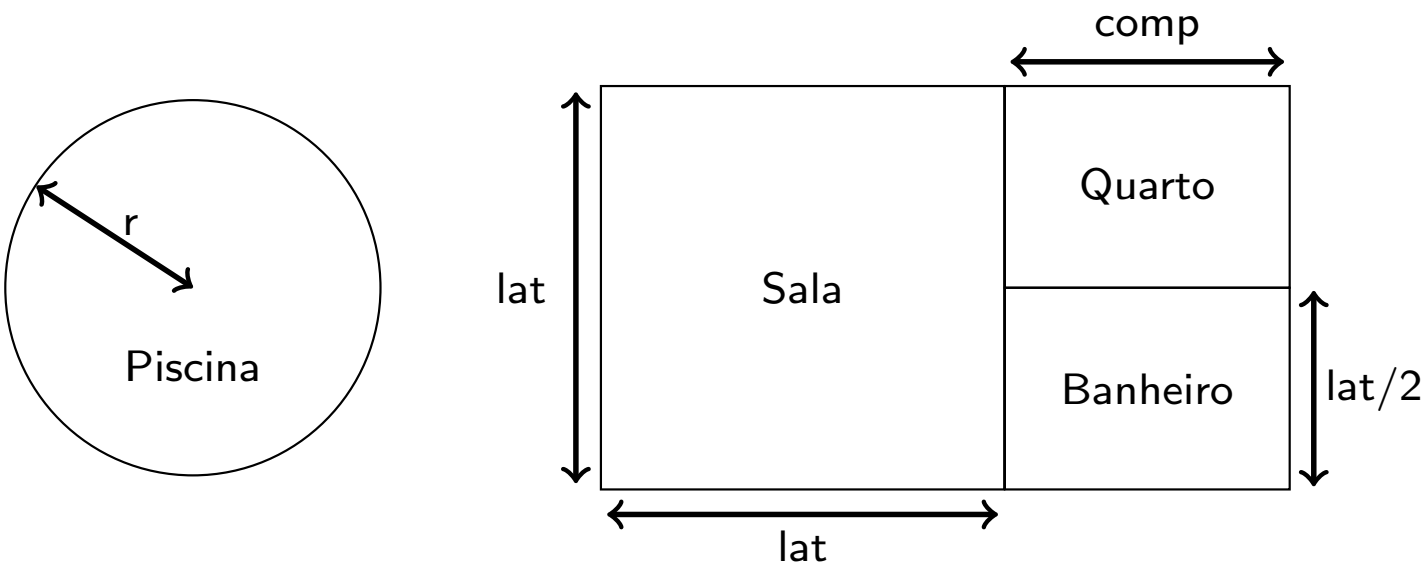
Aula 25 – Estruturas (parte 1)

Norton T. Roman & Luciano A. Digiampietri

Estruturas de Dados

- Voltemos à representação da nossa "cabana":



Estruturas de Dados

- Seria interessante agruparmos todas as informações da “cabana” em um único local

Estruturas de Dados

- Seria interessante agruparmos todas as informações da “cabana” em um único local
  - Temos informações da casa, propriamente dita:

Estruturas de Dados

- Seria interessante agruparmos todas as informações da “cabana” em um único local
  - Temos informações da casa, propriamente dita:
    - Lateral da sala quadrada e comprimento do quarto

Estruturas de Dados

- Seria interessante agruparmos todas as informações da "cabana" em um único local
  - Temos informações da casa, propriamente dita:
    - Lateral da sala quadrada e comprimento do quarto
  - Temos informações da piscina:

Estruturas de Dados

- Seria interessante agruparmos todas as informações da "cabana" em um único local
  - Temos informações da casa, propriamente dita:
    - Lateral da sala quadrada e comprimento do quarto
  - Temos informações da piscina:
    - Raio da piscina

Estruturas de Dados

- Como faremos essa estruturação/agrupamento?

Estruturas de Dados

- Como faremos essa estruturação/agrupamento?
- Na linguagem C, podemo agregar diferentes campos em uma **estrutura**

Estruturas de Dados

- Como faremos essa estruturação/agrupamento?
- Na linguagem C, podemo agregar diferentes campos em uma **estrutura**
- Uma estrutura pode agregar diversos *campos* de diferentes tipos

Estruturas de Dados

- Sintaxe:

Estruturas de Dados

- Sintaxe:

```
struct <identificador> {
   <listagem de campos>;
};
```

Estruturas de Dados

- Sintaxe:

```
struct <identificador> {
   <listagem de campos>;
};
```

  - `identificador`: nome da estrutura

Estruturas de Dados

- Sintaxe:

```
struct <identificador> {
   <listagem de campos>;
};
```

  - `identificador`: nome da estrutura
  - `lista de campos`: conjunto de campos da estrutura no formato `tipo nome;`

Estruturas de Dados

- Sintaxe:

```
struct <identificador> {
   <listagem de campos>;
};
```

Estruturas de Dados

- Sintaxe:

```
struct <identificador> {
   <listagem de campos>;
};
```

- Exemplo:

Estruturas de Dados

- Sintaxe:

```
struct <identificador> {
   <listagem de campos>;
};
```

- Exemplo:

```
struct casa {
   float lateral;
   float cquarto;
};
```

Estruturas de Dados

- As estruturas são como novos tipos de dados, compostos por diversos campos

Estruturas de Dados

- As estruturas são como novos tipos de dados, compostos por diversos campos
- Como criamos uma variável do "tipo" estrutura?

