SSC0101 - ICC1 – Teórica

Introdução à Ciência da Computação I

Modularização de Programas
Parte III

Prof. Vanderlei Bonato: vbonato@icmc.usp.br
Prof. Claudio Fabiano Motta Toledo: claudio@icmc.usp.br

Sumário

- Classes de armazenamento
  - auto
  - extern
  - register
  - static
- Variáveis locais x Variáveis globais
- Arquivos de cabeçalho
- Recursão

Classes de armazenamento

- Variáveis e funções em C possuem dois atributos:
  - Tipo
  - Classe de armazenamento
- Há 4 classes de armazenamento:
  - auto
  - extern
  - register
  - static

Classes de armazenamento - auto

- A classe de armazenamento **auto** é a mais comumente usada.
- Variáveis declaradas dentro do corpo de funções são da classe **auto** por *default*.
- Por isso, não precisa ser utilizada a palavra reservada **auto**.
- Elas apresentam escopo restrito ao bloco onde foram declaradas.
- O sistema aloca memória para as variáveis automáticas que é liberada quando há saída do bloco.

Classe de armazenamento - extern

- Variáveis externas permitem transmitir informação através dos blocos e funções.
- Ela é declarada antes e fora do escopo dos blocos e funções onde será utilizada.
- Logo, ela será global a todas as funções declaradas depois dela.

- Exemplo:

```
#include <stdio.h>
int a=1, b=2, c=3;              /* variáveis globais */
int f(void);
int main(void)
{
  printf("%3d\n", f( ));
  printf("%3d%3d%3d\n", a,b,c);
  return 0;
}

inf f(void)
{
extern int a; /*O compilador é informado de que  a  variável*/
                    /*é definida aqui ou em outro arquivo */
  int b, c;
  a=b=c=4;
  return (a+b+c);
}
```

Classes de armazenamento - register

- O compilador é avisado de que as variáveis **register** devem ser armazenadas em registradores.
- Isso se justifica quando um conjunto de variáveis é acessado frequentemente.
- Alguns parâmetros de funções e variáveis de estruturas de repetição (laços) podem justificar sua declaração como **register**.
- Essas variáveis retornam à classe **auto**, se o compilador não consegue alocar um registrador.
- Exemplo: register i; ⇔ register int i;

Classes de armazenamento - static

- A classe **static** permite a uma variável local manter seu valor, quando o fluxo de execução retorna ao bloco onde está declarada.
- Exemplo:

```
void f(void)
{
 static int cnt=0;
 ++cnt;
 if(cnt%2 ==0)
    .....
  else
    .....
}
```

1ª chamada da função: cnt=0 inicialmente;
2ª chamada da função: cnt=1 inicialmente;
3ª chamada da função: cnt=2 inicialmente;

Classes de armazenamento – static

- No exemplo anterior, a declaração de cnt como variável estática em f() preserva cnt como de uso privado da função f().
- Outro uso, ainda mais efetivo ,da classe de armazenamento **static** ocorre em combinação com a variáveis da classe **extern.**
- As variáveis **static extern** têm o escopo restrito das variáveis **extern.**
- Elas não estão disponíveis para funções definidas antes da sua declaração no arquivo.
- Também não estão disponíveis para funções definidas em outros arquivos.

Exemplo:

```
#define INITIAL_SEED              17
#define MULTIPLIER                25173
#define INCREMENT                 13849
#define MODULUS                    65536
#define FLOATING_MODULUS   65536.0
/* variável externa, mas privada a este arquivo*/
static unsigned  seed = INITIAL_SEED;
Unsigned random(void)
{
 seed = (MULTIPLIER*seed + INCREMENT)%MODULUS;
 Return seed;
}
Double probability(void)
{
  seed = (MULTIPLIER*seed + INCREMENT)%MODULUS;
  return (seed/FLOATING_MODULUS);
}
```

