CHALMERS
UNIVERSITY OF TECHNOLOGY

Machine-Oriented Programming

C-Programming part 3

Pedro Trancoso
ppedro@chalmers.se

Original slides by Ulf Assarsson

Objectives

- Pointers to struct
- Port addressing with structs
- Function pointer
- Structs with function pointers
- Dynamic memory allocation
- Linked lists
- File operations

Pointers to struct / arrow notation

```
Course mop;
Course *pmop; // Pointer to struct

pmop = &mop;

(*pmop).name = …

// Or easier:

pmop->name   = …
```

The arrow notation simplifies the code, as it is common to have pointers for structs

Pointer to struct

```c
#include <stdio.h>
#include <stdlib.h>

char* coursename = "Machine Oriented Programming";

typedef struct {
      char* name;
      float credits;
      int   numberOfParticipants;
} Course;

int main() {
      Course *pmop; // Pointer to struct
      pmop = (Course*)malloc(sizeof(Course));

      //(*pmop).name = coursename;
      pmop->name   = coursename;
      pmop->credits = 7.5;
      pmop->numberOfParticipants = 110;

      free(pmop);
      return 0;
}
```

Access to members (fields) via -> operator

```c
// or
Course mop, *pmop;
pmop = &mop;
// and of course without free()
```

In Java:

```
public class Course {
  String name;
  float   credits;
  int     numberOfParticipants;
}

Course mop = new Course();
mop.name = …
mop.credits = 7.5;
…
```

Port addressing with structs

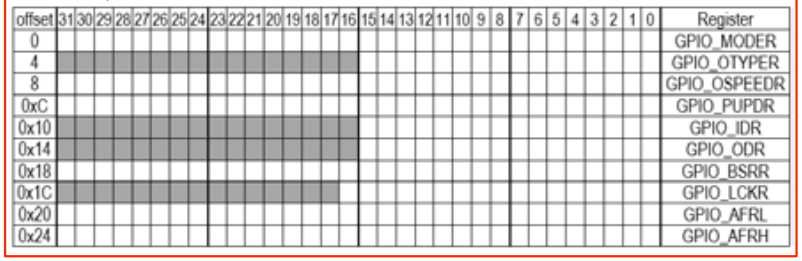
Instead of:

```
#define portModer   ((volatile unsigned int *)  (PORT_DISPLAY_BASE))
#define portOtyper  ((volatile unsigned short *)(PORT_DISPLAY_BASE+0x4))
#define portOspeedr ((volatile unsigned int *)  (PORT_DISPLAY_BASE+0x8))
#define portPupdr   ((volatile unsigned int *)  (PORT_DISPLAY_BASE+0xC))
#define portIdrLow  ((volatile unsigned char *) (PORT_DISPLAY_BASE+0x10))
#define portIdrHigh ((volatile unsigned char *) (PORT_DISPLAY_BASE+0x11))
#define portOdrLow  ((volatile unsigned char *) (PORT_DISPLAY_BASE+0x14))
#define portOdrHigh ((volatile unsigned char *) (PORT_DISPLAY_BASE+0x14+1))
```

Then we can use structs by writing :

```
typedef struct {
    uint32_t    moder;
    uint16_t    otyper;         // +0x4
    uint16_t    otReserved; //
    uint32_t    ospeedr;        // +0x8
    uint32_t    pupdr;          // +0xc (12)
    uint8_t     idrLow;         // +0x10
    uint8_t     idrHigh;        // +0x11
    uint16_t    idrReserved;
    uint8_t     odrLow;         // +0x14
    uint8_t     odrHigh;        // +0x15
    uint16_t    odrReserved;
} GPIO;
```

GPIO
General Purpose Input Output
GPIO A: 0x40020000
GPIO B: 0x40020400
GPIO C: 0x40020800
GPIO D: 0x40020C00
GPIO E; 0x40021000




```
#define GPIO_D (*((volatile GPIO*) 0x40020c00))
#define GPIO_E (*((volatile GPIO*) 0x40021000))

Example:
GPIO_E.moder    =  0x55555555;
GPIO_E.otyper   =  0x00000000;
GPIO_E.ospeedr  =  0x55555555;
GPIO_E.pupdr    &= 0x55550000;
```

