ALGORITMOS E ESTRUTURAS DE DADOS I

Prof. Caio César de Freitas Dantas

Ponteiros e Funções

- Uma função pode ser vista como um conjunto de comandos que realiza uma tarefa específica. Em outras palavras, pode-se dizer que é um pequeno "programa" utilizado por outros programas.

- A função é referenciada (chamada) pelo programa principal através de um nome atribuído a ela.

- A utilização de funções, muito comum na programação estruturada, visa subdividir um programa em partes (módulos) menores que realizam uma tarefa bem definida.

Ponteiros e Funções

Benefícios da utilização de funções:

- Permite o reaproveitamento de código já construído(por você ou por outros programadores);
- Evita que um mesmo trecho de código seja repetido várias vezes dentro de um mesmo programa e, com isso, qualquer alteração é feita apenas nesse trecho e de forma simples.
- Para que os blocos do programa não fiquem grandes demais e, por consequência, mais difíceis de entender;
- Facilita a leitura do programa de maneira que os blocos de código possam ser logicamente compreendidos de forma isolada.

Ponteiros e Funções

Esqueleto de uma função

```
tipo_de_retorno nome_da_função (lista de parâmetros)
{
   instruções;
   retorno_da_função;
}
```

Ponteiros e Funções

Parâmetros

- A Lista de Parâmetros, também é chamada de Lista de Argumentos. Funcionam como a interface de comunicação (passagem de valores/dados) entre o programa (chamador) e a função.

- Os parâmetros de uma função são definidos como se estivesse declarando uma variável, entre os parênteses do cabeçalho da função. Caso precise declarar mais de um parâmetro, basta separá-los por vírgulas.

