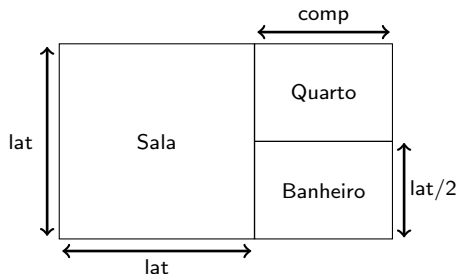
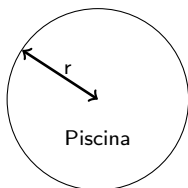


Aula 08 – Funções/Subrotinas

Norton T. Roman & Luciano A. Digiampietri

- Estamos incrementando nossa cabana com uma piscina:



Constantes

- Estamos implementando um programa que calcula a área da piscina.

```
#include <stdio.h>
#include <math.h>
int main() {
    // raio da piscina
    double raio = 2;
    // área da piscina
    double areap;
    areap = M_PI * raio * raio;
    printf("Área: %f\n", areap);

    return 0;
}
```

Constantes

- E como podemos mudar $\text{raio} * \text{raio}$?

```
#include <stdio.h>
#include <math.h>
int main() {
    // raio da piscina
    double raio = 2;
    // área da piscina
    double areap;
    areap = M_PI * pow(raio,2);
    printf("Área: %f\n", areap);

    return 0;
}
```

Constantes

- E como podemos mudar `raio * raio`?
- `pow(a,b)` dá o resultado de a^b
- O resultado também é double

```
#include <stdio.h>
#include <math.h>
int main() {
    // raio da piscina
    double raio = 2;
    // área da piscina
    double areap;
    areap = M_PI * pow(raio,2);
    printf("Área: %f\n", areap);

    return 0;
}
```

- math é uma biblioteca, que nos fornece a função pow (além da constante M_PI)

```
#include <stdio.h>
#include <math.h>
int main() {
    // raio da piscina
    double raio = 2;
    // área da piscina
    double areap;
    areap = M_PI * pow(raio,2);
    printf("Área: %f\n", areap);

    return 0;
}
```

- math é uma biblioteca, que nos fornece a função pow (além da constante M_PI)
- Função?

```
#include <stdio.h>
#include <math.h>
int main() {
    // raio da piscina
    double raio = 2;
    // área da piscina
    double areap;
    areap = M_PI * pow(raio,2);
    printf("Área: %f\n", areap);

    return 0;
}
```

Funções

- Uma **função** é uma implementação de uma subrotina
- Nesse caso, `pow(a,b)` recebe dois valores, `a` e `b`, devolvendo o resultado de a^b
- Os valores `a` e `b` fornecidos à função são chamados **argumentos** de seus **parâmetros**

```
#include <stdio.h>
#include <math.h>
int main() {
    // raio da piscina
    double raio = 2;
    // área da piscina
    double areap;
    areap = M_PI * pow(raio,2);
    printf("Área: %f\n", areap);

    return 0;
}
```


Funções

Vamos então juntar os dois programas que vimos até agora em um só:

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

int main {
    float lateral = 11;
    float quarto = 7;
    float areaq;
    float areas;
    float areat;
    double raio = 2;
    double areap;

    printf("Programa para cálculo da área da casa\n");
    areas = lateral*lateral;
    printf("A área da sala é %f\n", areas);
    areaq = quarto*(lateral/2);
    printf("A área do quarto é %f\n", areaq);
    printf("A área do banheiro é %f\n", areaq);
    areat = areas + 2*areaq;
    printf("A área total é %f\n", areat);
    areap = M_PI * pow(raio,2);
    printf("A área da piscina é %f\n", areap);
    return 0;
}
```

E qual a saída?

E qual a saída?