Port addressing – individual bytes

We defined idrLow and idrHigh as bytes and idrReserved as 16-bit. But we could instead have defined all these three as just

uint32_t idr; i.e. 4 bytes and then address individual bytes

```
uint8_t  x = *(uint8_t*)&GPIO_E.idr;          // idrLow
uint8_t  y = *((uint8_t*)&GPIO_E.idr  + 1);  // idrHigh
uint16_t z = *((uint16_t*)&GPIO_E.idr + 1);  // idrReserved
```

```
typedef struct {
    uint32_t    moder;
    uint16_t    otyper;          // +0x4
    uint16_t    otReserved;
    uint32_t    ospeedr;         // +0x8
    uint32_t    pupdr;           // +0xc
    uint8_t     idrLow;          // +0x10
    uint8_t     idrHigh;         // +0x11
    uint16_t    idrReserved;
    uint8_t     odrLow;          // +0x14
    uint8_t     odrHigh;         // +0x15
    uint16_t    odrReserved;
} GPIO;
typedef volatile GPIO* gpioptr;
#define GPIO_E (*((gpioptr) 0x40021000))
```

```
typedef struct _gpio {
    uint32_t    moder;
    uint32_t    otyper;          // +0x4
    uint32_t    ospeedr;         // +0x8
    uint32_t    pupdr;           // +0xc
    uint32_t    idr;             // +0x10
    uint32_t    odr;             // +0x14
} GPIO;
typedef volatile GPIO* gpioptr;
#define GPIO_E (*((gpioptr) 0x40021000))
```

Port addressing with structs

```
typedef struct tag_usart
{
    volatile unsigned short sr;
    volatile unsigned short Unused0;
    volatile unsigned short dr;
    volatile unsigned short Unused1;
    volatile unsigned short brr;
    volatile unsigned short Unused2;
    volatile unsigned short cr1;
    volatile unsigned short Unused3;
    volatile unsigned short cr2;
    volatile unsigned short Unused4;
    volatile unsigned short cr3;
    volatile unsigned short Unused5;
    volatile unsigned short gtpr;
} USART;
#define USART1 ((USART *) 0x40011000)
```

Example:

```
while (( (*USART).sr & 0x80) == 0)
    ; // wait until it is ok to write
(*USART1).dr = (unsigned short) 'a';
```

Or with the arrow notation:

```
while (( USART->sr & 0x80)==0)
    ;
USART1->dr = (unsigned short) 'a';
```

USART

Universal synchronous asynchronous receiver transmitter
USART1: 0x40011000
USART2: 0x40004400

offset	31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0	Register
0		USART_SR
4		USART_DR
8		USART_BRR
0xC		USART_CR1
0x10		USART_CR2
0x14		USART_CR3
0x18		USART_GTPR

Function pointer

```c
#include <stdio.h>

int square(int x)
{
    return x*x;
}

int main()
{
    int (*fp)(int);

    fp = square;

    printf("fp(5)=%i \n", fp(5));
    return 0;
}
```

A function pointer

```
fp(5)=25
```

Function pointer

```
int (*fp)(int);
```

A Function pointer type is defined by:

- Return type.
- Number of arguments and their types.

The value of a function pointer is an address.

Question:
What is the definition for a function pointer that points to function foo
char *foo(int *x, int y) {…}

```
LDR R4, =fp
BLX R4
```

```
LDR R4, =foo
BLX R4
```

Structs with function pointers

Programming object-oriented style in a non-object-oriented language!

📖 Arbetsbok pg 109-111

```
typedef struct tObj {
  PGEOMETRY geo;
  int dirx, diry;
  int posx, posy;
  void (*draw)(struct tObj *);
  void (*clear)(struct tObj *);
  void (*move)(struct tObj *);
  void (*set_speed)(struct tObj *, int, int);
} OBJECT, *POBJECT;
```

What are ”draw”, ”clear”, ”move”, ”set_speed”?

