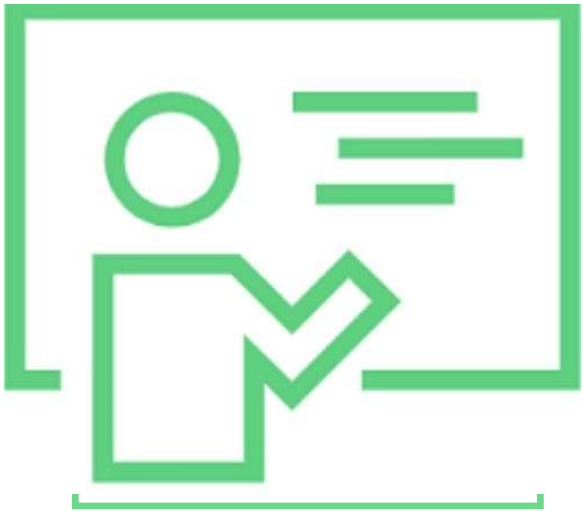

Structs etc

Stefan Holmberg, Systementor AB

Komplexa datastrukturer

I C finns endast en typ av “komplex” datastruktur:

- Sammansatta datatyp (*struct* och *union*)

struct används för att hålla ihop relaterade variabler

- Närmaste C kommer till klasser i C++ / Java

union används för att ha en variabel som kan vara en av fler olika typer

- C fördelar då tillräckligt mycket minne så att den största typ ryms
- Ovanligt
- Nyfiken? http://www.geeksforgeeks.org/union-c/

Deklarera en struct #1

struct1.c

```c
struct person {
   char name[50];
   int age;
};
int main() {
    struct person user;  user.age = 42;
    printf( "%d\n", user.age );
    return 0;
}
```

Struct

- Deklarera en *struct*, *pers*
- Deklarera en variabel so
  är av typen *person* (*stru*
- Tillgång till medlemmar
  med “.”

Initiera

struct2.c

```
struct point {
    int x = 0;
    int y = 0;
};
int main() {
    struct point start = { 0,0 };
    printf( "%d, %d\n", start.x, start.y);
    return 0;
}
```

Initialisera en struct:

- Går INTE att göra i deklarationen (ingen constructor)
- Gör istället där den används
  - Dvs: deklarationen av variabeln

Deklarera en struct #2

typedef

```
typedef struct {
    int Age;
    int Jersey;
    char Name[100];
}Player;

int main(){
    Player p1 = {25,21,"Foppa"};
    p1.Age = 50;
    strcpy(p1.Name,”Peter”);
    printf("%d",p1.Age);
    return 0;
}
```

Åtkomst till delar av en struct

```
typedef struct {
    int Age;
    int Jersey;
    char Name[100];
}Player;

int main(){
    Player p1 = {25,21,"Foppa"};
    p1.Age  = 50;
    printf("%d\n",p1.Age);
    printf("%d\n",p1.Jersey);
    printf("%s\n",p1.Name);
    return 0;
}
```

Med punkt

Som andra programmeringsspråk

Skicka struct till funktion

```
typedef struct {
    int Age;
    int Jersey;
    char Name[100];
}Player;

void PrintPlayer(Player p){
    printf("%d\n",p.Age);
    printf("%d\n",p.Jersey);
    printf("%s\n",p.Name);
}

int main(){
    Player p1 = {25,21,"Foppa"};
    p1.Age  = 50;
    PrintPlayer(p1);
    return 0;
}
```

OBS: Copy by value – som alltid!

En struct kan ta MÅÅÅNGA bytes.

```
printf("%d\n", (int)sizeof(p1));
```
→ 108

Copy by value kan ta mycket CPU+R

Copy by value proof

```
typedef struct {
    int Age;
    int Jersey;
    char Name[100];
}Player;

void PrintPlayer(Player p){
    strcpy(p.Name, "Whatever");
    printf("%s\n",p.Name);
}

int main(){
    Player p1 = {25,21,"Foppa"};
    PrintPlayer(p1);
    printf("%s\n", p1.Name);
    return 0;
}
```

```
Whatever
Foppa
```

Send by pointer – nytt tecken ->

```
typedef struct {
    int Age;
    int Jersey;
    char Name[100];
}Player;

void PrintPlayer(Player *p){
    strcpy(p->Name, "Whatever");
    printf("%d\n",p->Age);
    printf("%s\n",p->Name);
}

int main(){
    Player p1 = {25,21,"Foppa"};
    PrintPlayer(&p1);
    printf("%s\n", p1.Name);
    return 0;
}
```

Komma åt delar när det är en PEKARE

Då använder vi pil-operator

```
Whatever
Whatever
```

Kopiera structar

```
typedef struct {
    int Age;
    int Jersey;
    char Name[100];
}Player;


int main(){
    Player p1 = {25,21,"Foppa"};
    Player p2;
    p2 = p1;
    printf("%s\n", p2.Name);
    return 0;
}
```

För c är en struct bara en massa bytes…

Så det går bra att assigna en struct till en
= den bara koperar alla bytes från den e
den andra

Smidigt att slippa strcpy. Men SE UPP: fö
structar etc kanske man kan jobba med p
stället?

Dvs behöver du verkligen en kopia?

Arrayer med structar

```
typedef struct {
    int Age;
    int Jersey;
    char Name[100];
}Player;


int main(){
    Player team[100];
    Player p1 = {25,21,"Foppa"};
    Player p2;
    p2 = p1;
    team[0] =p1;
    printf("%s\n", team[0].Name);
    return 0;
}
```

Du måste hålla reda på hur MÅNGA som
Dvs ingen automatisk 0-terminering som
strängar!