```
Programa para cálculo da área da casa
```

```
A área da sala é 121.000000
```

```
A área do quarto é 38.500000
```

```
A área do banheiro é 38.500000
```

```
A área total é 198.000000
```

```
A área da piscina é 12.566371
```

- Esse programa está ficando confuso:
 - Mistura a casa com a piscina

- Esse programa está ficando confuso:
 - Mistura a casa com a piscina
- Que fazer?

- Esse programa está ficando confuso:
 - Mistura a casa com a piscina
- Que fazer?
 - Podemos dividi-lo em 2 partes: uma para o cálculo da casa e outra para o cálculo da piscina

- Esse programa está ficando confuso:
 - Mistura a casa com a piscina
- Que fazer?
 - Podemos dividi-lo em 2 partes: uma para o cálculo da casa e outra para o cálculo da piscina
- Como?

- Esse programa está ficando confuso:
 - Mistura a casa com a piscina
- Que fazer?
 - Podemos dividi-lo em 2 partes: uma para o cálculo da casa e outra para o cálculo da piscina
- Como?
 - Criando nossas próprias funções

Funções

```
void areaCasa(){
    float lateral = 11;
    float cquarto = 7;
    float areaq;
    float areas;
    float areat;
    printf("Programa para cálculo da
           área da casa\n");
    areas = lateral*lateral;
    printf("A área da sala é %f\n", areas);
    areaq = cquarto*(lateral/2);
    printf("A área do quarto é %f\n", areaq);
    printf("A área do banheiro é %f\n", areaq);
    areat = areas + 2*areaq;
    printf("A área total é %f\n", areat);
}
```

```
double areaPiscina(){
    double raio = 2;
    return M_PI * pow(raio,2);
}
```

Funções

```
void areaCasa(){
    float lateral = 11;
    float cquarto = 7;
    float areaq;
    float areas;
    float areat;
    printf("Programa para cálculo da
           área da casa\n");
    areas = lateral*lateral;
    printf("A área da sala é %f\n", areas);
    areaq = cquarto*(lateral/2);
    printf("A área do quarto é %f\n", areaq);
    printf("A área do banheiro é %f\n", areaq);
    areat = areas + 2*areaq;
    printf("A área total é %f\n", areat);
}
```

```
double areaPiscina(){
    double raio = 2;
    return M_PI * pow(raio,2);
}
```

Ambas dentro do
mesmo programa...

- O que significa o void?

```
void areaCasa() {  
    float lateral = 11;  
    float cquarto = 7;  
    float areaq;  
    float areas;  
    float areat;  
    printf("Programa para cálculo da  
           área da casa\n");  
    areas = lateral*lateral;  
    printf("A área da sala é %f\n", areas);  
    areaq = cquarto*(lateral/2);  
    printf("A área do quarto é %f\n", areaq);  
    printf("A área do banheiro é %f\n", areaq);  
    areat = areas + 2*areaq;  
    printf("A área total é %f\n", areat);  
}
```

- O que significa o void?
- Que a função não irá retornar nenhum valor
- Ela apenas executa a tarefa e termina

```
void areaCasa() {  
    float lateral = 11;  
    float cquarto = 7;  
    float areaq;  
    float areas;  
    float areat;  
    printf("Programa para cálculo da  
           área da casa\n");  
    areas = lateral*lateral;  
    printf("A área da sala é %f\n", areas);  
    areaq = cquarto*(lateral/2);  
    printf("A área do quarto é %f\n", areaq);  
    printf("A área do banheiro é %f\n", areaq);  
    areat = areas + 2*areaq;  
    printf("A área total é %f\n", areat);  
}
```

Funções

- O que significa o *double*?

```
double areaPiscina() {  
    double raio = 2;  
    return(M_PI * pow(raio,2));  
}
```

Funções

- O que significa o *double*?
- Que a função irá retornar um valor do tipo *double*

```
double areaPiscina() {  
    double raio = 2;  
    return(M_PI * pow(raio,2));  
}
```

Funções

- O que significa o *double*?
 - Que a função irá retornar um valor do tipo *double*
 - Semelhante à função `pow(a,b)`

```
double areaPiscina() {  
    double raio = 2;  
    return(M_PI * pow(raio,2));  
}
```

Funções

- O que significa o *double*?
- Que a função irá retornar um valor do tipo *double*
- Semelhante à função `pow(a,b)`
- E o `return`?