Classes de armazenamento - inicialização

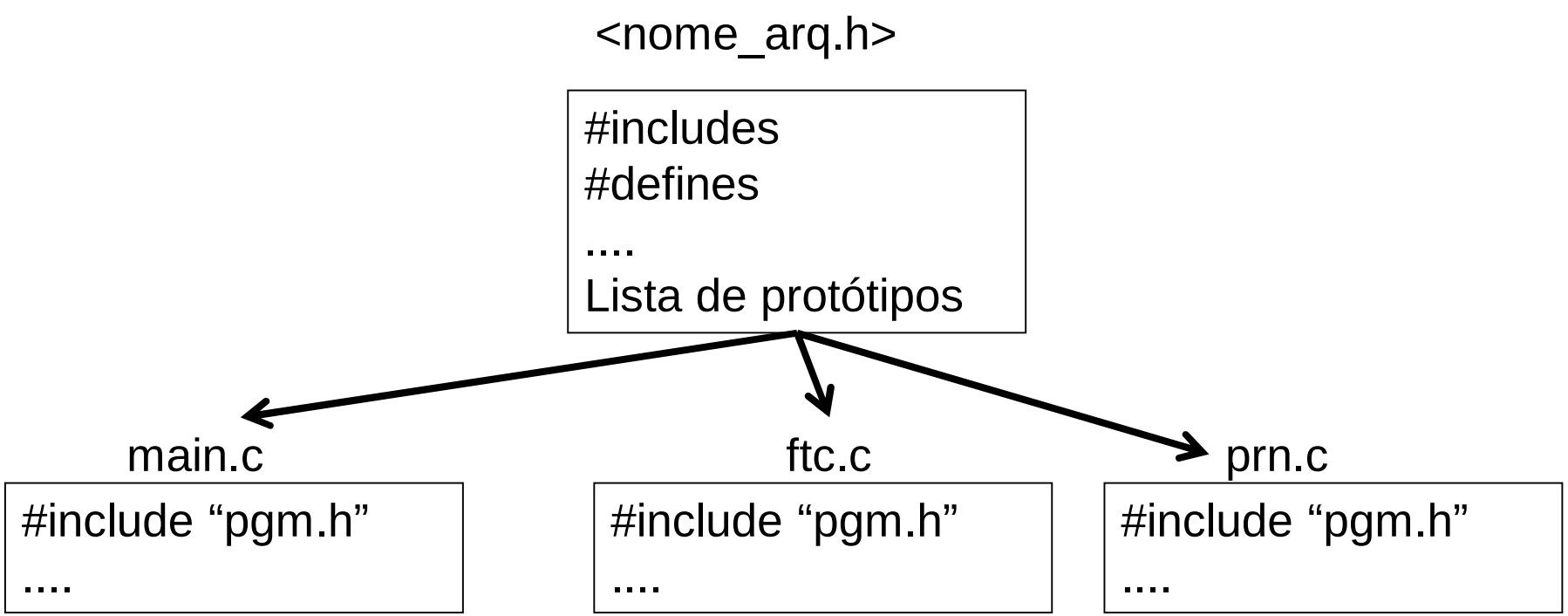
- Variáveis **extern** e **static**, que não são explicitamente iniciadas pelo programa, são iniciadas com zero pelo sistema.

- Isso inclui estruturas de dados como: array, strings, ponteiros, struct, union.

- Variáveis **auto** e **register** normalmente não são iniciadas pelo sistema. Logo, podem conter "lixo".

Arquivos de cabeçalho

- Vamos considerar o desenvolvimento de um programa composto por vários arquivos .
- Arquivos que compartilham declarações e definições podem ter essas informações centralizadas em arquivos de cabeçalho:



Arquivos de cabeçalho

pgm.h

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

#define   N   3

void  fct1(int k);
void fct2(void);
void  wrt_info (char *);
```

main.c

```
#include  “pgm.h”

int main (void){
 char ans;
 int i,n = N;
 printf (“deseja mais informação ?”);
 scanf(“ %c”, &ans);
 if(ans==‘y’ || ans=‘y’)
    wrt_info(“pgm”);
 for( i=0; i < n; ++i)
     fct1(i);
 return 0;
}
```

Arquivos de cabeçalho

pgm.h

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

#define   N   3

void  fct1(int k);
void fct2(void);
void  wrt_info (char *);
```

fct.c

```
#include  “pgm.h”


void fct1(int n) {
 int i;
 printf (“olá de fct1()\n”);
 for (i=0; i < n; ++i)
   fct2();;
}

void fct2(void) {
printf (“olá de fct2()\n”);
}
```

Recursão

- Uma função é recursiva quando chama a si própria.
- Exemplo:

```
int sum(int n){
 if (n<=1)
   return n;
 else
   return (n+sum(n-1));
}
```

Entrada	Saída	
Sum(1)	1	
Sum(2)	2 + sum(1)	2+ 1
Sum(3)	3 + sum(2)	3 + 2 + 1
Sum(4)	4 + sum(3)	4 + 3 + 2 + 1

Recursão

- Exemplo:

```
int fat(int n){
 if (n<=1)
   return 1;
 else
   return (n*fat(n-1));
}
```

- A partir de determinado valor de n, as saídas do programa podem fornecer valores errados.
- Qual o motivo?

Recursão

- Exemplo: Sequência de Fibonacci.

f(0)=0, f(1)=1, f(i+1)=f(i)+f(i-1), i=1,2,...

```
int fib(int n){
 if(n<=1)
  return n;
 else return((fib(n-1)+fib(n-2));
}
```

n	F(n)	Número de chamadas da função
0	0	1
1	1	1
2	1	3
...	...	...
23	28657	92735
24	46368	150049

Exercício

- Faça um programa que implemente a função recursiva:

  T(1)=1, T(2)=2, T(3) = 3,

  T(n)=T(n-1) + 2T(n-2)+3T(n-3) para n>3.

  No programa principal, exiba os valores para n, T(n) e o número de chamadas da função

Exercício

- Usando a série de Taylor abaixo crie uma função recursiva que calcule o seno com precisão de 10^{-3}

$$\sin x = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n+1)!} x^{2n+1} \qquad \text{para todo } x$$

Referências

Ascencio AFG, Campos EAV. Fundamentos de programação de computadores. São Paulo : Pearson Prentice Hall, 2006. 385 p.

Kelley, A.; Pohl, I*., A Book on C*: programming in C. 4ª Edição. Massachusetts: Pearson, 2010, 726p.

Kernighan, B.W.; Ritchie, D.M. C, *A Linguagem de Programação*: padrão ANSI. 2ª Edição. Rio de Janeiro: Campus, 1989, 290p.

FIM Aula 13