Structs with function pointers

Programming object-oriented style in a non-object-oriented language

📖 Arbetsbok pg 109-111

```
typedef struct tObj {
  PGEOMETRY geo;
  int dirx, diry;
  int posx, posy;
  void (*draw)(struct tObj *);
  void (*clear)(struct tObj *);
  void (*move)(struct tObj *);
  void (*set_speed)(struct tObj *, int, int);
} OBJECT, *POBJECT;
```

How do you initialize them?

```
static OBJECT ball = {
   &ball_geometry,
   0, 0,
   1, 1,
   draw_object,
   clear_object,
   move_object,
   set_object_speed
};
```

How do you use them?

```
POBJECT p = &ball;
. . .
p->set_speed(p,4,1);
. . .
p_>move(p);
```

Dynamic Memory Allocation

- `malloc()` Allocate memory
- `free()` De-allocate (free) memory

What happens if we do not call free?

Do you need to use some sort of free also in Java? Why?

Function prototypes available in:

```
#include <stdlib.h>
```

Dynamic Memory Allocation

```c
#include <stdlib.h>

char s1[] = "This is a long string. It is even more than one sentence.";

int main()
{
    char* p;

    // allocate memory dynamically
    p = (char*)malloc(sizeof(s1));

    // make something with the memory that we have reserved

    // free the memory space
    free(p);
    return 0;
}
```

Can you guess the prototype for malloc?

After malloc ALWAYS do:

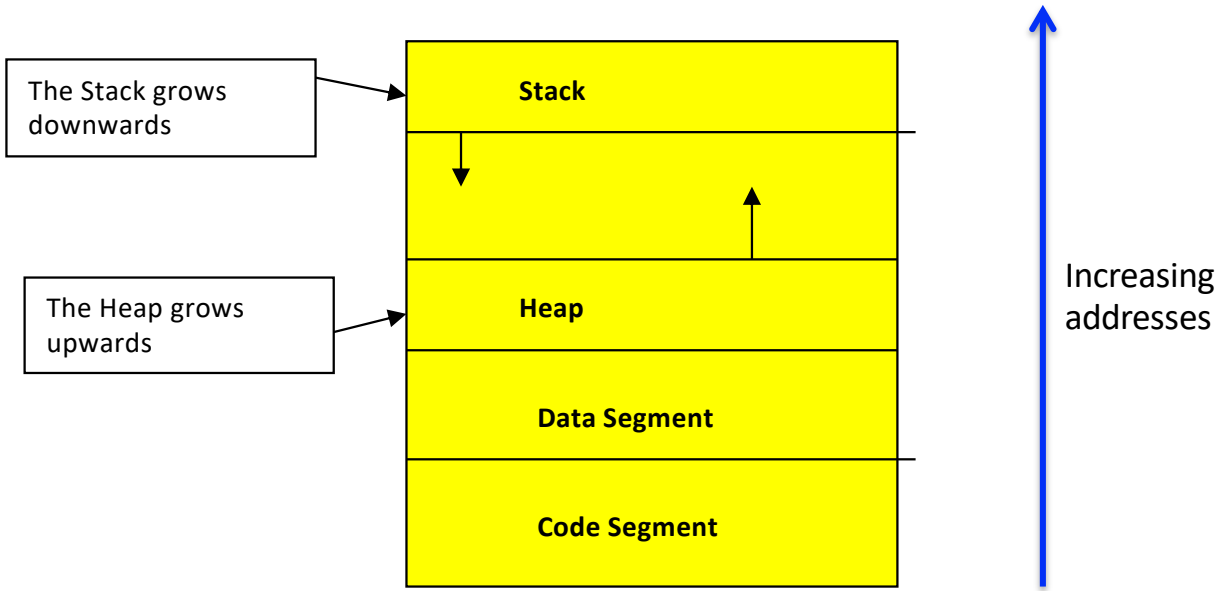
```c
if( !p ) {
   printf("ERROR: Could not allocate p\n");
   exit(-1);
}
```

Number of bytes we want to allocate

OUT OF MEMORY
virtual address space is divided into pages swapped to the secondary storage (e.g. hard disk). Even this may end -> crash.