```
double areaPiscina() {  
    double raio = 2;  
    return(M_PI * pow(raio,2));  
}
```


Funções

- O que significa o *double*?

- Que a função irá retornar um valor do tipo *double*

- Semelhante à função `pow(a,b)`

```
double areaPiscina() {  
    double raio = 2;  
    return(M_PI * pow(raio,2));  
}
```

- E o `return`?

- É quando o valor é efetivamente retornado

- A função/subrotina para aí

- Alternativas:

- `return(M_PI * pow(raio,2));`
- `return M_PI * pow(raio,2);`

Funções

- E como usamos isso no corpo do programa?

- E como usamos isso no corpo do programa?

```
int main() {  
    double areap;  
    areaCasa();  
  
    areap = areaPiscina();  
    printf("A área da piscina  
        é %f\n", areap);  
  
    return 0;  
}
```

- E como usamos isso no corpo do programa?
- Note que `areaPiscina()` retorna valor, então guardamos esse valor em `areap`

```
int main() {  
    double areap;  
    areaCasa();  
  
    areap = areaPiscina();  
    printf("A área da piscina  
        é %f\n", areap);  
  
    return 0;  
}
```

- E como usamos isso no corpo do programa?
- Note que `areaPiscina()` retorna valor, então guardamos esse valor em `areap`
- Já `areaCasa()` não retorna nada, então apenas a executamos

```
int main() {  
    double areap;  
    areaCasa();  
  
    areap = areaPiscina();  
    printf("A área da piscina  
        é %f\n", areap);  
  
    return 0;  
}
```

Visão Geral do Código

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

void areaCasa(){
    float lateral = 11;
    float quarto = 7;
    float areaq;
    float areas;
    float areat;
    printf("Programa para ...\n");
    areas = lateral*lateral;
    printf("A área ... %f\n", areas);
    areaq = quarto*(lateral/2);
    printf("A área ... %f\n", areaq);
    printf("A área ... %f\n", areaq);
    areat = areas + 2*areaq;
    printf("A área ... %f\n", areat);
}

double areaPiscina(){
    double raio = 2;
    return M_PI * pow(raio,2);
}

int main() {
    double areap;
    areaCasa();

    areap = areaPiscina();
    printf("A área da piscina
           é %f\n", areap);

    return 0;
}
```

- Qual a utilidade de criarmos nossas próprias funções?

```
int main() {
    double areap;

    areaCasa();
    areap = areaPiscina();
    printf("A área da piscina
           é %f\n",areap);
    return 0;
}
```

Funções

- Qual a utilidade de criarmos nossas próprias funções?
- Clareza: ao olharmos o corpo do programa, vemos claramente o que é feito, sem nos preocuparmos com detalhes
- A função top-down fica clara

```
int main() {  
    double areap;  
  
    areaCasa();  
    areap = areaPiscina();  
    printf("A área da piscina  
           é %f\n",areap);  
    return 0;  
}
```


Funções

- Qual a utilidade de criarmos nossas próprias funções?
- Clareza: ao olharmos o corpo do programa, vemos claramente o que é feito, sem nos preocuparmos com detalhes
 - A função top-down fica clara
- Portabilidade: se precisarmos, em outro programa, usar a mesma subrotina, ela já está separada

```
int main() {  
    double areap;  
  
    areaCasa();  
    areap = areaPiscina();  
    printf("A área da piscina  
           é %f\n",areap);  
    return 0;  
}
```

- Nossas funções, contudo, não são gerais:

```
int main() {
    double areap;
    areaCasa();

    areap = areaPiscina();
    printf("A área da piscina
           é %f\n", areap);

    return 0;
}
```

- Nossas funções, contudo, não são gerais:

- `areaCasa()` funciona apenas para casas da dimensão de nosso projeto

```
int main() {  
    double areap;  
    areaCasa();  
  
    areap = areaPiscina();  
    printf("A área da piscina  
          é %f\n", areap);  
  
    return 0;  
}
```

- Nossas funções, contudo, não são gerais:

- `areaCasa()` funciona apenas para casas da dimensão de nosso projeto
- `areaPiscina()` funciona apenas para piscinas redondas de raio 2

```
int main() {  
    double areap;  
    areaCasa();  
  
    areap = areaPiscina();  
    printf("A área da piscina  
           é %f\n", areap);  
  
    return 0;  
}
```

- Nossas funções, contudo, não são gerais:

- `areaCasa()` funciona apenas para casas da dimensão de nosso projeto
- `areaPiscina()` funciona apenas para piscinas redondas de raio 2

```
int main() {  
    double areap;  
    areaCasa();  
  
    areap = areaPiscina();  
    printf("A área da piscina  
          é %f\n", areap);  
  
    return 0;  
}
```

- Como poderíamos fazer para tornar essas funções mais gerais?

Parâmetros

- A ideia é manter o formato da casa e da piscina, mas permitir que seu tamanho varie
- Como fazê-lo?

Parâmetros

- A ideia é manter o formato da casa e da piscina, mas permitir que seu tamanho varie
- Como fazê-lo? Com parâmetros:

```
double areaPiscina(double raio){  
    return(M_PI * pow(raio,2));  
}
```

Parâmetros

- A ideia é manter o formato da casa e da piscina, mas permitir que seu tamanho varie
- Como fazê-lo? Com parâmetros:

```
double areaPiscina(double raio){  
    return(M_PI * pow(raio,2));  
}
```

- A função agora deve receber um valor (argumento) em seu parâmetro
 - Como o pow

Parâmetros

- Como chamamos essa função de outras partes do programa?

Parâmetros

- Como chamamos essa função de outras partes do programa?

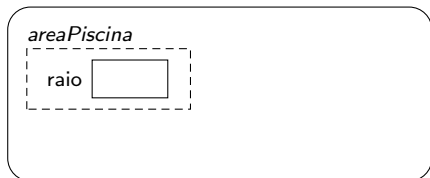
```
int main() {  
    double areap;  
    areaCasa();  
  
    areap = areaPiscina(2);  
    printf("A área da piscina  
           é %f\n", areap);  
  
    return 0;  
}
```

Parâmetros

- E o que acontece ao chamarmos `areaPiscina(2)` de dentro do `main`?

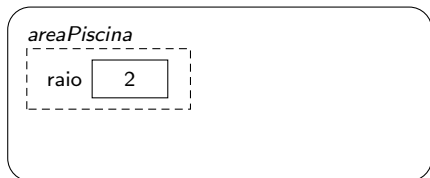
Parâmetros

- E o que acontece ao chamarmos `areaPiscina(2)` de dentro do main?
- O sistema irá alocar memória para todas as variáveis e parâmetros declarados dentro da função



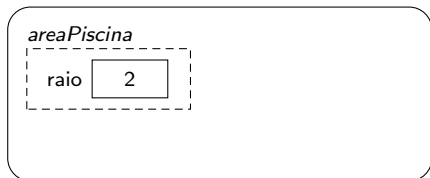
Parâmetros

- E o que acontece ao chamarmos `areaPiscina(2)` de dentro do main?
- O sistema irá alocar memória para todas as variáveis e parâmetros declarados dentro da função
- Colocando o valor passado como parâmetro lá



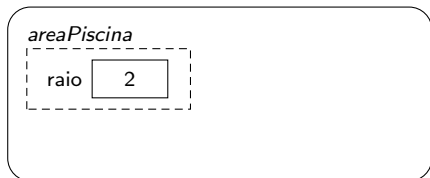
Parâmetros

- E o que acontece ao chamarmos `areaPiscina(2)` de dentro do main?
- O sistema irá alocar memória para todas as variáveis e parâmetros declarados dentro da função
- Colocando o valor passado como parâmetro lá
- Ao ato de passar um valor externo para dentro de um procedimento, via parâmetro, chamamos de **passagem por valor**



Parâmetros

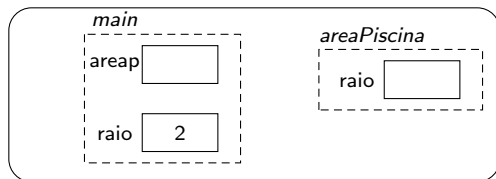
- E o que acontece ao chamarmos `areaPiscina(2)` de dentro do main?
- O sistema irá alocar memória para todas as variáveis e parâmetros declarados dentro da função
- Colocando o valor passado como parâmetro lá
- Ao ato de passar um valor externo para dentro de um procedimento, via parâmetro, chamamos de **passagem por valor**
- Nesse caso, o valor externo é copiado para a região de memória correspondente ao parâmetro



Parâmetros

- O que acontece se tivermos algo assim?

