Funções

- Escopo de variáveis

O escopo de variáveis é o conjunto de regras que determinam o uso e a validade de variáveis nas diversas partes do programa.

Veremos agora três tipos de variáveis, no que se refere ao escopo:

Funções

- Variáveis locais

Variáveis locais são aquelas que só têm validade dentro do bloco no qual são declaradas. **Podemos declarar variáveis dentro de qualquer bloco.**

Só para lembrar: um bloco começa quando abrimos uma chave e termina quando fechamos a chave. A declaração de variáveis locais é a primeira coisa que devemos colocar num bloco.

UNIVASF

```
func1 (...)
{
        int abc,x,z;
        ...
}
func (...)
{
        int z;
        ...
}
main ()
{
   int x,y;
   if (...)
   {
      float A,B,C,x;
      ...
    }
   ...
}
```

Funções

- Parâmetros formais

Os parâmetros formais são declarados como sendo as entradas de uma função. Um parâmetro formal é uma variável local da função que é inicializada com valores externos à função.

Ao se alterar o valor de um parâmetro formal esta alteração não terá efeito na variável que foi passada à função. Isto ocorre pois, em C, quando se passa parâmetros para uma função, são copiados os valores das variáveis, expressões ou constantes para os parâmetros formais. Isto é, os parâmetros formais existem independentemente das variáveis que foram passadas para a função, estes apenas são inicializados com uma cópia dos valores passados para a função.

Funções

- Parâmetros formais (continuação)

Exemplo:

```
#include <stdio.h>
void f1(int i)
{
   i=18;
}
main()
{
   int i=3;
   f1(i);
   printf (″%d″,i);
   return 0;
}
```

Funções

Exercício:

Escreva o código fonte de um programa, na linguagem C, que manipule um vetor de inteiros com dez elementos. O programa deve possuir uma função que receba o vetor e retorne o maior valor contido no mesmo. As seguintes manipulações devem ser feitas: o vetor deve ser inicializado e, por meio da utilização da função mencionada, o maior valor contido no vetor deve ser impresso na saída padrão.

```
#include <stdio.h>
#define n 10
int maior_valor (int vetor[n])
{
   int i,maior_v;
   for(i=0;i<n;++i)
      if(!i)
         maior_v=vetor[i];
      else
         if(maior_v<vetor[i])
            maior_v=vetor[i];
   return maior_v;
}
main()
{
   int i,v[n];
   for(i='\0';i<n;++i)
   {
      printf("\nEntre com o elemento v[%d]: ",i+1);
      scanf("%d",&v[i]);
   }
   printf("\nO maior valor contido no vetor eh: %d",maior_valor(v));
   return 0;
}
```

```
#include <stdio.h>
#define n 2
void teste (int vetor[n]) {
   int aux;
   aux = vetor[0];
   vetor[0] = vetor[1];
   vetor[1] = aux;
}
main() {
   int i,v[n];
   for(i='\0'; i<n; ++i) {
      printf("\nEntre com o elemento v[%d]: ",i+1);
      scanf("%d",&v[i]);
   }
   printf("\nO vetor fornecido eh: \nVetor = {");
   for(i='\0'; i<n; ++i)
      printf(" %d,", v[i]);
   printf ("\b }");
   teste (v);
   printf("\nO vetor apos a chamda da funcao eh: \nVetor = {");
   for(i='\0'; i<n; ++i)
      printf(" %d,", v[i]);
   printf ("\b }");
   return 0;
}
```

No caso de parâmetros formais que são vetores ao serem alterados dentro da função estas alterações se refletem fora!

Funções

- Variáveis globais

Variáveis globais são declaradas fora de todas as funções do programa. Elas são conhecidas e podem ser alteradas por todas as funções do programa que sucedem sua declaração. Quando uma função tem uma variável local com o mesmo nome de uma variável global a função dará preferência à variável local. Vamos ver um exemplo:

UNIVASF

```
func1 (...)
{
        int x,y;
        ...
}
int z,k;
func2 (...)
{
        int x,y,z;
        ...
        z=10;
        ...
}
main ()
{
        int count;
        z=7;
        func2(...);
        ... /* z?*/
}
```

Exercício:

Analise o seguinte programa e indique o que será impresso na saída padrão.

```
#include <stdio.h>
int num;
int func(int first, int sec)
{
   first = (first+sec)/2;
   num -= first+1;
   return first;
}
 main()
{
   int first = 0, sec = 50;
   num = 10;
   printf("\nnum antes = %d\tfirst antes = %d\tsec antes = %d", num,
first, sec);
   num += func(first, sec);
   printf("\nnum depois = %d\tfirst depois = %d\tsec depois = %d",
num, first, sec);
}
```

Funções

Exercício:

Escreva o código fonte de um programa, na linguagem C, que manipule um vetor de inteiros com dez elementos. O programa deve possuir uma função responsável pela inicialização do vetor e outra função que efetue a impressão do vetor na saída padrão com um layout adequado. Por meio das funções mencionadas, o programa deve inicializar o vetor e em seguida retorná-lo no monitor.

```
#include <stdio.h>
#define n 10
int vetor[n];
void inicializar_vetor()
{
   int i;
   for(i='\0';i<n;++i)
   {
      printf("\nEntre com o elemento v[%d]: ",i+1);
      scanf("%d",&vetor[i]);
   }
}
```

```
void imprimir_vetor()
{
   int i;
   for(i='\0';i<n;++i)
      if(!i)
         printf("vetor = { %d, ", vetor[i]);
      else
         if(i==n-1)
            printf("%d }", vetor[i]);
         else
            printf("%d, ", vetor[i]);
}
```

```
int main()
{
   inicializar_vetor();
   imprimir_vetor();
   return 0;
}
```

Funções

- Protótipos de funções

Protótipos são declarações de funções. Isto é, você declara uma função que irá usar. O compilador toma então conhecimento do formato daquela função antes de compilá-la. Possibilitando assim, antes da compilação da função, a validação da utilização da mesma.

Um protótipo tem o seguinte formato:

TipoDeRetorno NomeDaFunção (DeclaraçãoDeParâmetros);

onde o *TipoDeRetorno*, o *NomeDaFunção* e a *DeclaraçãoDeParâmetros* são os mesmos que você pretende usar quando realmente escrever a função.

```
#include <stdio.h>
float Square (float a);
int main ()
{
  float num;
  printf ("Entre com um numero: ");
  scanf ("%f",&num);
  num=Square(num);
  printf ("\n\nO seu quadrado vale: %f\n",num);
  return 0;
}
float Square (float a)
{
  return (a*a);
}
```

Tipos de Dados Definidos pelo Usuário

1. Estruturas

Uma estrutura agrupa várias variáveis numa só. A estrutura, então, serve para agrupar um conjunto de dados não similares, formando um novo tipo de dado.

Tipos de Dados Definidos pelo Usuário

1. Estruturas (continuação)

Para se criar uma estrutura usa-se o comando **struct**. Sua forma geral é:

```
struct nome_do_tipo_da_estrutura
{
  tipo_1 nome_campo1;
  tipo_2 nome_campo2;
  ...
  tipo_n nome_campon;
} variaveis_do_tipo_da_estrutura;
```

Tipos de Dados Definidos pelo Usuário

```
struct nome_do_tipo_da_estrutura
{
   tipo_1 nome_campo1;
   tipo_2 nome_campo2;
   ...
   tipo_n nome_campon;
};
struct
{
   tipo_1 nome_campo1;
   tipo_2 nome_campo2;
   ...
   tipo_n nome_campon;
} variáveis_estrutura;
```

Tipos de Dados Definidos pelo Usuário

1. Estruturas (continuação)

Vamos criar uma estrutura de endereço:

```
struct tipo_endereco
{
   char rua [50];
   int numero;
   char bairro [20];
   char cidade [30];
   char sigla_estado [3];
   long int CEP;
};
```

Tipos de Dados Definidos pelo Usuário

1. Estruturas (continuação)

Vamos agora criar uma estrutura chamada ficha_pessoal com os dados pessoais de uma pessoa:

```
struct ficha_pessoal
{
  char nome [50];
  long int telefone;
  struct tipo_endereco endereco;
};
```

```
#include <stdio.h>
#include <string.h>
/* aqui, entrariam as declarações das struct's vistas
   anteriormente */
main ()
{
    struct ficha_pessoal ficha;
    strcpy (ficha.nome,"Luiz Osvaldo Silva");
    ficha.telefone=4921234;
    strcpy (ficha.endereco.rua,"Rua das Flores");
    ficha.endereco.numero=10;
    strcpy (ficha.endereco.bairro,"Cidade Velha");
    strcpy (ficha.endereco.cidade,"Belo Horizonte");
    strcpy (ficha.endereco.sigla_estado,"MG");
    ficha.endereco.CEP=31340230;
    return 0;
}
```