Estruturas de Dados

- As estruturas são como novos tipos de dados, compostos por diversos campos
- Como criamos uma variável do "tipo" estrutura?
- `struct <idetificador> <nome da variável>;`
  - `identificador`: nome da estrutura que criamos
  - `nome da variável`: nome da variável que será do "tipo" da estrutura

Estruturas de Dados

```
#include <stdio.h>


struct casa {
  float lateral;
  float cquarto;
};

int main() {
  struct casa c1;
  return 0;
}
```

Estruturas de Dados

```c
#include <stdio.h>

/* Estrutura para agrupar as informações de uma casa */
struct casa {
  float lateral;  // lateral da casa
  float cquarto;  // comprimento do quarto
};

int main() {
  struct casa c1;
  return 0;
}
```

- Lembrem-se de inserir comentários nos códigos

Estruturas de Dados

```
#include <stdio.h>

/* Estrutura para agrupar as informações de uma casa */
struct casa {
  float lateral;  // lateral da casa
  float cquarto;  // comprimento do quarto
};

int main() {
  struct casa c1;
  return 0;
}
```

- E como acessamos os campos de c1?

Estruturas de Dados

- Com um `.` (ponto): `nome_da_variavel.campo`

Estruturas de Dados

- Com um . (ponto): `nome_da_variavel.campo`

```
#include <stdio.h>

struct casa {
  float lateral;
  float cquarto;
};

int main() {
  struct casa c1;
  c1.lateral = 11;
  c1.cquarto = 7;

  printf("A lateral da casa vale: %.2f\n", c1.lateral);
  return 0;
}
```

Estruturas de Dados

- É comum definirmos estruturas em conjunto com o uso do `typedef`

Estruturas de Dados

- É comum definirmos estruturas em conjunto com o uso do `typedef`
  - Assim, não precisamos usar a palavra `struct` sempre que iremos criar variáveis do tipo da estrutura ou na hora de definir parâmetros ou retornos de funções

Estruturas de Dados

- É comum definirmos estruturas em conjunto com o uso do `typedef`
  - Assim, não precisamos usar a palavra `struct` sempre que iremos criar variáveis do tipo da estrutura ou na hora de definir parâmetros ou retornos de funções
  - Sintaxe do *typedef*:
    `typedef <tipo de dado/estrutura> <novo nome>`

```
typedef struct aux {
  float lateral;
  float cquarto;
} casa;
```

Estruturas de Dados

```c
#include <stdio.h>

typedef struct aux {
  float lateral;
  float cquarto;
} casa;

int main() {
  struct aux c1;
  casa c2;
  c1.lateral = 11;
  c2.lateral = 15;

  printf("A lateral da casa1 vale: %.2f\n", c1.lateral);
  printf("A lateral da casa2 vale: %.2f\n", c2.lateral);
  return 0;
}
```

Estruturas de Dados

- Estruturas podem ser compostas por diversos campos e outras estruturas

Estruturas de Dados

- Estruturas podem ser compostas por diversos campos e outras estruturas

```
typedef struct auxCasa {
  float lateral;
  float cquarto;
} casa;

typedef struct auxPisc {
  float raio;
} piscina;

typedef struct auxCaba {
  casa cas;
  piscina pis;
} cabana;
```

Estruturas de Dados

- Estruturas podem ser compostas por diversos campos e outras estruturas

```
typedef struct auxCasa {
  float lateral;
  float cquarto;
} casa;

typedef struct auxPisc {
  float raio;
} piscina;

typedef struct auxCaba {
  casa cas;
  piscina pis;
} cabana;
```

Estruturas de Dados

- Estruturas podem ser compostas por diversos campos e outras estruturas

```
typedef struct auxCasa {
  float lateral;
  float cquarto;
} casa;

typedef struct auxPisc {
  float raio;
} piscina;

typedef struct auxCaba {
  casa cas;
  piscina pis;
} cabana;
```

Estruturas de Dados

```c
#include <stdio.h>


typedef struct auxPisc {
  float raio;
} piscina;

typedef struct auxCasa {
  float lateral;
  float cquarto;
} casa;

typedef struct auxCaba {
  casa cas;
  piscina pis;
} cabana;
```

```c
int main() {
  casa c1;
  c1.lateral = 11;
  c1.cquarto = 15;

  piscina p1;
  p1.raio = 3.5;

  cabana cab1;
  cab1.cas = c1;
  cab1.pis = p1;
  cab1.cas.lateral = 12;

  printf("A lateral da casa da cabana1
      vale: %.2f\n", cab1.cas.lateral);
  printf("O raio da piscina da cabana1
        vale: %.2f\n", cab1.pis.raio);
  return 0;
}
```

Estruturas de Dados

```c
#include <stdio.h>


typedef struct auxPisc {
  float raio;
} piscina;

typedef struct auxCasa {
  float lateral;
  float cquarto;
} casa;

typedef struct auxCaba {
  casa cas;
  piscina pis;
} cabana;
```

```c
int main() {
  casa c1;
  c1.lateral = 11;
  c1.cquarto = 15;

  piscina p1;
  p1.raio = 3.5;

  cabana cab1;
  cab1.cas = c1;
  cab1.pis = p1;
  cab1.cas.lateral = 12;

  printf("A lateral da casa da cabana1
      vale: %.2f\n", cab1.cas.lateral);
  printf("O raio da piscina da cabana1
        vale: %.2f\n", cab1.pis.raio);
  return 0;
}
```