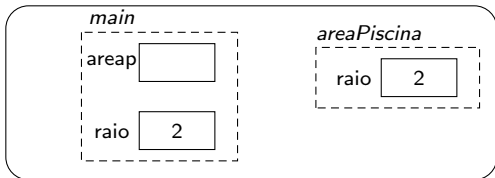
```
int main() {  
    double areap;  
    double raio = 2;  
  
    areaCasa();  
  
    areap = areaPiscina(raio);  
    printf("A área da  
           piscina é %f\n",areap);  
    return 0;  
}
```



Parâmetros

- O que acontece se tivermos algo assim?
- O valor de raio, em main, é copiado para dentro da variável raio em areaPiscina

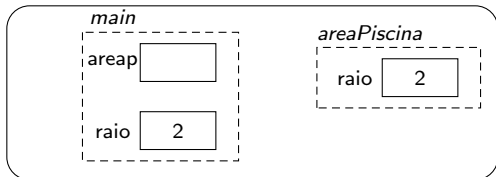
```
int main() {  
    double areap;  
    double raio = 2;  
  
    areaCasa();  
  
    areap = areaPiscina(raio);  
    printf("A área da  
           piscina é %f\n",areap);  
    return 0;  
}
```



Parâmetros

- O que acontece se tivermos algo assim?
- O valor de raio, em main, é copiado para dentro da variável raio em areaPiscina
- São duas regiões de memória diferentes

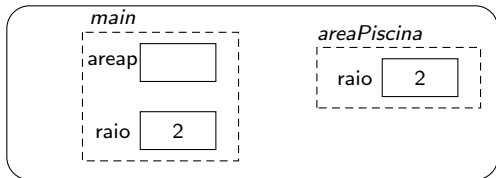
```
int main() {  
    double areap;  
    double raio = 2;  
  
    areaCasa();  
  
    areap = areaPiscina(raio);  
    printf("A área da  
           piscina é %f\n",areap);  
    return 0;  
}
```



Parâmetros

- O que acontece se tivermos algo assim?
- O valor de raio, em main, é copiado para dentro da variável raio em areaPiscina
- São duas regiões de memória diferentes
- Sim... main é uma função também

```
int main() {  
    double areap;  
    double raio = 2;  
  
    areaCasa();  
  
    areap = areaPiscina(raio);  
    printf("A área da  
           piscina é %f\n",areap);  
    return 0;  
}
```



Parâmetros

Incluindo parâmetros em *areaCasa()*:

```
void areaCasa(float lateral,
             float cquarto){
    float areaq;
    float areas;
    float areat;

    printf("Programa para ... \n");
    areas = lateral*lateral;
    printf("A área ... %f \n", areas);
    areaq = cquarto*(lateral/2);
    printf("A área ... %f \n", areaq);
    printf("A área ... %f \n", areaq);
    areat = areas + 2*areaq;
    printf("A área ... %f \n", areat);
}
```

```
double areaPiscina(double raio){
    return M_PI*.pow(raio,2);
}

int main() {
    double areap;
    areaCasa();

    areap = areaPiscina(2);
    printf("A área da piscina
           é %f \n", areap);

    return 0;
}
```