Tipos de Dados Definidos pelo Usuário

2. Definição de tipo

O comando typedef é utilizado para definir um novo tipo de dado. Ele é utilizado da seguinte forma:

typedef tipo nome_do_tipo;

Tipos de Dados Definidos pelo Usuário

Exemplo:

```
typedef struct
{
        char dia[15];
        int hora;
        int minuto;
 } data;
...
data d1;
...
```

Tipos de Dados Definidos pelo Usuário

Exercício:

Defina um tipo de dado capaz de armazenar as seguintes informações sobre um determinado aluno: nome, data de nascimento, número de matricula, CPF e coeficiente de rendimento.

Posteriormente, construa um programa que manipule um vetor com 5 registros de alunos, onde cada registro é um elemento do tipo de dado definido. A manipulação do vetor é feita através das seguintes funções: inicializar vetor, imprimir um determinado registro com base no valor do campo CPF e imprimir um determinado registro com base em sua posição no vetor. O programa deve se utilizar de forma satisfatória das funções mencionadas.

UNIVASF

```
#include<stdio.h>
#include<string.h>
#define num_reg 5
typedef struct
{
   char nom[30];
   char dat[9];
   unsigned long int n_mat;
   char cpf[13];
   float cr;
}registro;
registro v[num_reg];
void inicializar_vet ();
void imprimir_reg_pos (unsigned int);
void imprimir_reg_cpf (char chave[13]);
```

```
main()
{
   int i;
   char cpf[13];
   inicializar_vet();
   printf("\nImprimir registro da posicao: ");
   scanf("%d",&i);
   imprimir_reg_pos (i);
   printf("\n\n\nImprimir registro com CPF: ");
   scanf("%s",cpf);
   imprimir_reg_cpf (cpf);
   return 0;
}
/* Com o objetivo de não tornar a resolução muito extensa, visando
evitar o desvio do foco central do exercício, a validação da entrada
CPF foi omitida. Contudo, implemente esta como um exercício. */
```

```
void inicializar_vet ()
{ /* Com o objetivo de não tornar a resolução muito extensa, visando
evitar o desvio do foco central do exercício, as validações das entradas
foram omitidas. Contudo, estas devem ser implementadas! */
   int i;
   for (i=0; i<num_reg; i++) {
      printf("\nEntre com as informacoes do registro numero
      %d.\n", i+1);
      printf("\nNome: ");
      scanf("%29s",v[i].nom);
      printf("\nData (dd/mm/aa): ");
      scanf("%s",v[i].dat);
      printf("\nNumero de matricula: ");
      scanf("%ld",&v[i].n_mat);
      printf("\nCPF: ");
      scanf("%s",v[i].cpf);
      printf("\nCoeficinte de rendimento: ");
      scanf("%f",&v[i].cr);  }
}
```

```
void imprimir_reg_pos (unsigned int pos)
{
  if (pos>num_reg || !pos)
  {
    printf("\nErro!\nPosicao invalida para impressao!\n");
    exit(1);
  }
  else
  {
    printf("\n\n\nRegistro numero %d.\n",pos);
    printf("\nNome: \t\t\t\t%s",v[pos-1].nom);
    printf("\nData: \t\t\t\t%s",v[pos-1].dat);
    printf("\nNumero de matricula: \t\t%ld",v[pos-1].n_mat);
    printf("\nCPF: \t\t\t\t%s",v[pos-1].cpf);
    printf("\nCoeficinte de rendimento: \t%.3f",v[pos-1].cr);
  }
}
```

```
void imprimir_reg_cpf (char chave[13])
{
   int i;
   for (i=0; i<num_reg; i++)
      if(!strcmp(v[i].cpf, chave))
      {
         printf("\n\n\nRegistro numero %d.\n",i+1);
         printf("\nNome: \t\t\t\t%s",v[i].nom);
         printf("\nData: \t\t\t\t%s",v[i].dat);
         printf("\nNumero de matricula: \t\t%ld",v[i].n_mat);
         printf("\nCPF: \t\t\t\t%s",v[i].cpf);
         printf("\nCoeficinte de rendimento: \t%.3f",v[i].cr);
         return;
      }
   printf("\n\nNao existe registro com o CPF especificado.\n");
}
```