Estruturas de Dados

```c
#include <stdio.h>


typedef struct auxPisc {
  float raio;
} piscina;

typedef struct auxCasa {
  float lateral;
  float cquarto;
} casa;

typedef struct auxCaba {
  casa cas;
  piscina pis;
} cabana;
```

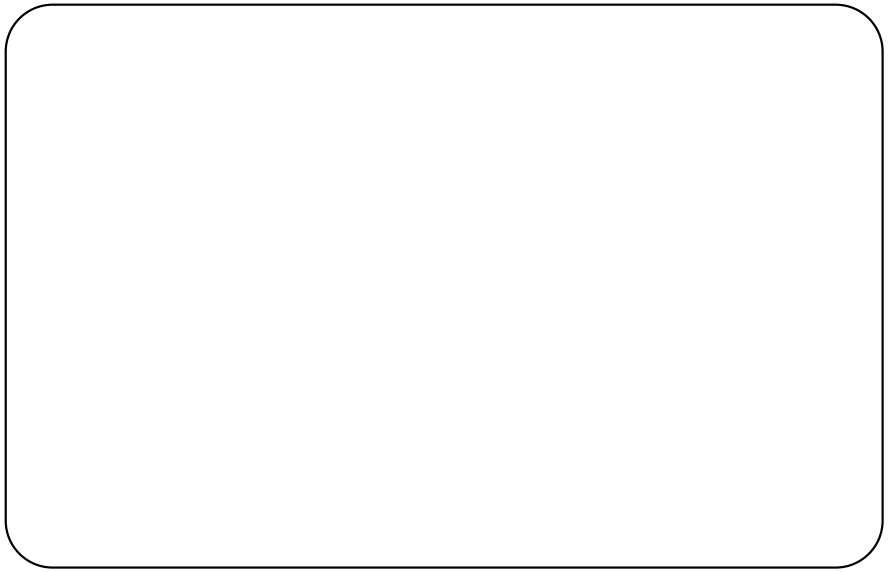
```c
int main() {
  casa c1;
  c1.lateral = 11;
  c1.cquarto = 15;

  piscina p1;
  p1.raio = 3.5;

  cabana cab1;
  cab1.cas = c1;
  cab1.pis = p1;
  cab1.cas.lateral = 12;

  printf("A lateral da casa da cabana1
      vale: %.2f\n", cab1.cas.lateral);
  printf("O raio da piscina da cabana1
        vale: %.2f\n", cab1.pis.raio);
  return 0;
}
```

Estruturas e Memória



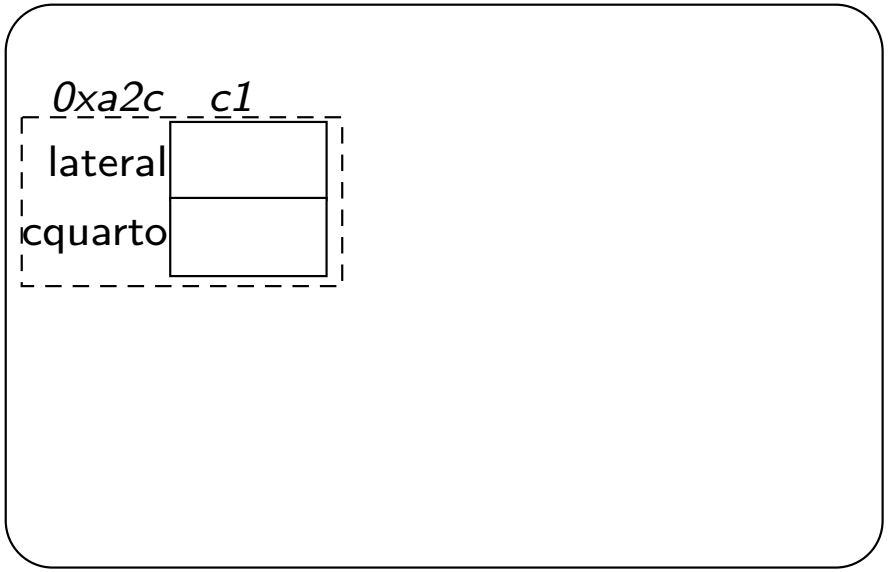
```
int main() {
  casa c1;
  c1.lateral = 11;
  c1.cquarto = 15;

  piscina p1;
  p1.raio = 3.5;

  cabana cab1;
  cab1.cas = c1;
  cab1.pis = p1;
  cab1.cas.lateral = 12;

  printf("A lateral da casa da cabana1
      vale: %.2f\n", cab1.cas.lateral);
  printf("O raio da piscina da cabana1
         vale: %.2f\n", cab1.pis.raio);
  return 0;
}
```

