};
```

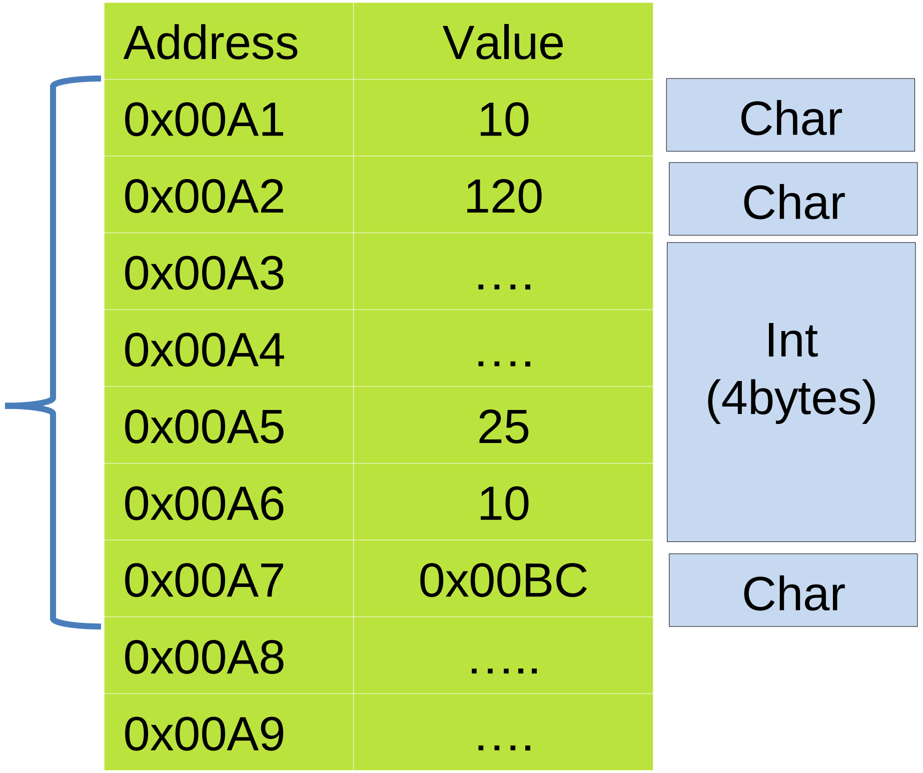
Address	Value	
0x00A1	10	Char
0x00A2	120	Char
0x00A3	….	Padding
0x00A4	….	
0x00A5	25	Int (4bytes)
0x00A6	10	
0x00A7	0x00BC	
0x00A8	…..	
0x00A9	….	Char

Struct and Typedef

- It is possible to give instructions to compiler not to add padding
  - For GCC __attribute__((__packed__))



```
struct begin address: 0x…9c0
a: 0x…9c0 (expected: 0x…9c0)
b: 0x…9c1 (expected: 0x…9c1)
c: 0x…9c2 (expected: 0x…9c2)
d: 0x…9c6 (expected: 0x…9c6)
```

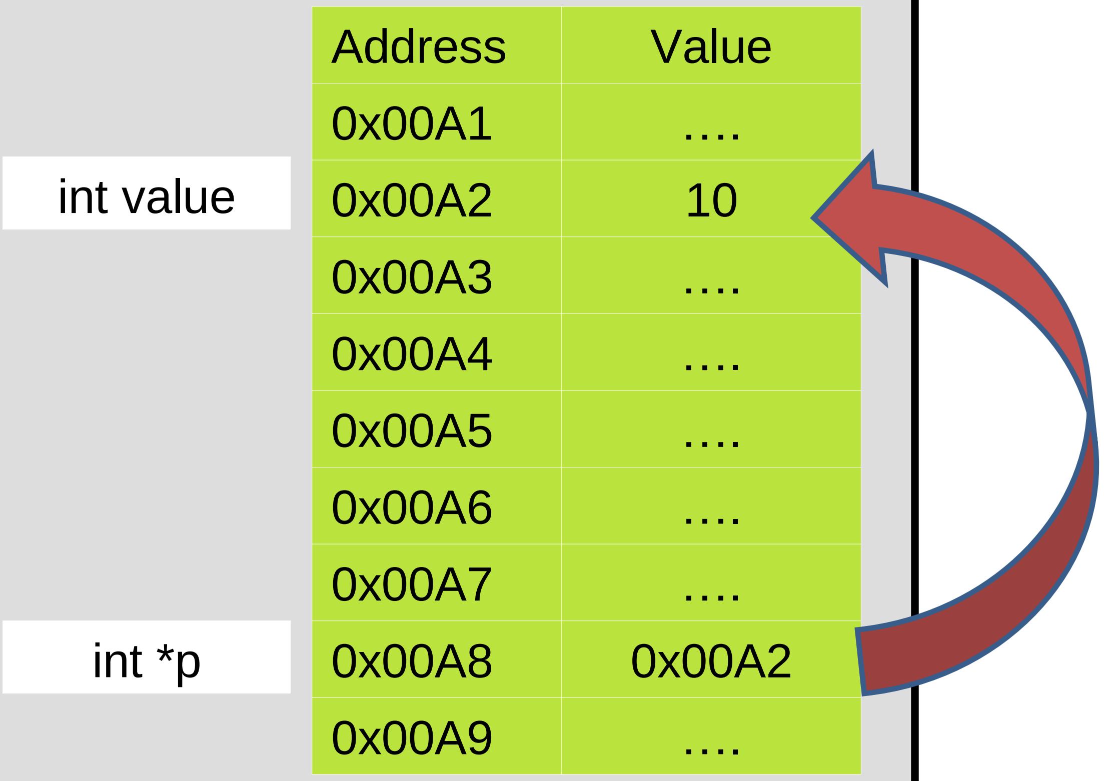


Address	Value
0x00A1	10
0x00A2	120
0x00A3	….
0x00A4	….
0x00A5	25
0x00A6	10
0x00A7	0x00BC
0x00A8	…..
0x00A9	….

Pointers

Pointers

