CHALMERS
UNIVERSITY OF TECHNOLOGY

Maskinorienterad Programmering

C-Programming part 4

Erik Sintorn
erik.sintorn@chalmers.se

Original slides by Pedro Trancoso and Ulf Assarsson

Förra Lektionen

- Structs, pekare till structs, arrayer av structs
- Portaddressering med structar
- Funktionspekare
- Structar med funktionspekare

Innehåll

- Minnesrymder
  - Stack, Static och Heap
- Unions
- Enums
- Dynamic Memory

Minnesrymder i C

Stack Memory

- Fixed size minnespool (e.g. 8MB)
- Variabler i en funktion allokeras (normalt) här
- LIFO kö

```
int A() {
    int a = 1;
    return a;
}

int B() {
    int b = 2;
    return b;
}

int C() {
    int c = A() + B();
    return c;
}

int main()
{
    printf("%i\n", C());
}
```

Minnesrymder i C

Stack Memory

- Fixed size minnespool (e.g. 8MB)
- Variabler i en funktion allokeras (normalt) här
- LIFO kö

```
int F(int i)
{
    int a, b;
    if(i==0) return 0;
    else if(i==1) return 1;
    else {
        a = F(i - 1);
        b = F(i - 2);
        return a + b;
    }
}

int main()
{
    printf("%i\n", F(5000000));
}
```

Minnesrymder i C

Stack Memory

-
-
-

Static Memory

- Allokeras när programmet startar
- Globala variabler hamnar här
- Variabler med *static* qualifier hamnar här
- Finns allokerat under programets livslängd

```
int sum = 0;

void A() {
    sum += 1;
}

void B() {
    sum += 2;
}

void C() {
    A();
    B();
}

int main()
{
    C();
    printf("%i\n", sum);
}
```

Minnesrymder i C

Stack Memory

-
-
-

Static Memory

- Allokeras när programmet startar
- Globala variabler hamnar här
- Variabler med *static* qualifier hamnar här
- Finns allokerat under programets livslängd

```c
#include <stdio.h>
#define NUM_IMAGES 10

typedef struct
{
    int pixel[1024][1024];
} Image;

int main(int argc, char **argv)
{
    Image images[NUM_IMAGES];

    for(int i=0; i < NUM_IMAGES; i++)
    {
        for(int r=0; r<1024; r++){
            for(int c=0; c<1024; c++){
                images[i].pixel[r][c] = 0;
            }
        }
    }
}
```

Minnesrymd

Stack Memory

-
-
-

Static Memory

- Allokeras när programmet startar
- Globala variabler hamnar här
- Variabler med *static* qualifier hamnar här
- Finns allokerat under programets livslängd

```c
#include <stdio.h>
#define NUM_IMAGES 10

typedef struct
{
    int pixel[1024][1024];
} Image;

Image images[NUM_IMAGES];

int main(int argc, char **argv)
{
    for(int i=0; i < NUM_IMAGES; i++)
    {
        for(int r=0; r<1024; r++){
            for(int c=0; c<1024; c++){
                images[i].pixel[r][c] = 0;
            }
        }
    }
}
```

Minnesrymder i C

Stack Memory

-
-
-

Static Memory

- Allokeras när programmet startar
- Globala variabler hamnar här
- Variabler med *static* qualifier hamnar här
- Finns allokerat under programets livslängd

```c
#include <stdio.h>
#define NUM_IMAGES 10

typedef struct
{
    int pixel[1024][1024];
} Image;

int main(int argc, char **argv)
{
    static Image images[NUM_IMAGES];

    for(int i=0; i < NUM_IMAGES; i++)
    {
        for(int r=0; r<1024; r++){
            for(int c=0; c<1024; c++){
                images[i].pixel[r][c] = 0;
            }
        }
    }
}
```

Minnesrymder i C

Stack Memory

-
-
-

Static Memory

-
-
-
-

Heap Memory

- Resten av minnet ☺
- Kontrolleras av operativsystemet
- Allokeras dynamiskt med *malloc*

```
int main()
{
    BigObject * big_object = malloc(sizeof(BigObject));
    // ... use big_object ...
    free(big_object);
}
```