Address Space of a Program

- All programs as they execute, have some associated memory, which is typically divided into:
  - Code Segment
  - Data Segment (Holds Global Data)
  - Stack (where the local variables and other temporary information is stored)
  - Heap



Dynamic Memory Allocation Example

```
// Copy from s1 to the allocated memory pointed by p.
#include <stdio.h>
#include <stdlib.h>

char s1[] = "This is a long string. It is even more than one sentence.";
int main()
{
        char* p;
        int i;
        // allocate memory dynamically
        p = (char*)malloc(sizeof(s1));

        // make something with the memory that is allocated
        for( i=0; i<sizeof(s1); i++)
                *(p+i) = s1[i];
        printf("%s", p);

        // free memory
        free(p);

        return 0;
}
```

Do I need to do something between allocation and use?

What happens if I do *(p+1000) = 1024

Why does free(p) not need information about its size?

Dynamic Memory Allocation – other types

```
#include <stdio.h>
#include <stdlib.h>

typedef struct {
  char *name;
  int   number;
} xyz;

int main() {
  int    *x;
  char   *y;
  double *z;
  xyz    *u;

  //allocate 100 int, char, double and xyz

}
```

```
x = (int*) malloc( sizeof(int)*100 );
if( !x ) exit(-1);
```

```
y = (char*) malloc( sizeof(char)*100 );
if( !y ) exit(-1);
```

```
z = (double*) malloc( sizeof(double)*100 );
if( !z ) exit(-1);
```

```
u = (xyz*) malloc( sizeof(xyz)*100 );
if( !u ) exit(-1);
```

Other dynamic memory allocation functions…

```
ptr = (int*) calloc(n, sizeof(int));
```

```
ptr = (int*) malloc(n1 * sizeof(int));
. . .
ptr = realloc(ptr, n2 * sizeof(int));
```

realloc()

```
#include <stdio.h>
#include <stdlib.h>

int main() {
    int *ptr, i , n1, r
    printf("Enter size
    scanf("%d", &n1)
    ptr = (int*) malloc
    printf("Addresses o
    printf("%u - %u\n",
    printf("\nEnter new
    scanf("%d", &n2);
    ptr = realloc(ptr,
    printf("Addresses o
    printf("%u - %u\n",
    return 0;
}
```

```
Enter size of array: 10
Addresses of previously allocated memory: 1599078400 - 1599078436
Enter new size of array: 20
Addresses of newly allocated memory: 1599078400 - 1599078476

Enter size of array: 10000
Addresses of previously allocated memory: 2877292544 - 2877332540
Enter new size of array: 40000
Addresses of newly allocated memory: 84406272 - 84566268

Enter size of array: 10000
Addresses of previously allocated memory: 3313500160 - 3313540156
Enter new size of array: 2000
Addresses of newly allocated memory: 3313500160 - 3313508156
```

Memory Leaks

- A memory leak occurs when we do not free a memory item that we have dynamically allocated (with malloc()).
- Memory leaks can cause system crash if it runs out of memory.
- Memory leakage disappears when program terminates. (or it should…)

Why and How?

Find a memory leak

- You can use a memory analyzer such as DrMemory (http://www.drmemory.org/) or Valgrind (http://valgrind.org)
- DrMemory replaces the default library, and analyzes calls to malloc() and free().