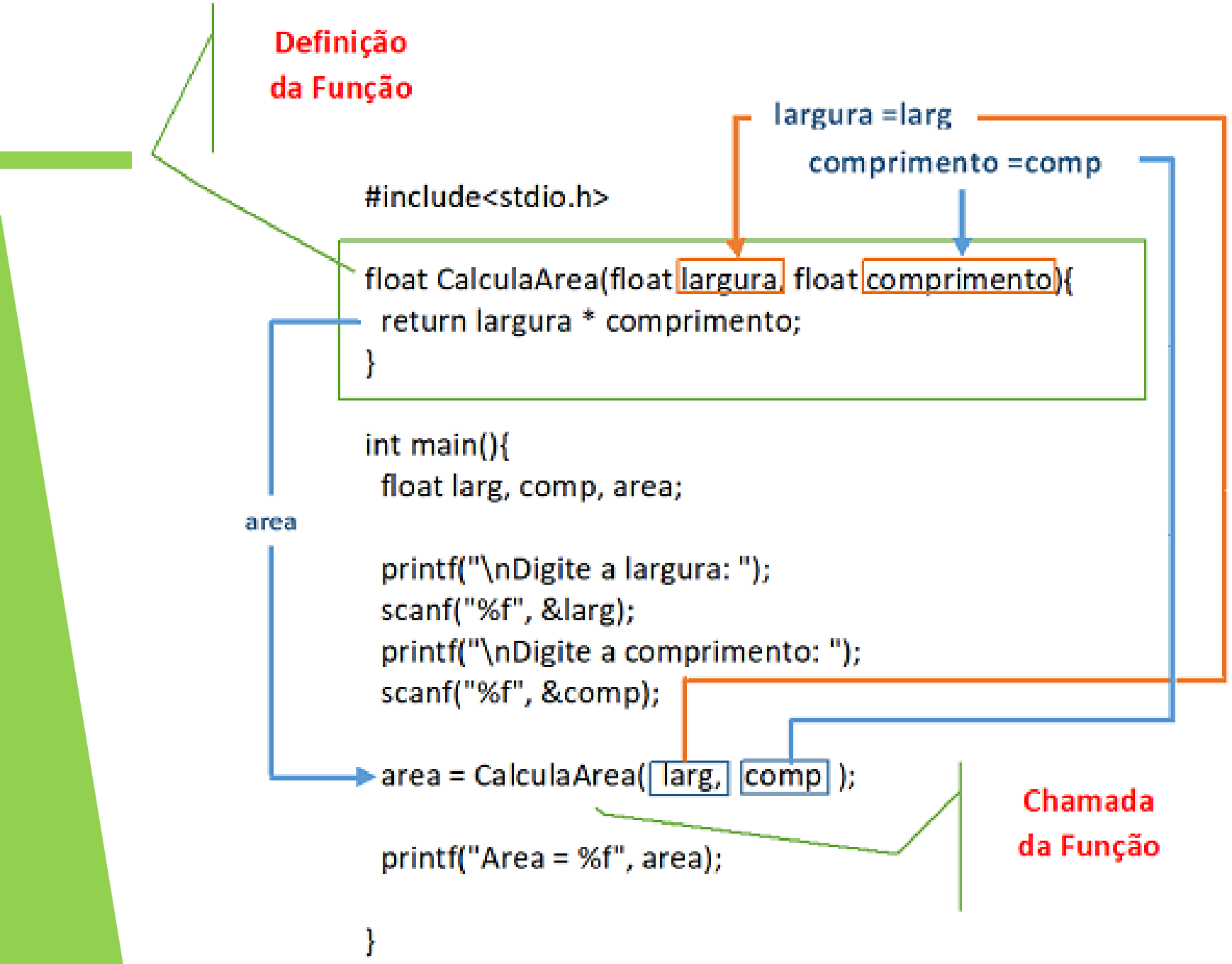
Ponteiros e Funções

Parâmetros

Existem 2 maneiras de passar valor através dos parâmetros: Por valor ou Por referência. Por valor:

- Os parâmetros são passados para uma função de acordo com a sua posição.
  - Ou seja, o primeiro parâmetro da chamada (programa) define o valor do primeiro parâmetro da função, o segundo parâmetro do programa define o valor do segundo parâmetro da função e assim por diante.

- Os nomes dos parâmetros na chamada não tem relação com os nomes dos parâmetros na definição da função.


Definição
da Função
largura =larg
comprimento =comp
#include<stdio.h>
float CalculaArea(float largura, float comprimento){
return largura * comprimento;
}
int main(){
float larg, comp, area;
area
printf("\nDigite a largura: ");
scanf("%f", &larg);
printf("\nDigite a comprimento: ");
scanf("%f", &comp);
area = CalculaArea(larg, comp);
printf("Area = %f", area);
}
Chamada
da Função

Ponteiros e Funções

Eu tenho 2 variáveis e quero trocar o valor delas.

```c
#include <stdio.h>

int main(){
   int a = 5, b = 10, temp;
   printf ("%d %d\n", a, b);

   temp = a;
   a = b;
   b = temp;

   printf ("%d %d\n", a, b);
   return 0;
}
```

Ponteiros e Funções

Eu tenho 2 variáveis e quero trocar o valor delas.

```c
#include <stdio.h>

int main(){
   int a = 5, b = 10, temp;
   printf ("%d %d\n", a, b);

   temp = a;
   a = b;
   b = temp;

   printf ("%d %d\n", a, b);
   return 0;
}
```

Esse exemplo funcionará exatamente como esperado: primeiramente ele imprimirá "5 10" e depois ele imprimirá "10 5".

Mas e se quisermos trocar várias vezes o valor de duas variáveis?

Ponteiros e Funções

É muito mais conveniente criar uma função que faça isso.

```c
#include <stdio.h>

void swap(int i, int j){
   int temp;
   temp = i;
   i = j;
   j = temp;
}

int main(){
   int a, b;
   a = 5;
   b = 10;
   printf ("%d %d\n", a, b);
   swap (a, b);
   printf ("%d %d\n", a, b);
   return 0;
}
```

Ponteiros e Funções

É muito mais conveniente criar uma função que faça isso.

O programa imprime duas vezes "5 10".

As variáveis a e b são locais à função main( ), e quando as passamos como argumentos para swap(), seus valores são copiados e passam a ser chamados de i e j; a troca ocorre entre i e j, de modo que quando voltamos à função main() nada mudou.

```c
#include <stdio.h>

void swap(int i, int j){
   int temp;
   temp = i;
   i = j;
   j = temp;
}

int main(){
   int a, b;
   a = 5;
   b = 10;
   printf ("%d %d\n", a, b);
   swap (a, b);
   printf ("%d %d\n", a, b);
   return 0;
}
```