```
#include <stdio.h>

int main(int argc, char *argv[])
{
   int value = 10;
   int *p;
   p = &value;

   printf(“value is: %d, “
          ”(address: %x)\n”,
           *p, p);
   return 0;
}
```



	Address	Value
	0x00A1	….
int value	0x00A2	10
	0x00A3	….
	0x00A4	….
	0x00A5	….
	0x00A6	….
	0x00A7	….
int *p	0x00A8	0x00A2
	0x00A9	….

Pointers

```
#include <stdio.h>

struct rectangle { int width; int height; point_t top_left;};

void print_rect(struct rectangle *p) {
    printf(“<w: %d, h: %d, x: %d, y: %d>\n”,
           p->width, p->height, p->top_left.x,
           p->top_left.y);
}

int main(int argc, char *argv[]) {
   point_t p1 = {.x=2, .y=-3};
   struct rectangle r1 = {.width=10, .height=5, top_left=p1};

   print_rect(&r1);
   return 0;
}
```

- Pass the address of the structure to the function.
- Reduces memory copy.

Pointers

```
#include <stdio.h>

struct rectangle { int width; int height; point_t top_left;};

void print_rect(struct rectangle *p) {
    printf(“<w: %d, h: %d, x: %d, y: %d>\n”,
           p->width, p->height, p->top_left.x,
           p->top_left.y);
}

int main(int argc, char *argv[]) {
   point_t p1 = {.x=2, .y=-3};
   struct rectangle r1 = {.width=10, .height=5,

   print_rect(&r1);
   return 0;
}
```

- p->top_left.x
- (*p).top_left.x
- get the struct from address pointed to by `p` and select `top_left` member of the struct.

Pointers

Both instructions below are equivalent:

- p→width = 10;
- (*p).width = 10;

Pointers: Arithmetic

- When incrementing a pointer the address is changed with respect to the size of data type of the pointer.

```
{
   int64_t val = 10;
   int64_t *p64 = &val;
   printf(“p64:%x,%x\n”, p64, p64+1);

   int8_t *p8 = (int8_t *)(&val);
   printf(“p8: %x, %x\n”, p8, p8+1);
   return 0;
}
```

- P8 moved 1 byte
- P64 moved 8 bytes

```
p64, p64+1: 4a7d3c60, 4a7d3c68
p8,   p8+1:   4a7d3c60, 4a7d3c61
```

Pointers: sizeof()

- Size of an pointer is the address size:
  - On a 32 bit system sizeof(*p) == 4
  - On a 64 bit system sizeof(*p) == 8

Memory Allocation

Memory Allocation

- Local variables are allocated from stack memory.
  - Local variables are freed when they are out of scope (for example function return)
- Allocating memory with `malloc` or `calloc` uses heap memory.
  - Memory should be explicitly freed using `free` function.

Memory Allocation