Enums

```c
#define MONSTER_TYPE_GOOD 0x01
#define MONSTER_TYPE_NEUTRAL 0x02
#define MONSTER_TYPE_EVIL 0x03

typedef struct
{
    int type;
} Monster;

int attack(Monster * monster, Player * player)
{
    if(monster->type == MONSTER_TYPE_GOOD) {
        // Do something nice
    }
    else if(monster->type == MONSTER_TYPE_NEUTRAL) {
        // Do nothing
    }
    else if(monster->type == MONSTER_TYPE_EVIL) {
        // Do something nasty
    }
}
```

```c
typedef enum { Nice, Neutral, Evil } MonsterType;

typedef struct
{
    MonsterType type;
} Monster;

int attack(Monster * monster, Player * player)
{
    if(monster->type == Nice) {
        // Do something nice
    }
    else if(monster->type == Neutral) {
        // Do nothing
    }
    else if(monster->type == Evil) {
        // Do something nasty
    }
}
```

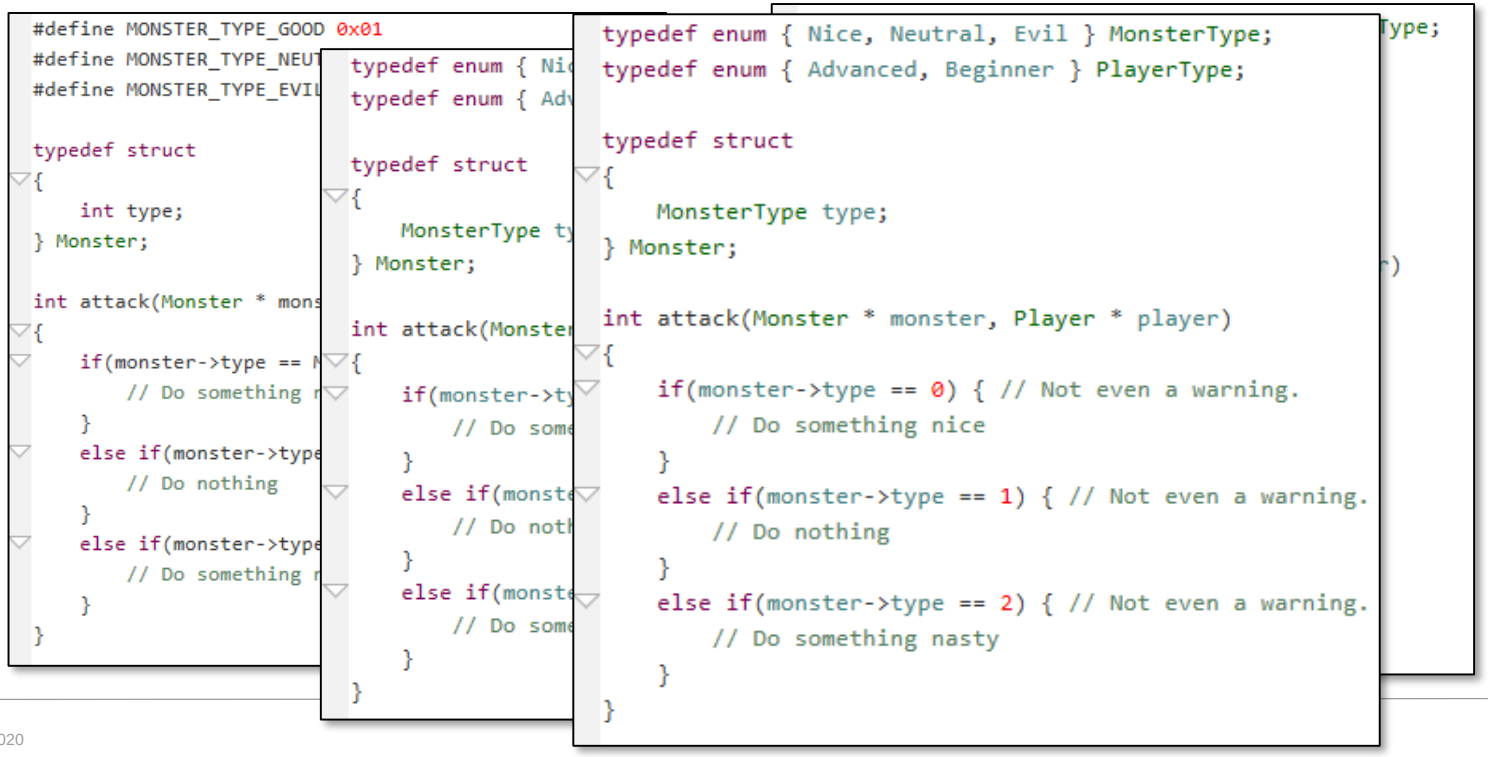
Enums

```
#define MONSTER_TYPE_GOOD 0x01
#define MONSTER_TYPE_NEUT
#define MONSTER_TYPE_EVIL

typedef struct
{
    int type;
} Monster;

int attack(Monster * mons
{
    if(monster->type == M
        // Do something n
    }
    else if(monster->type
        // Do nothing
    }
    else if(monster->type
        // Do something n
    }
}
```

```
typedef enum { Nice, Neutral, Evil } MonsterType;
typedef enum { Advanced, Beginner } PlayerType;

typedef struct
{
    MonsterType type;
} Monster;

int attack(Monster * monster, Player * player)
{
    if(monster->type == Nice) {
        // Do something nice
    }
    else if(monster->type == Beginner) { // Warning, but not error!
        // Do nothing
    }
    else if(monster->type == Evil) {
        // Do something nasty
    }
}
```

```
typedef enum { Nice, Neutral, Evil } MonsterType;

yer * player)

ral) {

) {
```