Ponteiros e Funções

- Definimos a função swap( ) como uma função que toma como argumentos dois ponteiros para inteiros;
- A função faz a troca entre os valores apontados pelos ponteiros.
- Já na função main( ), passamos os endereços das variáveis para a função swap().
- Deve-se colocar um & na frente das variáveis que estivermos passando para a função.
- Quando uma função recebe como parâmetros os endereços e não os valores das variáveis, dizemos que estamos fazendo uma chamada por referência;

Ponteiros e Funções

```c
#include <stdio.h>

void swap (int *i, int *j){
   int temp;
   temp = *i;
   *i = *j;
   *j = temp;
}

int main (){
   int a, b;
   a = 5;
   b = 10;
   printf ("\n\nEles valem %d, %d\n", a, b);
   swap (&a, &b);
   printf ("\n\nEles agora valem %d, %d\n", a, b);
   return 0;
}
```

Ponteiros e Estruturas

- Uma estrutura é uma coleção de variáveis referenciada por um nome, também chamada de tipo de dado agregado.

- Utilizando ponteiros para estruturas a forma como os membros são referenciados é alterada.

Ponteiros e Estruturas

- Uma estrutura é uma coleção de variáveis referenciada por um nome, também chamada de tipo de dado agregado.
- Utilizando ponteiros para estruturas a forma como os membros são referenciados é alterada.
- Um ponteiro para estrutura pode ser declarado da seguinte forma:

*struct <tipo> * <nome do ponteiro>;*

Ponteiros e Estruturas

- Um ponteiro para estrutura pode ser declarado da seguinte forma:

*struct <tipo> * <nome do ponteiro>;*

- A declaração do ponteiro para estruturas é exatamente igual à declaração de ponteiros para outros tipos de variáveis.

- Para evitar erros de compilação do tipo da estrutura deve ser previamente definido.

Ponteiros e Estruturas

```
struct NumeroComplexo
{
    float Real;
    float Imaginario;
};
```

```
struct NumeroComplexo * ptNC;
```

Ponteiros e Estruturas

- Para inicializar o ponteiro com um endereço válido é necessário utilizar o operador ‘&’ antes de uma estrutura.

```
struct NumeroComplexo Num1;

ptNC = &Num1;
```

Ponteiros e Estruturas

Referenciando Elementos da Estrutura

- Para acessar os membros da estrutura o operador seta ‘->’ deve ser utilizado.
- Esse operador é utilizado no lugar do operador ponto e determina que o acesso ao membro será realizado por um ponteiro.

Ponteiros e Estruturas

Referenciando Elementos da Estrutura

- O acesso ao membro *Real* que foi definido na estrutura NumeroComplexo.

```
ptNC->Real
```

Ponteiros e Estruturas

Referenciando Elementos da Estrutura

- Ao utilizar o operador seta estamos acessando o conteúdo do membro da estrutura.
- É possível obter o endereço individual de cada variável declarada na estrutura.
- Neste caso, o operador ‘&’ deve ser utilizado antes do nome da estrutura.