Estruturas e Memória



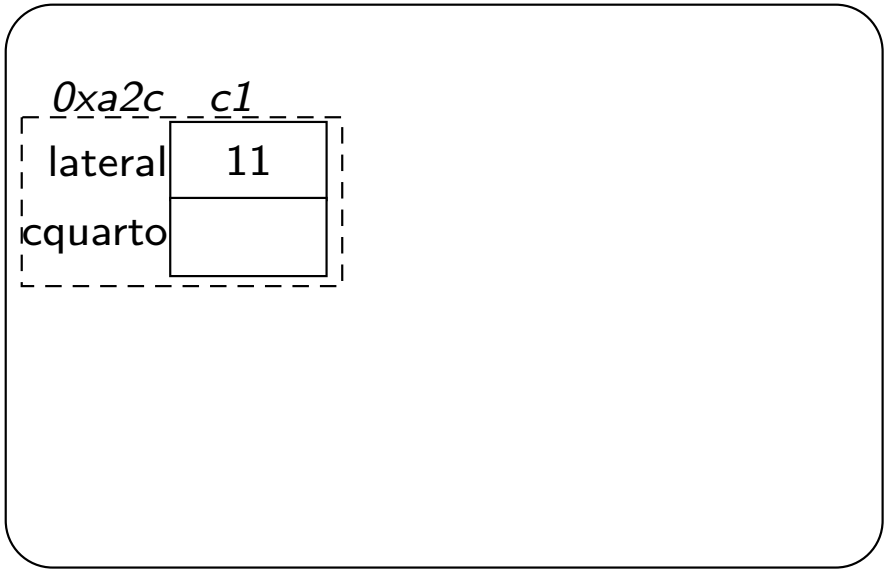

```
int main() {
  casa c1;
  c1.lateral = 11;
  c1.cquarto = 15;

  piscina p1;
  p1.raio = 3.5;

  cabana cab1;
  cab1.cas = c1;
  cab1.pis = p1;
  cab1.cas.lateral = 12;

  printf("A lateral da casa da cabana1
      vale: %.2f\n", cab1.cas.lateral);
  printf("O raio da piscina da cabana1
         vale: %.2f\n", cab1.pis.raio);
  return 0;
}
```

Estruturas e Memória



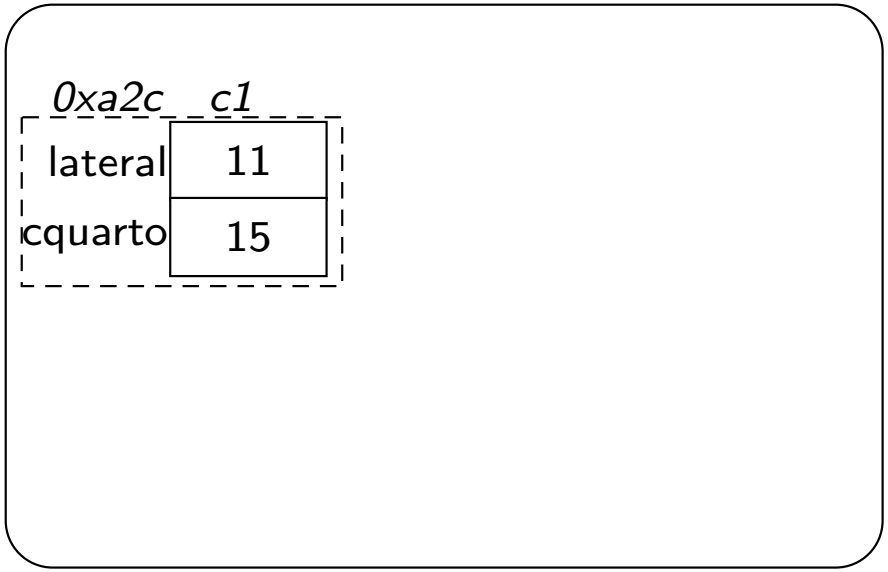

```
int main() {
  casa c1;
  c1.lateral = 11;
  c1.cquarto = 15;

  piscina p1;
  p1.raio = 3.5;

  cabana cab1;
  cab1.cas = c1;
  cab1.pis = p1;
  cab1.cas.lateral = 12;

  printf("A lateral da casa da cabana1
      vale: %.2f\n", cab1.cas.lateral);
  printf("O raio da piscina da cabana1
         vale: %.2f\n", cab1.pis.raio);
  return 0;
}
```

Estruturas e Memória



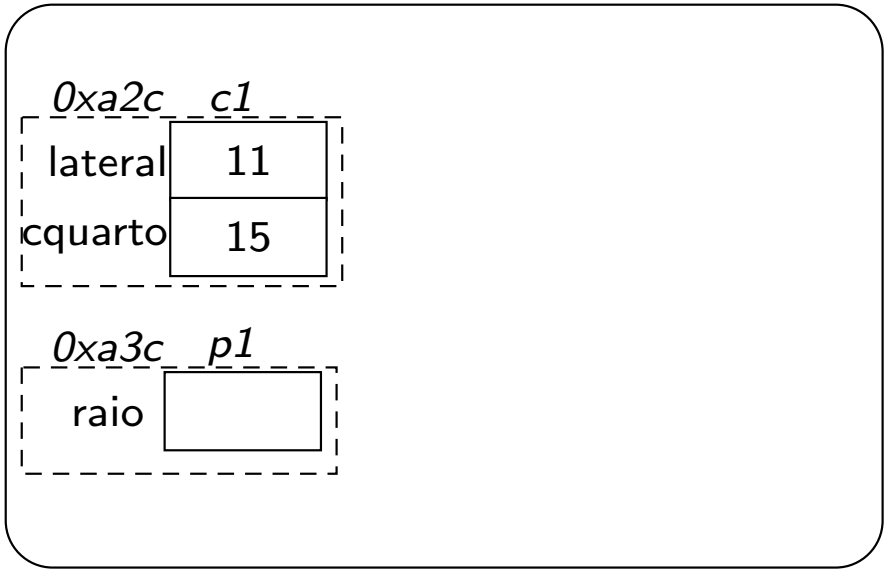

```
int main() {
  casa c1;
  c1.lateral = 11;
  c1.cquarto = 15;

  piscina p1;
  p1.raio = 3.5;

  cabana cab1;
  cab1.cas = c1;
  cab1.pis = p1;
  cab1.cas.lateral = 12;

  printf("A lateral da casa da cabana1
      vale: %.2f\n", cab1.cas.lateral);
  printf("O raio da piscina da cabana1
         vale: %.2f\n", cab1.pis.raio);
  return 0;
}
```