Enums



```
#define MONSTER_TYPE_GOOD 0x01
#define MONSTER_TYPE_NEUT
#define MONSTER_TYPE_EVIL

typedef struct
{
    int type;
} Monster;

int attack(Monster * mons
{
    if(monster->type == M
        // Do something n
    }
    else if(monster->type
        // Do nothing
    }
    else if(monster->type
        // Do something n
    }
}
```

```
typedef enum { Nic
typedef enum { Adv

typedef struct
{
    MonsterType ty
} Monster;

int attack(Monster
{
    if(monster->ty
        // Do some
    }
    else if(monste
        // Do noth
    }
    else if(monste
        // Do some
    }
}
```

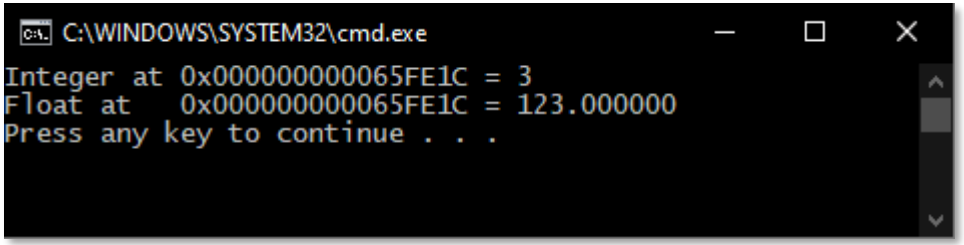
```
typedef enum { Nice, Neutral, Evil } MonsterType;
typedef enum { Advanced, Beginner } PlayerType;

typedef struct
{
    MonsterType type;
} Monster;

int attack(Monster * monster, Player * player)
{
    if(monster->type == 0) { // Not even a warning.
        // Do something nice
    }
    else if(monster->type == 1) { // Not even a warning.
        // Do nothing
    }
    else if(monster->type == 2) { // Not even a warning.
        // Do something nasty
    }
}
```

```
Type;

r)
```

```c
typedef enum {  INFO,     // = 0
                WARNING,  // = 1
                ERROR,    // = 2
                FATAL     // = 3
             } LogLevel;

LogLevel log_verbosity = WARNING; // = 1

void log(char * message, LogLevel level)
{
    if(level >= log_verbosity) {
        printf("Log: %s\n", message);
    }
}

int main()
{
    log_verbosity = ERROR;
    log("Program started.", INFO);
    for(int counter=0; counter<100; counter++) {
        log("Counter increased", INFO);
        if(counter > 50) log("Counter is pretty high.", WARNING);
        if(counter == 80) log("Counter is too high!", ERROR);
        if(counter == 99) log("Explosion is imminent!", FATAL);
    }
}
```