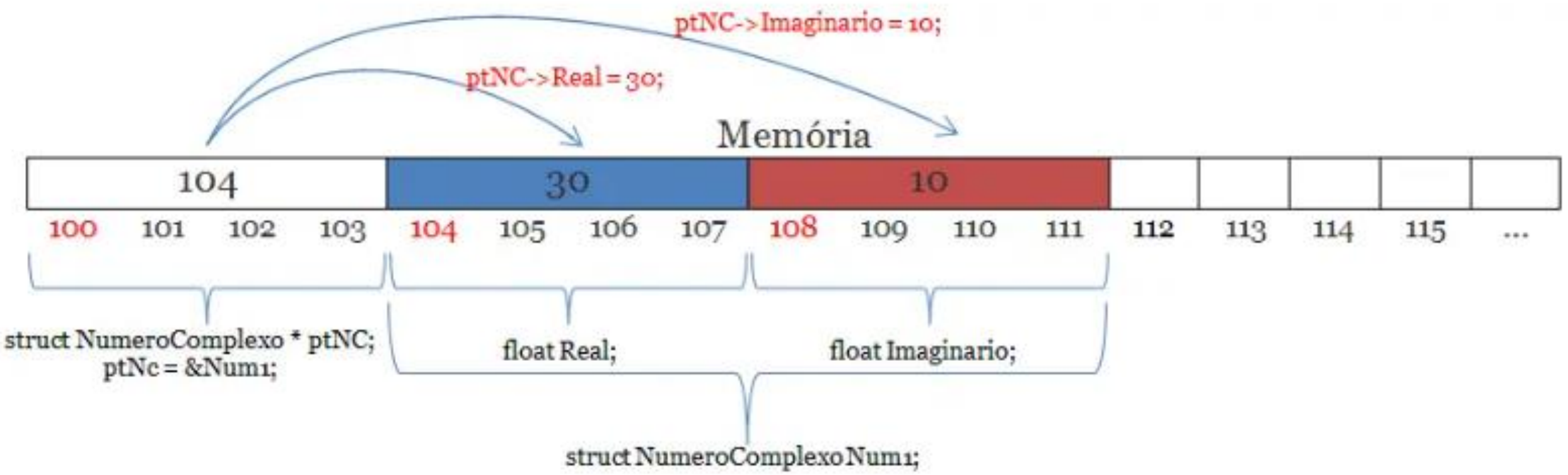
Ponteiros e Estruturas

Referenciando Elementos da Estrutura

- Obter o endereço de um membro a partir de um ponteiro para estrutura.

```
&ptNC->Real
```

Ponteiros e Estruturas



Ponteiros e Estruturas

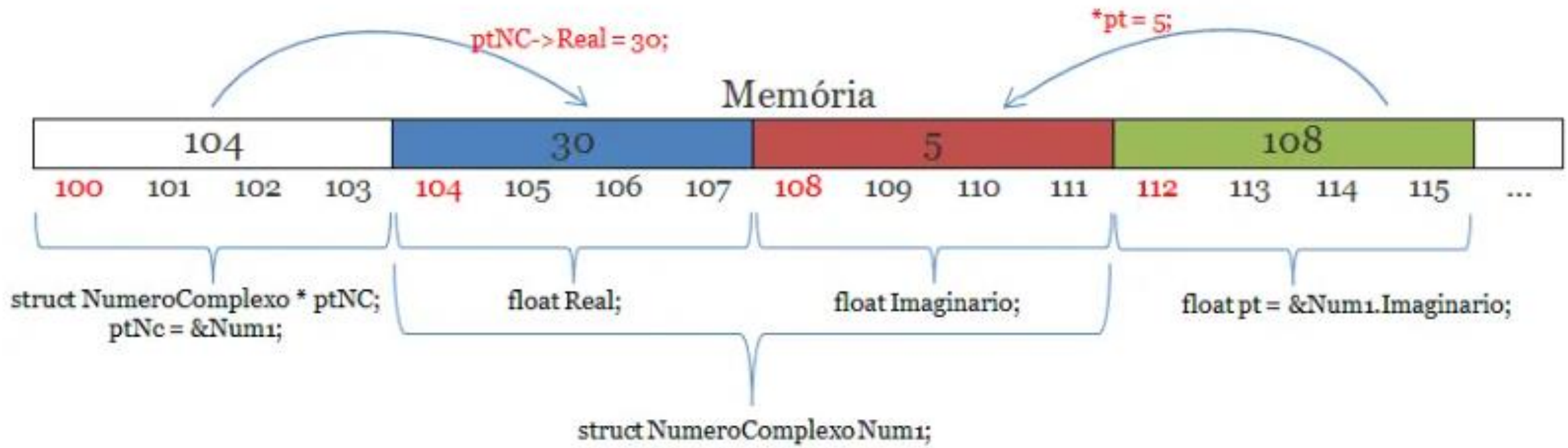
- Obter o endereço de um membro específico pode ser interessante em algumas situações:
    - Passar apenas alguns valores de uma estrutura para uma determinada função.

- Um ponteiro para *float* é utilizado para acessar o membro *Imaginario* da estrutura NumeroComplexo.

```
float * pt = &Num1.Imaginario;
```

Ponteiros e Estruturas

```
float * pt = &Num1.Imaginario;
```



Ponteiros e Estruturas

Passando Estruturas para Funções

- Uma das vantagens de poder criar um ponteiro para função é utilizar um ponteiro como parâmetro de função, ou seja, uma estrutura pode ser passada por referência para uma determinada função.

- Isso pode ser considerado um ganho de desempenho, pois ao passar uma estrutura por referência estaremos apenas manipulando o seu endereço, evitando que todos os seus membros sejam passados por valor.

FIM!