Estruturas e Memória



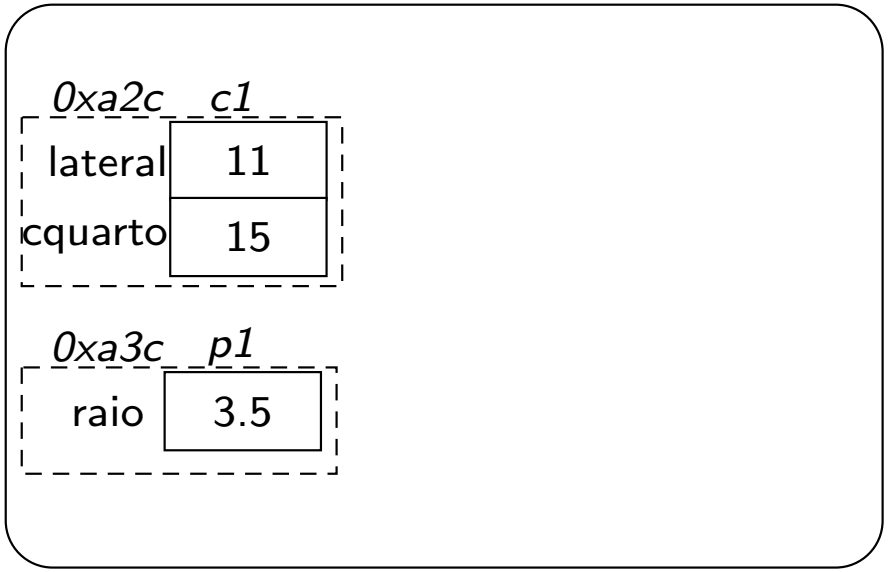

```
int main() {
  casa c1;
  c1.lateral = 11;
  c1.cquarto = 15;

  piscina p1;
  p1.raio = 3.5;

  cabana cab1;
  cab1.cas = c1;
  cab1.pis = p1;
  cab1.cas.lateral = 12;

  printf("A lateral da casa da cabana1
      vale: %.2f\n", cab1.cas.lateral);
  printf("O raio da piscina da cabana1
         vale: %.2f\n", cab1.pis.raio);
  return 0;
}
```

Estruturas e Memória



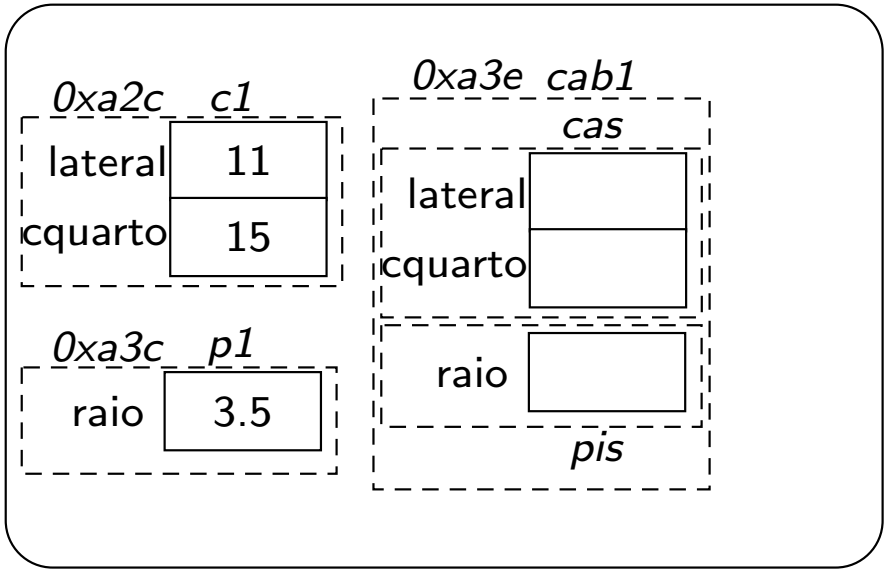

```
int main() {
  casa c1;
  c1.lateral = 11;
  c1.cquarto = 15;

  piscina p1;
  p1.raio = 3.5;

  cabana cab1;
  cab1.cas = c1;
  cab1.pis = p1;
  cab1.cas.lateral = 12;

  printf("A lateral da casa da cabana1
      vale: %.2f\n", cab1.cas.lateral);
  printf("O raio da piscina da cabana1
         vale: %.2f\n", cab1.pis.raio);
  return 0;
}
```

