1.Funções de programas em C

Geralmente, os programas em C são escritos combinando novas funções que o programador escreve com funções pré-definidas disponíveis na biblioteca padrão do C. Embora as funções da biblioteca padrão não sejam tecnicamente parte da linguagem C, são sempre fornecidas com os sistemas ANSI C. O programador pode escrever funções para definir tarefas específicas e que podem ser utilizadas em muitos locais dos programas. Algumas vezes elas são chamadas de funções definidas pelo programador. As instruções reais que definem a função são escritas apenas uma vez e são escondidas das outras funções.

As ***funções*** permitem ao programador modularizar um programa. ***Todas as variáveis*** declaradas em definições de funções são ***variáveis locais***. A maioria das funções tem uma ***lista de parâmetros***. Os parâmetros de uma função também são variáveis locais. Há vários motivos para "funcionalizar" um programa. O método dividir-para-conquistar torna o desenvolvimento do programa mais flexível. Outra motivação é a capacidade de reutilização do software.

As instruções entre as chaves formam o corpo da função, também chamado de ***bloco***. Um bloco é simplesmente uma instrução composta que inclui declarações. As variáveis podem ser declaradas em qualquer bloco e os blocos podem estar aninhados. Uma função não pode ser definida no interior de outra função sob quaisquer circunstâncias.

As funções são invocadas por uma ***chamada de função***. A chamada da função específica o nome da função e fornece informações de que a referida função necessita para realizar a tarefa designada.

Um dos recursos mais importantes do ANSI C é o ***protótipo de função***. Um protótipo de função diz ao compilador do tipo do dado retornado pela função, o número de parâmetros que a função espera receber, os tipos dos parâmetros e a ordem na qual esses parâmetros são esperados. O compilador usa protótipos de funções para validar as chamadas de funções.

A ***lista de parâmetros*** é uma lista separada por vírgulas contendo as declarações dos parâmetros recebidos pela função quando ela é chamada. Se uma função não recebe nenhum valor a lista de parâmetros é ***void***. Deve ser listado explicitamente um tipo para cada parâmetro, a menos que o parâmetro seja do tipo ***int***.

Se o protótipo de uma função não for incluído em um programa, o compilador forma seu próprio protótipo de função usando a primeira ocorrência da mesma. Por *default*, o compilador assume que a função retorna um ***int*** e nada é assumido no que se diz respeito aos argumentos.

Uma chamada de função que não corresponda a um protótipo de função causa um erro de sintaxe. Também é gerado um erro se o protótipo da função e sua definição discordarem. Outro recurso importante dos protótipos de funções é a coerção de argumentos e a imposição de argumentos do tipo apropriado.

```
tipo_de_retorno função (tipo_de_valores);


main(){
   instruções em main;

   função(valores);

   return (0);

}


tipo_de_retorno função (tipo_de_valores valores){

   instruções em função;

   return (tipo_de_retorno);

}
```

As duas maneiras de ativar as funções em muitas linguagens de programação são ***chamadas por valor***. Quando os argumentos são passados através de uma chamada por valor, é feita uma cópia do valor dos argumentos e a mesma é passada para a função chamada. As modificações na cópia não afetam o valor original de uma variável na função que realizou a chamada. A chamada por valor deve ser usada sempre que a função chamada não precisar modificar o valor da variável original da função chamadora. Isso evita os efeitos colaterais que retardam o desenvolvimento de sistemas corretos e confiáveis de software. Em C, todas as chamadas são por valor.

```
tipo_de_retorno passagemPor_valor (tipo_de_valores);


main(){
   instruções em main;
```

```
    passagemPor_valor(valores); // qualquer valor passado aqui, declarado dentro de main, não terá seu valor alterado DENTRO de main

   return (0);

}
```

```
tipo_de_retorno passagemPor_valor (tipo_de_valores valores){

   instruções em função;

   return (tipo_de_retorno);

}
```

Quando um argumento é passado através de uma ***chamada por referência***, a função invocadora permite realmente que a função chamada modifique o valor original da variável. A chamada por referência só deve ser usada com funções confiáveis que precisem modificar a variável original. É possível simular a chamada por referência usando operadores de endereços e de ações indiretas.

```
tipo_de_retorno passagemPor_referência (tipo_de_valores*);


main(){
   instruções em main;

    passagemPor_referência(&valores); // qualquer valor passado aqui, com o sinal de e-comercial (&), será modificado TANTO na função, quanto em main

   return (0);

}


tipo_de_retorno passagemPor_referência (tipo_de_valores *valores){
```

```
    instruções em função;

    return (tipo_de_retorno);

}
```

Existem três maneiras de retornar controle ao ponto no qual uma função foi chamada. Se a função não fornecer um valor como resultado, o controle é retornado simplesmente quando a chave que indica o término da função é alcançada, ou executando a instrução ***return***. Se a função fornecer um valor como resultado, a instrução ***return (expressão);*** retorna o valor da expressão à função que realizou a chamada como mostrado nos protótipos acima.