```
#include <stdio.h>
struct rectangle { int width; int height; point_t top_left;};

struct rectangle new_rect(int w, int h) {
    struct rectangle rect;
    rect.width = w;
    rect.hight = h;
    return rect;
}

int main(int argc, char *argv[])
{
   struct rectangle rect = new_rect(10, 5);
   // do some processing
   free(rect);
     return 0;
}
```

Danger:
On return the rect data structure is copied.

Memory Allocation

```
#include <stdio.h>

struct rectangle { int width; int height; point_t top_left;};

struct rectangle *new_rect(int w, int h) {
    struct rectangle rect;
    rect.width = w;
    rect.hight = h;
    return &rect;
}

int main(int argc, char *argv[])
{
   struct rectangle *rect = new_rect(10, 5);
   // do some processing
   free(rect);
     return 0;
}
```

Danger:
On return the context of the function is destroyed and, the returned pointer is invalid

Memory Allocation

```
#include <stdio.h>

struct rectangle { int width; int height; point_t top_left;};

struct rectangle *new_rect(int w, int h) {
    struct rectangle *rect = \
       malloc(sizeof(struct rectangle *));
    rect→width = w;
    rect→hight = h;
    return rect;
}

int main(int argc, char *argv[])
{
   struct rectangle *rect = new_rect(10, 5);
   // do some processing
   free(rect);
     return 0;
}
```

Allocate memory from heap

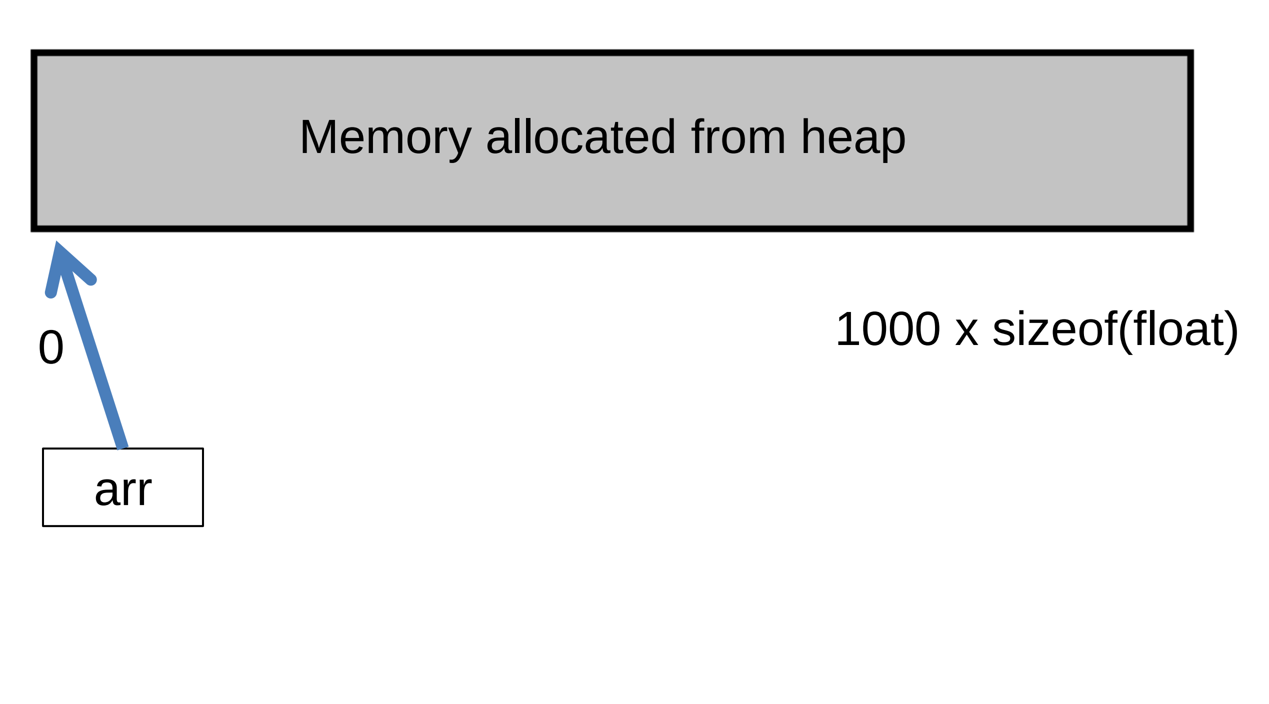
Pointers Revisited

Pointers Revisited: Pointer to Pointer

```
void new_rect(struct rectangle **p) {
   struct rectangle *r;
   r = malloc(sizeof(struct rectangle));