Funções e Memória

- Como fica a função `areaCasa` na memória?

```
void areaCasa(float lateral,
              float quarto) {
    float areaq;
    float areas;
    float areat;

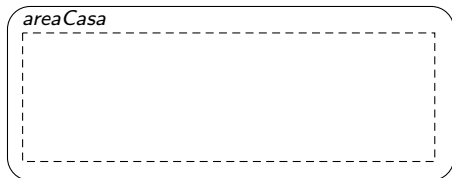
    printf("Programa para ...\n");
    areas = lateral*lateral;
    printf("A área ... %f\n", areas);
    areaq = quarto*(lateral/2);
    printf("A área ... %f\n", areaq);
    printf("A área ... %f\n", areaq);
    areat = areas + 2*areaq;
    printf("A área ... %f\n", areat);
}
```

Funções e Memória

- Como fica a função `areaCasa` na memória?
- Ao ser **chamado** (ou **invocado**) em `main`, será separada uma região na memória para essa função

```
void areaCasa(float lateral,
              float quarto) {
    float areaq;
    float areas;
    float areat;

    printf("Programa para ...\n");
    areas = lateral*lateral;
    printf("A área ... %f\n", areas);
    areaq = quarto*(lateral/2);
    printf("A área ... %f\n", areaq);
    printf("A área ... %f\n", areaq);
    areat = areas + 2*areaq;
    printf("A área ... %f\n", areat);
}
```

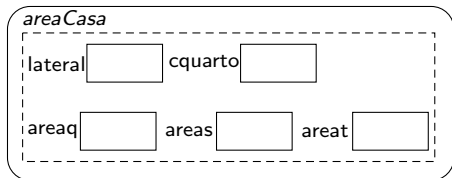


Funções e Memória

- Como fica a função `areaCasa` na memória?
- Ao ser **chamado** (ou **invocado**) em `main`, será separada uma região na memória para essa função
- Essa região conterá todas suas variáveis internas (**locais**), e todos seus parâmetros

```
void areaCasa(float lateral,
              float quarto) {
    float areaq;
    float areas;
    float areat;

    printf("Programa para ...\n");
    areas = lateral*lateral;
    printf("A área ... %f\n", areas);
    areaq = quarto*(lateral/2);
    printf("A área ... %f\n", areaq);
    printf("A área ... %f\n", areaq);
    areat = areas + 2*areaq;
    printf("A área ... %f\n", areat);
}
```

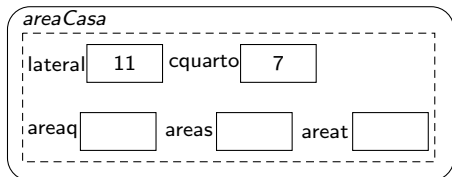


Funções e Memória

```
int main() {  
    ...  
    areaCasa(11,7);  
    ...  
}
```

- Os valores de entrada são então copiados para dentro dos parâmetros

```
void areaCasa(float lateral, float cquarto  
    float areaq;  
    float areas;  
    float areat;  
  
    printf("Programa para ... \n");  
    areas = lateral*lateral;  
    printf("A área ... %f \n", areas);  
    areaq = cquarto*(lateral/2);  
    printf("A área ... %f \n", areaq);  
    printf("A área ... %f \n", areaq);  
    areat = areas + 2*areaq;  
    printf("A área ... %f \n", areat);  
}
```



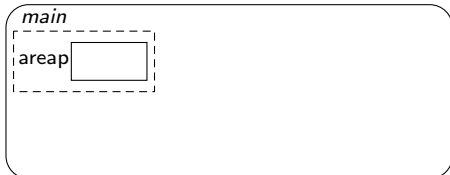
Funções e Memória

- Considerando o programa como um todo, como agirá na memória?

```
int main() {  
    double areap;  
  
    areaCasa(11,7);  
    areap = areaPiscina(2);  
    printf("A área da piscina  
           é %p\n",areap);  
    return 0;  
}
```

Funções e Memória

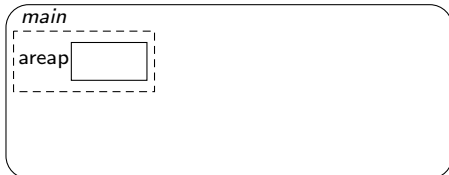
- Considerando o programa como um todo, como agirá na memória?
- Ao iniciar *main*, será alocado espaço para suas variáveis e parâmetros



```
int main() {  
    double areap;  
  
    areaCasa(11,7);  
    areap = areaPiscina(2);  
    printf("A área da piscina  
           é %p\n",areap);  
    return 0;  
}
```