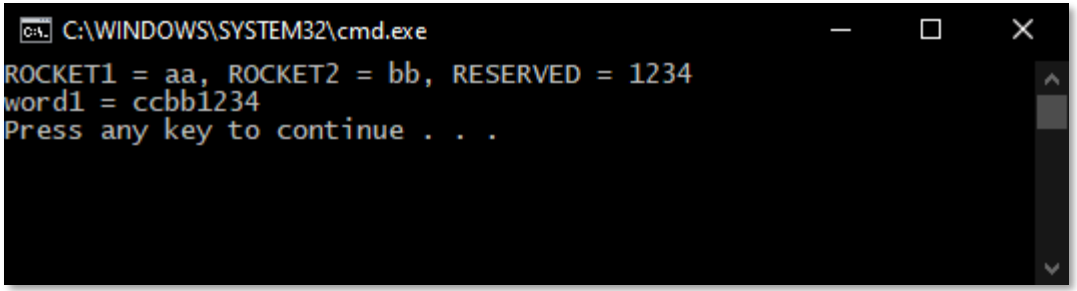
Unions

```c
int main()
{
    union { int an_integer; float a_float } value;
    value.an_integer = 3;
    printf("Integer at 0x%p = %i\n", &value.an_integer, value.an_integer);
    value.a_float = 123.0f;
    printf("Float at 0x%p = %f\n", &value.a_float, value.a_float);
}
```



```c
int main()
{
    typedef struct {
        union {
            unsigned int word0;
            unsigned int status;
        };
        union {
            unsigned int word1;
            struct {
                unsigned short reserved;
                unsigned char rocket2;
                unsigned char rocket1;
            };
        };
    } DDD;

    DDD ddd;
    ddd.word0 = 0xFFAA3322;
    ddd.word1 = 0xAABB1234;
    printf("ROCKET1 = %x, ROCKET2 = %x, RESERVED = %x\n",
            ddd.rocket1, ddd.rocket2, ddd.reserved);
    ddd.rocket1 = 0xCC;
    printf("word1 = %x\n", ddd.word1);
}
```

Unions



Doomsday Device Port Specification

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
0	STATUS																															
4	ROCKET1																ROCKET2															
	reserved																															

```
typedef enum { Fighter, Wizard } MonsterType;
typedef struct
{
    char * name;
    MonsterType type;
    union {
        struct FighterStats {
            int damage;
            int armour;
        };
        struct WizardStats {
            int mana_points;
            int hat_size;
            float staff_damage;
        }
    }
} Monster;
```

Unions

```
void PrintMonster(Monster * monster)
{
    printf("Name: %s\n", monster->name);
    switch(monster->type)
    {
        case Fighter:
            printf("Fighter\n");
            printf("Weapon damage: %i\n", monster->damage);
            printf("Armour: %i\n", monster->armour);
            break;
        case Wizard:
            printf("Wizard\n");
            printf("Mana points: %i\n", monster->mana_points);
            printf("Hat Size: %i\n", monster->hat_size);
            printf("Staff Damage: %f\n", monster->staff_damage);
            break;
        default:
            printf("Warning: Unknown monster type.\n");
    }
    printf("\n");
}
```

Unions

```
typedef enum { Fighter, Wizard } MonsterType;
typedef struct
{
    char * name;
    MonsterType type;
    union {
        struct FighterStats {
            int damage;
            int armour;
        };
        struct WizardStats {
            int mana_points;
            int hat_size;
            float staff_damage;
        }
    }
} Monster;
```