Estruturas e Memória



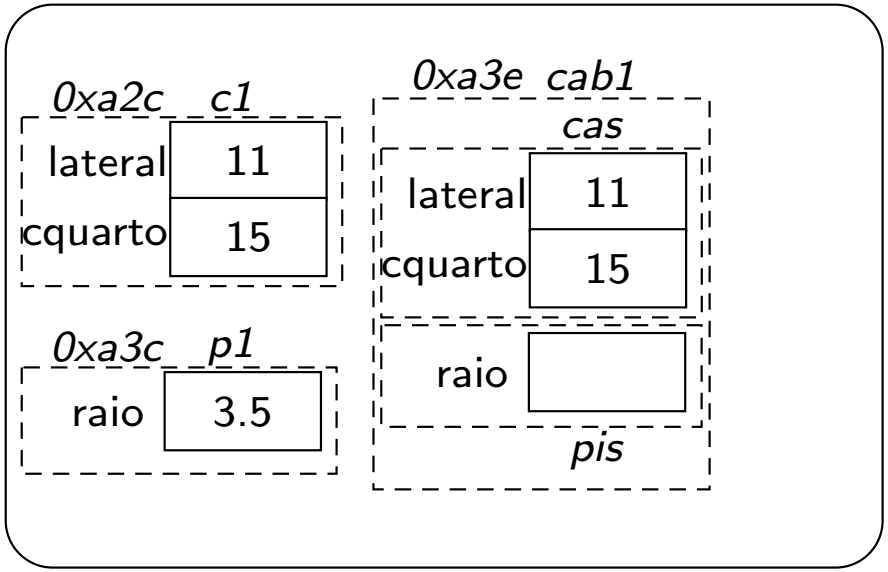

```
int main() {
  casa c1;
  c1.lateral = 11;
  c1.cquarto = 15;

  piscina p1;
  p1.raio = 3.5;

  cabana cab1;
  cab1.cas = c1;
  cab1.pis = p1;
  cab1.cas.lateral = 12;

  printf("A lateral da casa da cabana1
      vale: %.2f\n", cab1.cas.lateral);
  printf("O raio da piscina da cabana1
        vale: %.2f\n", cab1.pis.raio);
  return 0;
}
```

Estruturas e Memória



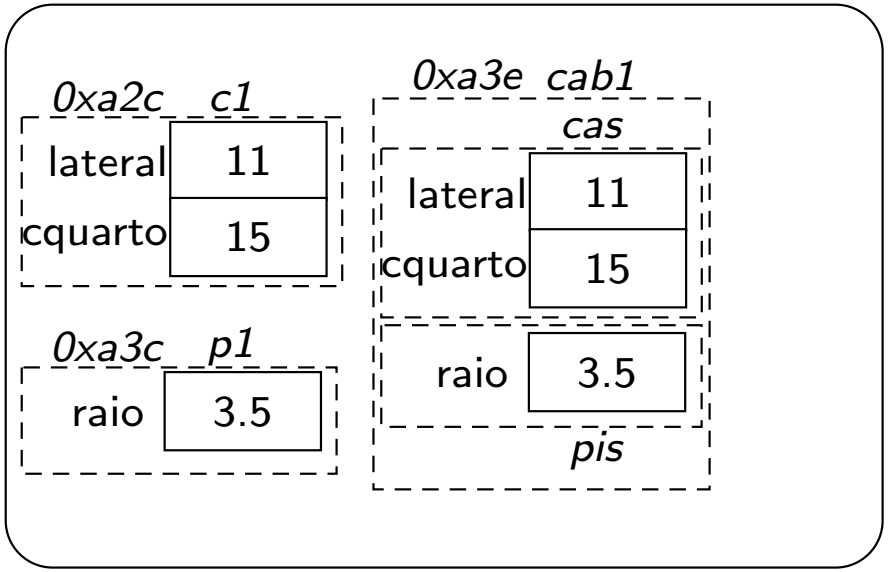

```
int main() {
  casa c1;
  c1.lateral = 11;
  c1.cquarto = 15;

  piscina p1;
  p1.raio = 3.5;

  cabana cab1;
  cab1.cas = c1;
  cab1.pis = p1;
  cab1.cas.lateral = 12;

  printf("A lateral da casa da cabana1
      vale: %.2f\n", cab1.cas.lateral);
  printf("O raio da piscina da cabana1
         vale: %.2f\n", cab1.pis.raio);
  return 0;
}
```

Estruturas e Memória



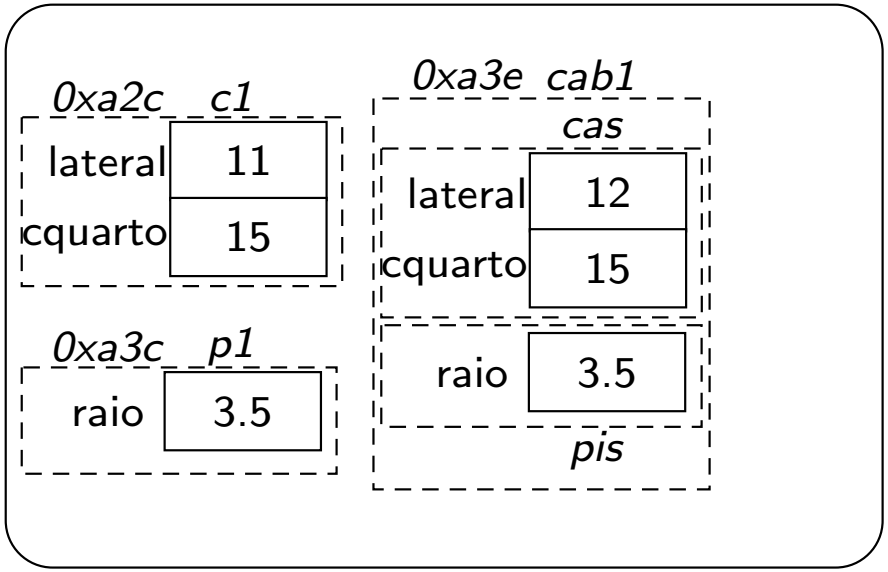

```
int main() {
  casa c1;
  c1.lateral = 11;
  c1.cquarto = 15;

  piscina p1;
  p1.raio = 3.5;

  cabana cab1;
  cab1.cas = c1;
  cab1.pis = p1;
  cab1.cas.lateral = 12;

  printf("A lateral da casa da cabana1
      vale: %.2f\n", cab1.cas.lateral);
  printf("O raio da piscina da cabana1
        vale: %.2f\n", cab1.pis.raio);
  return 0;
}
```