Funções e Memória

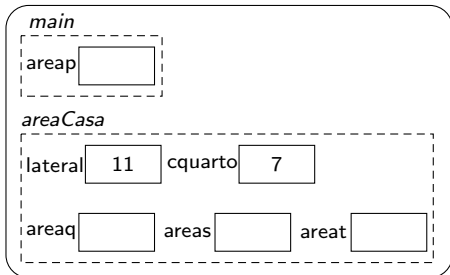
- Considerando o programa como um todo, como agirá na memória?
- Ao iniciar *main*, será alocado espaço para suas variáveis e parâmetros
- Então *areaCasa(11,7)* é executada, e o mesmo processo ocorre



```
int main() {  
    double areap;  
  
    areaCasa(11,7);  
    areap = areaPiscina(2);  
    printf("A área da piscina  
           é %p\n",areap);  
    return 0;  
}
```

Funções e Memória

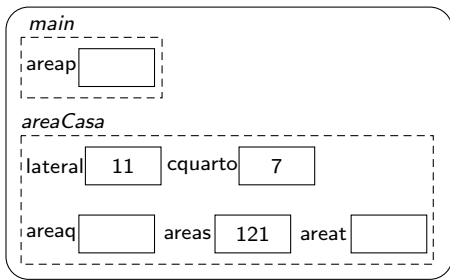
- Aloca-se espaço, copiando-se os valores aos parâmetros:



```
int main() {  
    ...  
    areaCasa(11,7);  
    ...  
}  
  
void areaCasa(float lateral,  
              float cquarto) {  
    float areaq;  
    float areas;  
    float areat;  
  
    printf("Programa para ...\n");  
    areas = lateral*lateral;  
    printf("A área ... %f\n", areas);  
    areaq = cquarto*(lateral/2);  
    printf("A área ... %f\n", areaq);  
    printf("A área ... %f\n", areaq);  
    areat = areas + 2*areaq;  
    printf("A área ... %f\n", areat);  
}
```

Funções e Memória

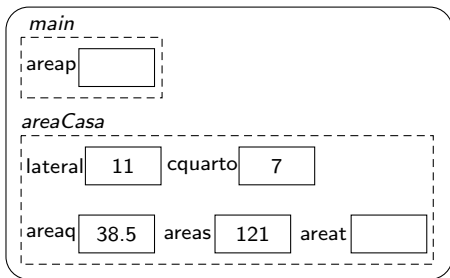
- A cada atribuição, a memória correspondente é atualizada



```
int main( ){  
    ...  
    areaCasa(11,7);  
    ...  
}  
  
void areaCasa(float lateral,  
              float cquarto) {  
    float areaq;  
    float areas;  
    float areat;  
  
    printf("Programa para ...\n");  
    areas = lateral*lateral;  
    printf("A área ... %f\n", areas);  
    areaq = cquarto*(lateral/2);  
    printf("A área ... %f\n", areaq);  
    printf("A área ... %f\n", areaq);  
    areat = areas + 2*areaq;  
    printf("A área ... %f\n", areat);  
}
```

Funções e Memória

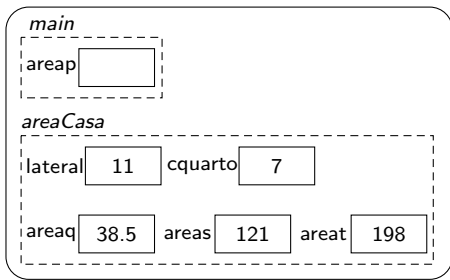
- A cada atribuição, a memória correspondente é atualizada



```
int main( ){  
    ...  
    areaCasa(11,7);  
    ...  
}  
  
void areaCasa(float lateral,  
              float cquarto) {  
    float areaq;  
    float areas;  
    float areat;  
  
    printf("Programa para ...\n");  
    areas = lateral*lateral;  
    printf("A área ... %f\n", areas);  
    areaq = cquarto*(lateral/2);  
    printf("A área ... %f\n", areaq);  
    printf("A área ... %f\n", areaq);  
    areat = areas + 2*areaq;  
    printf("A área ... %f\n", areat);  
}
```