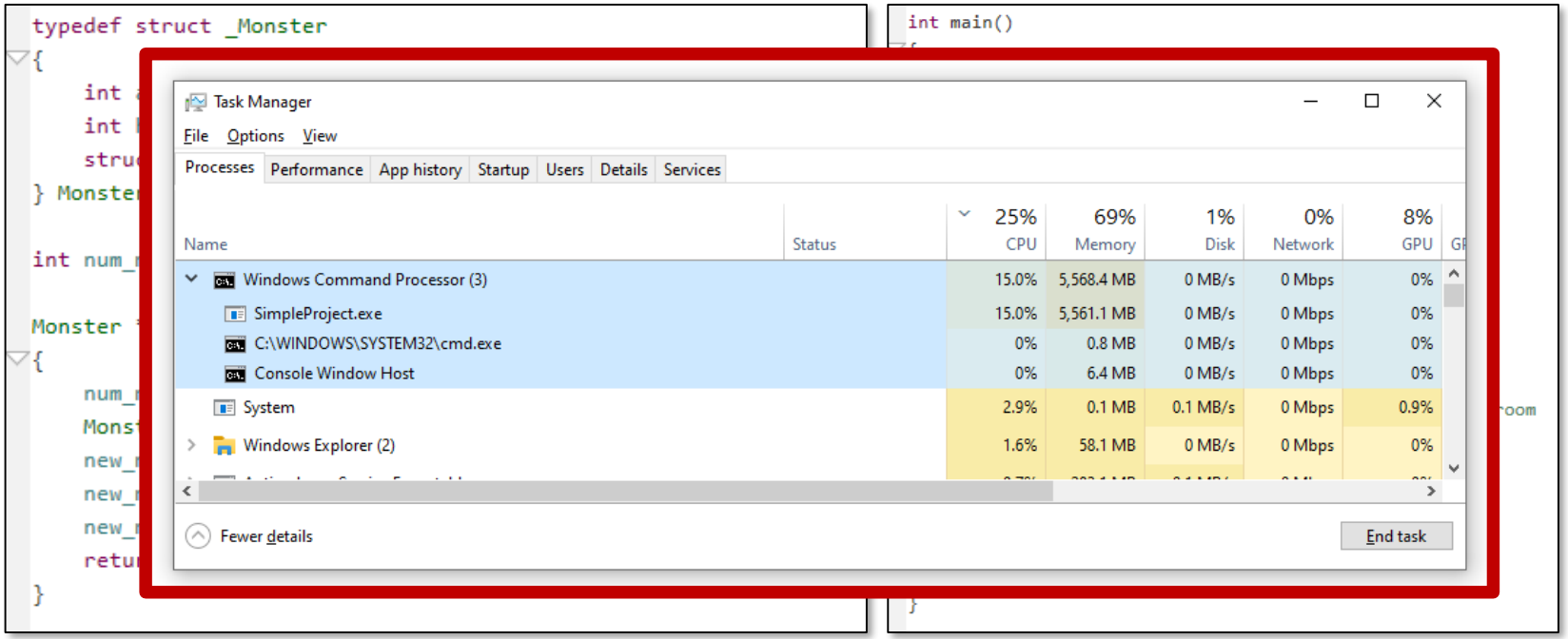
```
int main()
{
    Monster monsters[] = {
        {.name = "Conan", .type = Fighter, .damage = 5, .armour = 2},
        {.name = "Gandalf", .type = Wizard, .mana_points = 4,
         .hat_size = 100, .staff_damage = 5}
    };
    PrintMonster(&monsters[0]);
    PrintMonster(&monsters[1]);
}
```

```
C:\WINDOWS\SYSTEM32\cmd.exe
Name: Conan
Fighter
Weapon damage: 5
Armour: 2

Name: Gandalf
Wizard
Mana points: 4
Hat Size: 100
Staff Damage: 5.000000

Press any key to continue . . .
```