Estruturas e Memória



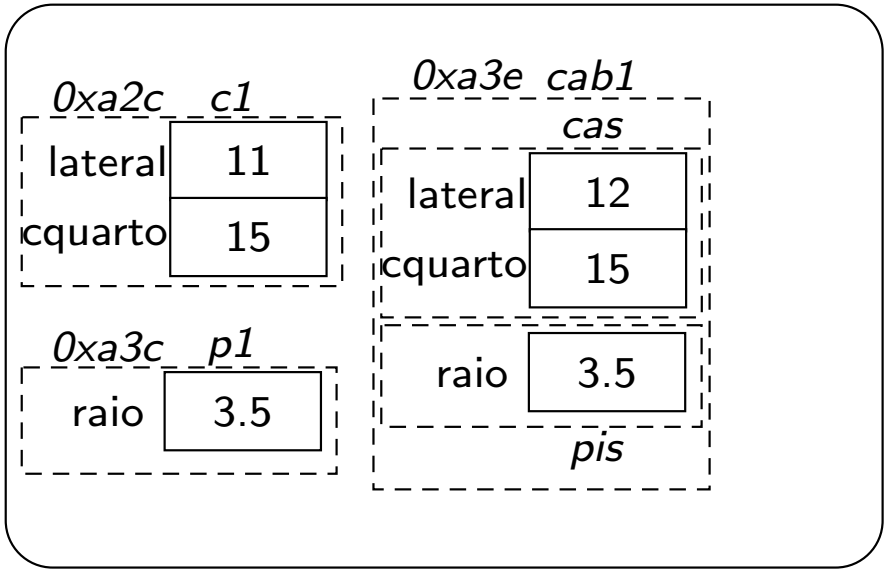

```
int main() {
  casa c1;
  c1.lateral = 11;
  c1.cquarto = 15;

  piscina p1;
  p1.raio = 3.5;

  cabana cab1;
  cab1.cas = c1;
  cab1.pis = p1;
  cab1.cas.lateral = 12;

  printf("A lateral da casa da cabana1
      vale: %.2f\n", cab1.cas.lateral);
  printf("O raio da piscina da cabana1
        vale: %.2f\n", cab1.pis.raio);
  return 0;
}
```

Estruturas e Memória




```
int main() {
  casa c1;
  c1.lateral = 11;
  c1.cquarto = 15;

  piscina p1;
  p1.raio = 3.5;

  cabana cab1;
  cab1.cas = c1;
  cab1.pis = p1;
  cab1.cas.lateral = 12;

  printf("A lateral da casa da cabana1
      vale: %.2f\n", cab1.cas.lateral);
  printf("O raio da piscina da cabana1
        vale: %.2f\n", cab1.pis.raio);
  return 0;
}
```

Saída:

```
A lateral da casa da cabana1 vale: 12.00
O raio da piscina da cabana1 vale: 3.50
```

Estruturas de Dados

- Estruturas podem possuir em seus campos referências a variáveis/espaços de memória de seu próprio tipo

Estruturas de Dados

- Estruturas podem possuir em seus campos referências a variáveis/espaços de memória de seu próprio tipo

```
typedef struct auxP {
    int cpf;
    struct auxP* conjuge;
} Pessoa;
```

Estruturas de Dados

- Estruturas podem possuir em seus campos referências a variáveis/espaços de memória de seu próprio tipo

```
typedef struct auxP {
    int cpf;
    struct auxP* conjuge;
} Pessoa;
```

Estruturas de Dados

- Estruturas podem possuir em seus campos referências a variáveis/espaços de memória de seu próprio tipo

```
typedef struct auxP {
    int cpf;
    struct auxP* conjuge;
} Pessoa;
```

- `conjuge` é do tipo referência (endereço de memória) para uma `Pessoa`

Estruturas de Dados

```
#include <stdlib.h>
typedef struct auxP {
  int cpf;
  struct auxP* conjuge;
} Pessoa;

int main() {
  Pessoa p1;
  p1.cpf = 12345;

  Pessoa* p2 = (Pessoa*)
             malloc(sizeof(Pessoa));
  p2->cpf = 56789;

  Pessoa p3;
  p3.cpf = 44444;

  p1.conjuge = p2;
  p2->conjuge = &p1;
  p3.conjuge = NULL;

  return 0;
}
```

Estruturas de Dados

```
#include <stdlib.h>
typedef struct auxP {
  int cpf;
  struct auxP* conjuge;
} Pessoa;

int main() {
  Pessoa p1;
  p1.cpf = 12345;

  Pessoa* p2 = (Pessoa*)
             malloc(sizeof(Pessoa));
  p2->cpf = 56789;

  Pessoa p3;
  p3.cpf = 44444;

  p1.conjuge = p2;
  p2->conjuge = &p1;
  p3.conjuge = NULL;

  return 0;
}
```