   *r = (struct rec..) {
          .width = 1,
          .height = 2,
          .top_left = (point_t) {.x=3, .y=4},
   };
   *p = r;
}

int main(int argc, char *argv[]) {
   struct rectangle *r1 = NULL;
   new_rect(&r1);

   print_rect(&r1);
   return 0;
}
```

Pointers Revisited: Allocate array from heap

```
int main(int argc, char *argv[]) {
    float *arr;
    arr = malloc( 1000 * sizeof(*arr));

    for (int i = 0; i < 1000; i++)
        arr[i] = 3.14;
    return 0;
}
```



Memory allocated from heap

1000 x sizeof(float)

0

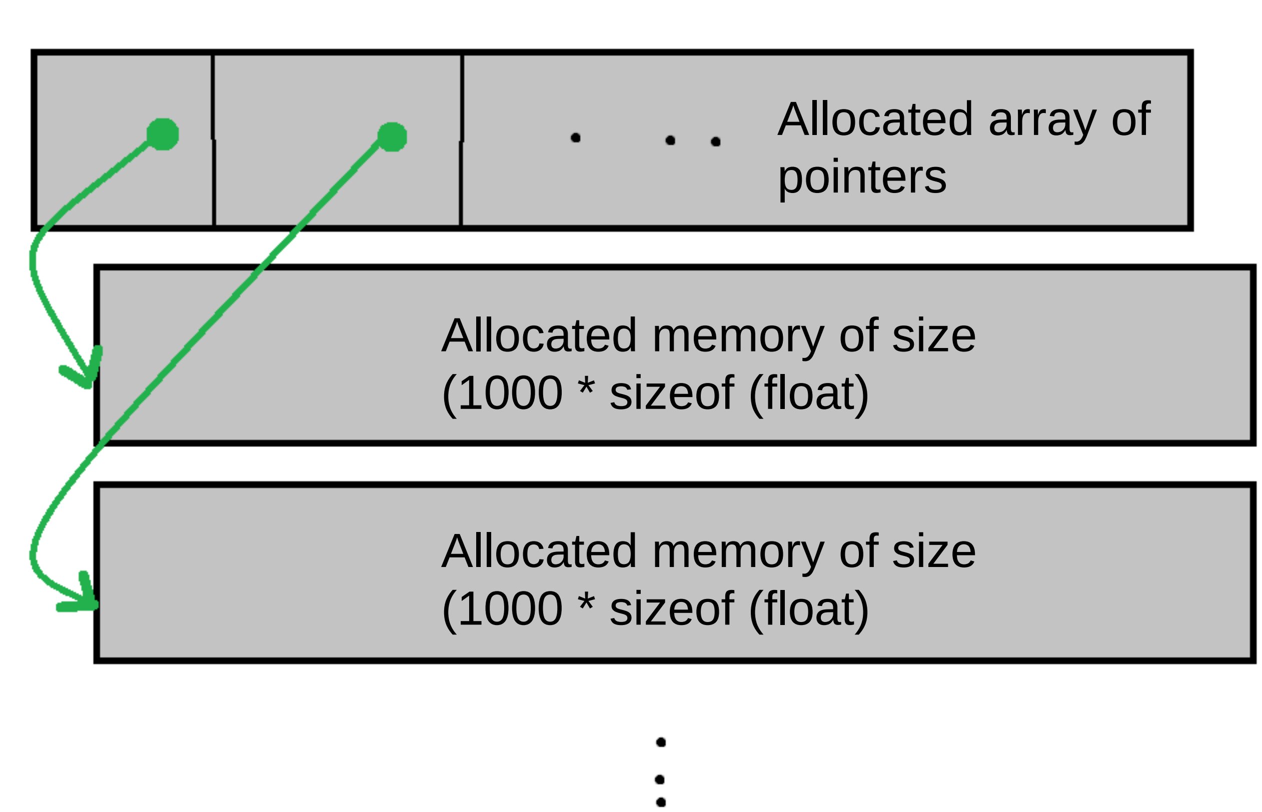
arr

Pointers Revisited: Allocate 2d arrays