Dynamic Memory

```
int main()
{
    while(1)
    {
        // Add a monster to the room

        // For each monster
        {
            // Fight the monster
            // If the monster dies, remove it from the room
        }

        // If player health < 0, player loses
        // If no monsters left in room, player wins!
    }
    printf("Game Over.\n");
}
```

Dynamic Memory



Dynamic Memory

```
int main()
{
    Monster * first_monster = 0;
    int health = 100;
    int turn = 0;

    while(1)
    {
        // Add a monster to the room
        Monster * new_monster = CreateMonster();
        new_monster->next_monster = first_monster;
        first_monster = new_monster;

        // For each monster
        Monster * current_monster = first_monster;
        do
        {
            // Fight the monster
            int dice = rand() % 6;
            current_monster->health -= dice;
            if(current_monster->health < 0) {
                    // If the monster dies, remove it from the room
            }
            else {
                health -= current_monster->attack;
            }
            current_monster = current_monster->next_monster;
        } while(current_monster != 0);

        // If player health < 0, player loses
        if(health < 0) {
            printf("You died!\n");
            break;
        }
        // If no monsters left in room, player wins!

        printf("Turn %i: %i monsters, health = %i\n", turn++, num_monsters, health);
    }
    printf("Game Over.\n");
}
```

Dynamic Memory

```
Monster * RemoveMonster(Monster * first_monster, Monster * dead_monster)
{
    printf("Killed a monster!\n");
    if(first_monster == dead_monster) {
        return first_monster->next_monster;
    }
    Monster * current_monster = first_monster;
    while(current_monster->next_monster != dead_monster)
        current_monster = current_monster->next_monster;
    current_monster->next_monster = dead_monster->next_monster;
    free(dead_monster);
    return current_monster;
}
```

Memory Leak!

```
// For each monster
Monster * current_monster = first_monster;
do
{
    // Fight the monster
    int dice = rand() % 6;
    current_monster->health -= dice;
    // If the monster dies, remove it from the room
    if(current_monster->health < 0)
        current_monster = RemoveMonster(first_monster, current_monster);
    else {
        health -= current_monster->attack;
    }
    if(current_monster == 0) break;
    else current_monster = current_monster->next_monster;
} while(current_monster != 0);
```

Dynamic Memory

```
Monster * RemoveMonster(Monster * first_monster, Monster * dead_monster)
{
    printf("Killed a monster!\n");
    num_monsters -= 1;
    if(first_monster == dead_monster) {
        free(dead_monster);
        return first_monster->next_monster;
    }
    Monster * current_monster = first_monster;
    while(current_monster->next_monster != dead_monster)
        current_monster = current_monster->next_monster;
    current_monster->next_monster = dead_monster->next_monster;
    free(dead_monster);
    return current_monster;
}
```

Dead Memory!

```
// For each monster
Monster * current_monster = first_monster;
do
{
    // Fight the monster
    int dice = rand() % 6;
    current_monster->health -= dice;
    // If the monster dies, remove it from the room
    if(current_monster->health < 0)
        current_monster = RemoveMonster(first_monster, current_monster);
    else {
        health -= current_monster->attack;
    }
    if(current_monster == 0) break;
    else current_monster = current_monster->next_monster;
} while(current_monster != 0);
```

Dynamic Memory

```
Monster * RemoveMonster(Monster * first_monster, Monster * dead_monster)
{
    printf("Killed a monster!\n");
    num_monsters -= 1;
    if(first_monster == dead_monster) {
        Monster * next_monster = first_monster->next_monster;
        free(dead_monster);
        return next_monster;
    }
    Monster * current_monster = first_monster;
    while(current_monster->next_monster != dead_monster)
        current_monster = current_monster->next_monster;
    current_monster->next_monster = dead_monster->next_monster;
    free(dead_monster);
    return current_monster;
}
```

```
// For each monster
Monster * current_monster = first_monster;
do
{
    // Fight the monster
    int dice = rand() % 6;
    current_monster->health -= dice;
    // If the monster dies, remove it from the room
    if(current_monster->health < 0)
        current_monster = RemoveMonster(first_monster, current_monster);
    else {
        health -= current_monster->attack;
    }
    if(current_monster == 0) break;
    else current_monster = current_monster->next_monster;
} while(current_monster != 0);
```