- It also finds accesses to uninitialized memory

DrMemory Example

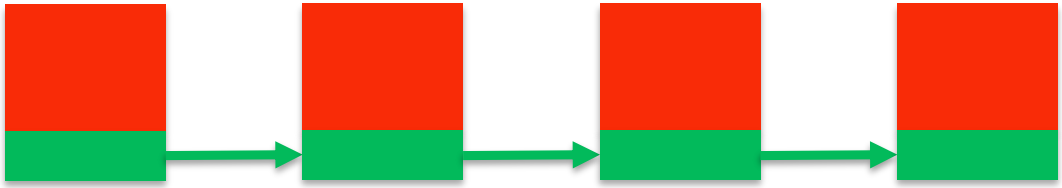
```
#include <stdio.h>
#include <stdlib.h>

void foo() {
  int *y;
  y = (int*) malloc( sizeof(int)*10 );
  //free(y);
}

int main() {
  char *x = "This is a long string";
  foo();
  foo();
  foo();
  foo();
}
```

```
$ gcc -m32 -g -fno-inline -fno-omit-frame-pointer t1.c -o t1
$ bin/drmemory -- ./t1
~~Dr.M~~ Error #3: POSSIBLE LEAK 40 direct bytes 0x005701c8-
0x005701f0 + 0 indirect bytes
~~Dr.M~~ # 0 replace_malloc
~~Dr.M~~ # 1 foo                                [/Users/t1.c:6]
~~Dr.M~~ # 2 main                               [/Users/t1.c:17]
~~Dr.M~~
~~Dr.M~~ ERRORS FOUND:
~~Dr.M~~  0 unique, 0 total unaddressable access(es)
~~Dr.M~~  0 unique, 0 total uninitialized access(es)
~~Dr.M~~  0 unique, 0 total invalid heap argument(s)
~~Dr.M~~  0 unique, 0 total warning(s)
~~Dr.M~~  0 unique, 0 total,   0 byte(s) of leak(s)
~~Dr.M~~  4 unique, 4 total, 160 byte(s) of possible leak(s)
~~Dr.M~~ ERRORS IGNORED:
~~Dr.M~~  0 unique, 0 total,   0 byte(s) of still-reachable
allocation(s)
```

Linked Lists

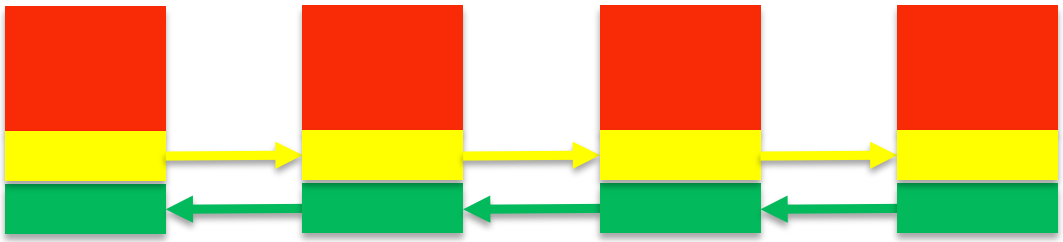


```
typedef struct node_t {
  char firstName[100];
  char lastName[100];
  int  idnumber;
  struct node_t* next;
} node_t;
```

```
int findInList(int num) {…}
int insertInList(node_t* node) {}
int removeFromList(int num) {}


int findInList(node_t* head, int num) {…}
node_t* insertInList(node_t* head, node_t* node) {}
node_t* removeFromList(node_t* head, int num) {}
```

Doubly Linked Lists



```
typedef struct node_t {
  char firstName[100];
  char lastName[100];
  int  idnumber;
  struct node_t* next;
  struct node_t* prev;
} node_t;
```

```
int findInList(int num) {…}
int insertInList(node_t* node) {}
int removeFromList(int num) {}
```

File operations

```
#include <stdio.h>

#define MAX 100

int main(void) {
  FILE *fp;
  char *filename = "x1.c";
  char str[MAX];
  int linenum = 0;

  fp = fopen(filename, "r");
  if( fp == NULL ) {
    printf( "ERROR: Could not find file %s\n", filename );
  }
  while(!feof(fp)) {
    fgets( str, MAX, fp );
    printf("%d: %s", linenum++, str);
  }
  printf("\n");
  fclose(fp);
}
```