- p1 é do tipo Pessoa

Estruturas de Dados

```
#include <stdlib.h>
typedef struct auxP {
  int cpf;
  struct auxP* conjuge;
} Pessoa;

int main() {
  Pessoa p1;
  p1.cpf = 12345;

  Pessoa* p2 = (Pessoa*)
             malloc(sizeof(Pessoa));
  p2->cpf = 56789;

  Pessoa p3;
  p3.cpf = 44444;

  p1.conjuge = p2;
  p2->conjuge = &p1;
  p3.conjuge = NULL;

  return 0;
}
```

- p1 é do tipo Pessoa

Estruturas de Dados

```
#include <stdlib.h>
typedef struct auxP {
  int cpf;
  struct auxP* conjuge;
} Pessoa;

int main() {
  Pessoa p1;
  p1.cpf = 12345;

  Pessoa* p2 = (Pessoa*)
             malloc(sizeof(Pessoa));
  p2->cpf = 56789;

  Pessoa p3;
  p3.cpf = 44444;

  p1.conjuge = p2;
  p2->conjuge = &p1;
  p3.conjuge = NULL;

  return 0;
}
```

- p1 é do tipo Pessoa, o campo cpf é acessado usando um ponto (.)
- p2 é do tipo ponteiro/referência para Pessoa, aponta para uma memória alocada pela função `malloc`

Estruturas de Dados

```
#include <stdlib.h>
typedef struct auxP {
  int cpf;
  struct auxP* conjuge;
} Pessoa;

int main() {
  Pessoa p1;
  p1.cpf = 12345;

  Pessoa* p2 = (Pessoa*)
             malloc(sizeof(Pessoa));
  p2->cpf = 56789;

  Pessoa p3;
  p3.cpf = 44444;

  p1.conjuge = p2;
  p2->conjuge = &p1;
  p3.conjuge = NULL;

  return 0;
}
```

- p1 é do tipo Pessoa, o campo cpf é acessado usando um ponto (.)
- p2 é do tipo ponteiro/referência para Pessoa, aponta para uma memória alocada pela função malloc, o campo cpf é acessado usando uma seta (->)

Estruturas de Dados

```
#include <stdlib.h>
typedef struct auxP {
  int cpf;
  struct auxP* conjuge;
} Pessoa;

int main() {
  Pessoa p1;
  p1.cpf = 12345;

  Pessoa* p2 = (Pessoa*)
             malloc(sizeof(Pessoa));
  p2->cpf = 56789;

  Pessoa p3;
  p3.cpf = 44444;

  p1.conjuge = p2;
  p2->conjuge = &p1;
  p3.conjuge = NULL;

  return 0;
}
```

- p1 é do tipo Pessoa, o campo cpf é acessado usando um ponto (.)
- p2 é do tipo ponteiro/referência para Pessoa, aponta para uma memória alocada pela função malloc, o campo cpf é acessado usando uma seta (->)
- O campo conjuge de p1 recebe o valor de p2 (que contém um endereço de memória)

Estruturas de Dados

```
#include <stdlib.h>
typedef struct auxP {
  int cpf;
  struct auxP* conjuge;
} Pessoa;

int main() {
  Pessoa p1;
  p1.cpf = 12345;

  Pessoa* p2 = (Pessoa*)
             malloc(sizeof(Pessoa));
  p2->cpf = 56789;

  Pessoa p3;
  p3.cpf = 44444;

  p1.conjuge = p2;
  p2->conjuge = &p1;
  p3.conjuge = NULL;

  return 0;
}
```

- p1 é do tipo Pessoa, o campo cpf é acessado usando um ponto (.)
- p2 é do tipo ponteiro/referência para Pessoa, aponta para uma memória alocada pela função malloc, o campo cpf é acessado usando uma seta (->)
- O campo conjuge de p1 recebe o valor de p2 (que contém um endereço de memória)
- O campo conjuge referenciado por p2 recebe o valor do endereço de p1 (&p1)

Estruturas de Dados

```c
#include <stdlib.h>
typedef struct auxP {
  int cpf;
  struct auxP* conjuge;
} Pessoa;

int main() {
  Pessoa p1;
  p1.cpf = 12345;

  Pessoa* p2 = (Pessoa*)
             malloc(sizeof(Pessoa));
  p2->cpf = 56789;

  Pessoa p3;
  p3.cpf = 44444;

  p1.conjuge = p2;
  p2->conjuge = &p1;
  p3.conjuge = NULL;

  return 0;
}
```

- p1 é do tipo Pessoa, o campo cpf é acessado usando um ponto (.)
- p2 é do tipo ponteiro/referência para Pessoa, aponta para uma memória alocada pela função malloc, o campo cpf é acessado usando uma seta (->)
- O campo conjuge de p1 recebe o valor de p2 (que contém um endereço de memória)
- O campo conjuge referenciado por p2 recebe o valor do endereço de p1 (&p1)
- O campo conjuge de p3 recebe NULL (endereço nulo)
  - NULL não é uma palavra reservada em C, mas é uma constante presente na biblioteca stdlib

Aula 25 – Estruturas (parte 1)

Norton T. Roman & Luciano A. Digiampietri