```
int main(int argc, char *argv[]) {
    // mat [50][1000]
    float **arr;
    arr = malloc ( 50 * sizeof(float *));

    for (int i = 0; i < 50; i++)
        arr[i] = malloc( 1000 * sizeof(float));

    for (int i = 0; i < 50; i++)
        for (int j = 0; j < 1000; j++)
            arr[i][j] = 3.14;

    return 0;
}
```

Pointers Revisited: Allocate 2d arrays

```
int main(int argc, char *argv[]) {
  // mat [50][1000]
  float **arr;
  arr = malloc ( 50 * sizeof(float *));

  for (int i = 0; i < 50; i++)
    arr[i] = malloc( 1000 *
                sizeof(float));
```



Allocated array of pointers

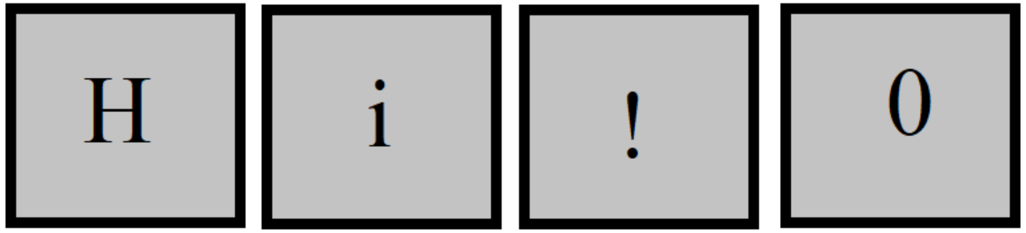
Allocated memory of size (1000 * sizeof (float)

Allocated memory of size (1000 * sizeof (float)

String Processing

String Processing

- Strings are an array of characters
  - char str[100];
  - char *str = malloc(….);
- The end of string is usually determined by ‘\0’
  - It is called null-terminated string



H	i	!	0

String Processing

- \n
- \r
- \t
- \0

String Processing

- Header file <string.h>
- size_t strlen(const char *s);
- size_t strnlen(const char *s, size_t maxlen);

String Processing

- Header file <string.h>
- char *strcpy(char *dest, const char *src);
- char *strncpy(char *dest, const char *src, size_t n);

String Processing

- Header file <stdlib.h>
- int atoi(const char *nptr);
- long atol(const char *nptr);
- long long atoll(const char *nptr);

String Processing

- Header file <stdio.h>
- int scanf(const char *format, ...);
- int sscanf(const char *str, const char *format, ...);

String Processing

- Header file <stdio.h>
- int printf(const char *format, ...);
- int sprintf(char *str, const char *format, ...);
- int snprintf(char *str, size_t size, const char *format, ...);

String Processing

- %d: integer
- %ld: long
- %s: string
- %x: hex
- %p: pointer

Pointer to Function

Pointer to Function

- To define a variable having type of pointer to a function:
  - <function return type> (*<variable name>)(<list of input parameters>)
  - int (*count_even)(int arr[], int count)
- typedef can be used to define a type and create abstraction

Pointer to Function

```
typedef int(*on_btn_clk_t)(struct event*);

int my_func(struct *event) {
  // …
  return 0;
}

int main(void)
{
   on_btn_clk_t _func = &my_func;
   // …
   if (condition) {
     _func(ev);
   }
   exit();
}
```

XV6 Shell

XV6 Shell

- XV6 is a UNIX like operating system implemented for educational purposes by MIT students.
- Last session we examined how an operating system boots. In this section we assume that operating system has been booted and the kernel is ready. We focus on the shell program letting users to interact with the system.

XV6 Shell

```
int
main(void)
{
    // ….
    exit();
}
```

- By convention starts from main function.

XV6 Shell

```
int
main(void)
{
  static char buf[100];
  int fd;
  // Ensure that three file descriptors are open.
  while((fd = open("console", O_RDWR)) >= 0){
    if(fd >= 3){
      close(fd);
      break;
    }
  }
  // ….
  exit();
}
```

- Make sure at least three file descriptors are open
- 0: stdin
- 1: stdout
- 2: stderr

XV6 Shell

```
int
main(void)
{
  static char buf[100];
  int fd;
  // ….
  while(getcmd(buf, sizeof(buf)) >= 0){
    if(buf[0] == 'c' && buf[1] == 'd' && buf[2] == ' '){
      // Chdir must be called by the parent, not the child.
      buf[strlen(buf)-1] = 0;  // chop \n
      if(chdir(buf+3) < 0)
        printf(2, "cannot cd %s\n", buf+3);
      continue;
    }
    if(fork1() == 0)
      runcmd(parsecmd(buf));
    wait();
  }
  exit();
}
```

- Read a command and execute…

XV6 Shell

```
int
getcmd(char *buf, int nbuf)
{
  printf(2, "$ ");
  memset(buf, 0, nbuf);
  gets(buf, nbuf);
  if(buf[0] == 0) // EOF
    return -1;
  return 0;
}
```

Questions?

?