Dynamic Memory

```c
typedef struct _Monster
{
    int attack;
    int health;
    struct _Monster * next_monster;
} Monster;

int num_monsters = 0;

Monster * CreateMonster()
{
    num_monsters += 1;
    Monster * new_monster = (Monster *)malloc(sizeof(Monster));
    new_monster->attack = rand() % 5;
    new_monster->health = 10;
    new_monster->next_monster = 0;
    return new_monster;
}

Monster * RemoveMonster(Monster * first_monster, Monster * dead_monster)
{
    printf("Killed a monster!\n");
    num_monsters -= 1;
    if(first_monster == dead_monster) {
        Monster * next_monster = first_monster->next_monster;
        free(dead_monster);
        return next_monster;
    }
    Monster * current_monster = first_monster;
    while(current_monster->next_monster != dead_monster)
        current_monster = current_monster->next_monster;
    current_monster->next_monster = dead_monster->next_monster;
    free(dead_monster);
    return current_monster;
}

int main()
{
    Monster * first_monster = 0;
    int health = 100;
    int turn = 0;

    while(1)
    {
        // Add a monster to the room
        Monster * new_monster = CreateMonster();
        new_monster->next_monster = first_monster;
        first_monster = new_monster;

        // For each monster
        Monster * current_monster = first_monster;
        do
        {
            // Fight the monster
            int dice = rand() % 6;
            current_monster->health -= dice;
            // If the monster dies, remove it from the room
            if(current_monster->health < 0)
                current_monster = RemoveMonster(first_monster, current_monster);
            else {
                health -= current_monster->attack;
            }
            if(current_monster == 0) break;
            else current_monster = current_monster->next_monster;
        } while(current_monster != 0);

        // If player health < 0, player loses
        if(health < 0) {
            printf("You died!\n");
            break;
        }
        // If no monsters left in room, player wins!
        if(num_monsters == 0) {
            printf("You won!\n");
            break;
        }
        printf("Turn %i: %i monsters, health = %i\n", turn++, num_monsters, health);
    }
    printf("Game Over.\n");
}
```

```
typedef struct _Monster
{
    int attack;
    int health;
    struct _Monster * next_mons
} Monster;

int num_monsters = 0;

Monster * CreateMonster()
{
    num_monsters += 1;
    Monster * new_monster = (Mo
    new_monster->attack = rand(
    new_monster->health = 10;
    new_monster->next_monster =
    return new_monster;
}

Monster * RemoveMonster(Monster
{
    printf("Killed a monster!\n
    num_monsters -= 1;
    if(first_monster == dead_mo
        Monster * next_monster
        free(dead_monster);
        return next_monster;
    }
    Monster * current_monster =
    while(current_monster->next
        current_monster = curre
```

```
C:\WINDOWS\SYSTEM32\cmd.exe
Turn 0: 1 monsters, health = 99
Turn 1: 2 monsters, health = 94
Killed a monster!
Turn 2: 2 monsters, health = 86
Turn 3: 3 monsters, health = 74
Turn 4: 4 monsters, health = 60
Turn 5: 5 monsters, health = 44
Killed a monster!
Killed a monster!
Killed a monster!
Turn 6: 3 monsters, health = 38
Turn 7: 4 monsters, health = 31
Killed a monster!
Killed a monster!
Turn 8: 3 monsters, health = 26
Turn 9: 4 monsters, health = 19
Killed a monster!
Turn 10: 4 monsters, health = 13
Killed a monster!
Killed a monster!
Killed a monster!
Turn 11: 2 monsters, health = 10
Turn 12: 3 monsters, health = 4
Killed a monster!
You died!
Game Over.
Press any key to continue . . .
```

```
>next_monster;

*, turn++, num_monsters, health);
```