Funções e Memória

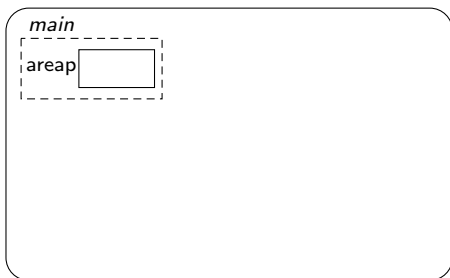
- A cada atribuição, a memória correspondente é atualizada



```
int main( ){  
    ...  
    areaCasa(11,7);  
    ...  
}  
  
void areaCasa(float lateral,  
              float cquarto) {  
    float areaq;  
    float areas;  
    float areat;  
  
    printf("Programa para ...\n");  
    areas = lateral*lateral;  
    printf("A área ... %f\n", areas);  
    areaq = cquarto*(lateral/2);  
    printf("A área ... %f\n", areaq);  
    printf("A área ... %f\n", areaq);  
    areat = areas + 2*areaq;  
    printf("A área ... %f\n", areat);  
}
```

Funções e Memória

- Ao terminar `areaCasa`, sua memória é limpa, e `areaPiscina` é rodada:



```
double areaPiscina(double raio)
{
    return M_PI * pow(raio,2);
}

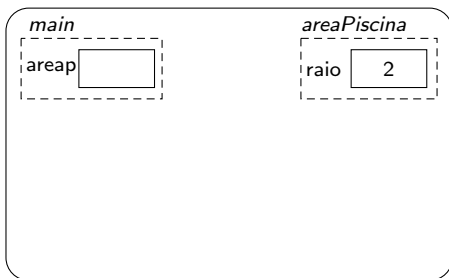
int main() {
    double areap;

    areaCasa(11,7);

    areap = areaPiscina(2);
    printf("A área da piscina
           é %f\n", areap);
    return 0;
}
```


Funções e Memória

- Ao terminar `areaCasa`, sua memória é limpa, e `areaPiscina` é rodada:



```
double areaPiscina(double raio)
{
    return M_PI * pow(raio,2);
}
```

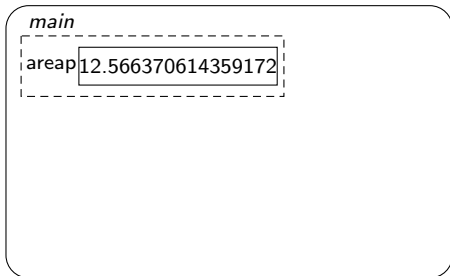
```
int main() {
    double areap;

    areaCasa(11,7);

    areap = areaPiscina(2);
    printf("A área da piscina
           é %f\n", areap);
    return 0;
}
```

Funções e Memória

- Ao terminar `areaPiscina`, sua memória é limpa, e o resultado é armazenado em `areap`:



```
int main() {  
    double areap;  
  
    areaCasa(11,7);  
  
    areap = areaPiscina(2);  
    printf("A área da piscina  
           é %f\n", areap);  
    return 0;  
}
```

Funções e Memória

- Ao terminar `areaPiscina`, sua memória é limpa, e o resultado é armazenado em `areap`:



```
int main() {  
    double areap;  
  
    areaCasa(11,7);  
  
    areap = areaPiscina(2);  
    printf("A área da piscina  
           é %f\n", areap);  
  
    return 0;  
}
```

Finalmente, quando *main* terminar, sua memória também será removida

Funções e Memória

- Repare que toda vez que uma função termina ela libera a memória que ocupava
- Então, qual a utilidade de criarmos nossas próprias funções além de clareza e portabilidade?
- Melhor uso da memória: as variáveis relevantes ao sub-problema (sub-rotina) ocupam a memória apenas durante a solução desse sub-problema

Aula 08 – Funções/Subrotinas

Norton T. Roman & Luciano A. Digiampietri