Funções

- Funções são trechos de programa que possuem um "escopo" (universo de definição das variáveis) exclusivo, sem relação com o escopo de outras funções.
  - A função `main` é uma função como as outras, com seu próprio escopo.
  - As variáveis globais fazem parte do escopo de todas as funções.
- Todas as variáveis de uma função só são visíveis dentro do seu escopo:
  - As variáveis criadas dentro da função.
  - As variáveis passadas como parâmetro para a função, que são criadas quando a função inicia e recebem um valor inicial igual ao do parâmetro correspondente.

```
void func(int P)  // Variaveis locais de func:
{                 // P e Q
  int Q;
}
```

- Funções `inline` não são compiladas como funções independentes. Elas são substituídas, no local onde são chamadas e a cada chamada, pelo seu código de execução.
  - Funções `inline` devem ser muito curtas. Substituem com vantagens os `#define`
  - Não altera em nada a lógica do programa a função ser `inline`. Apenas a velocidade de execução pode ser reduzida e o tamanho do programa executável ser aumentado.

```
double pow2(double x) {return x*x;}          // OK: Mais lento
inline double pow2(double x) {return x*x;} // OK: Mais rápido
// ERRO: função muito grande
inline double func(double x) { ... várias linhas ... }
```

Parâmetros

- Funções sem parâmetros têm a palavra reservada `void` como sua lista de parâmetros.
- As variáveis que são parâmetros de uma função são cópias dos valores das variáveis originais que correspondem aos parâmetros na chamada da função.
  - Alterações na cópia não alteram a variável original passada como parâmetro.
- Parâmetros do tipo ponteiro também são passados como cópias do endereço original.
  - Da mesma forma que com todos os outros tipos de variáveis, alterações no valor da cópia da variável não alteram o conteúdo (endereço) do ponteiro original.
  - Caso se altere o valor para onde a cópia do endereço aponta, essa alteração terá impacto na variável original, pois a cópia aponta para a mesma posição de memória.
- Passar parâmetro por referência (`tipo_var &nome_var`) tem o mesmo efeito de passar por endereço, e não por cópia (alterações se refletem no programa principal). Contudo, na chamada e no corpo da função, não se utiliza a notação de endereços e ponteiros.

TIPO DE PASSAGEM DE PARÂMETRO		
POR CÓPIA	**POR ENDEREÇO**	**POR REFERÊNCIA**

`void func(int P)`
`{ P=2;} //Sem efeito`
 	
`int main(void)`
`{ int N = 3;`
`  func(N);`
`  cout << N;} // 3`
	`void func(int *P)`
	`{*P=2;  //Altera`
	`  P=NULL;} //Sem efeito`
	`int main(void)`
	`{ int N = 3;`
	`  func(&N);`
	`  cout << N;} // 2`
		`void func(int &P)`
		`{ P=2;} //Altera`
		
		`int main(void)`
		`{ int N = 3;`
		`  func(N);`
		`  cout << N;} // 2`

- Parâmetros de função podem ter valores por *default*. Caso se chame a função sem colocar explicitamente o valor do parâmetro correspondente, isso terá o mesmo efeito que se tivesse chamado a função colocando o valor *default* na hora da chamada.

```
int func(int P=2) { ... }
i = func(5);      // OK: P=5
i = func();       // OK: P=2
```

  - Caso a função tenha mais de um parâmetro, é permitido atribuir valores *default* ao(s) último(s), mas não deixando um parâmetro intermediário sem valor *default*.

```
int func(int P1,   int P2=5, int P3=0) { ... }   // OK
int func(int P1,   int P2,   int P3=0) { ... }   // OK
int func(int P1=2, int P2,   int P3=0) { ... }   // ERRO
```

  - Ao chamar uma função, não é permitido omitir nem um valor intermediário nem um parâmetro que não tenha valor *default*.

```
void func(int P1, string S2="XX", int P3=0)
{ cout << P1 << ": " << S2 << "=" << P3;}
func();                 // ERRO
func(2,"Nome");         // OK: imprime 2: Nome=0
func(2,25);             // ERRO: 25 não eh valido para S2
```

Retorno

- Funções do tipo `void` não retornam nada.
- Funções de qualquer outro tipo que não seja `void` devem finalizar com uma instrução `return`, que retornará uma cópia do seu parâmetro.
- Quando uma função retorna um valor, é criada uma nova variável temporária no programa principal e que não tem um nome específico associado a ela. Essa variável sem nome (no escopo do programa principal) é uma cópia (tem o mesmo valor) da variável no escopo da função que foi passada como parâmetro para o `return`.

```
int main(void)
{ int i,j;       // Cria i e j; valor inicial indefinido
  i = 3;         // i recebe 3
  j = func(i);   // j recebe o valor da variável s/nome
  cout << j;}    // Imprime 5
int func(int P)  // Cria P; valor inicial 3 (i)
{ int N = P+2;   // N recebe 5
  return N;}     // Cria variável s/nome; valor inicial 5 (N)
```

- O retorno pode ser feito por cópia, por endereço ou por referência.
  - Ao se retornar um endereço ou uma referência a uma variável, o valor de retorno não pode ser uma constante ou uma variável local.

TIPO DE RETORNO		
POR CÓPIA	**POR ENDEREÇO**	**POR REFERÊNCIA**

`int  func1()`
`{ int P;`
`  ...`
`  return P;} //OK`
`int  func2()`
`{ return 5;} //OK`
	`int *func1()`
	`{ int P;`
	`  ...`
	`  return &P;} //ERRO`
	`int *func2()`
	`{ return 5;} //ERRO`
		`int &func1()`
		`{ int P;`
		`  ...`
		`  return P;} //ERRO`
		`int &func2()`
		`{ return 5;} //ERRO`