2a Tarefa -- PCS2056

Linguagens e Compiladores

Professor: Ricardo Luis de Azevedo Rocha

Grupo:
Filipe Morgado Simões de Campos 5694101
Rafael Barbolo Lopes 5691262

1. Identifique, em alguma linguagem de programação imperativa que você conheça bem, cada um dos conceitos mencionados acima.

A linguagem de programação imperativa escolhida é **C**. O grupo não possui muita fluência nesta linguagem, mas como o compilador será desenvolvido com ela e existe muito interesse em aprendê-la com mais profundidade, o exercício será feito com base nela.

Construções sintáticas básicas

```
int numero = 0; // declaração da variável número e atribuição de valor 0 para ela
a = b; // copia b para a
a = (5*b + 3)/2; // copia expressão matemática para a
a = funcao(b, 4); // copia retorno de funcao para a
```

Máquina de execução subjacente

Não possui máquina de execução auxiliar pois o código em C pode ser compilado diretamente para código de máquina nativo.
Vale comentar que em linguagens mais modernas e independentes de plataforma é muito comum a existência de uma máquina virtual. Essa máquina virtual pode ser um interpretador que executa o código fonte ou uma máquina que recebe o código objeto, como é o caso de Java, em que o código fonte é compilado para rodar na máquina virtual e não nativamente no sistema operacional e hardware do usuário.

Constantes

Pode ser adicionado o especificador de tipo *const* para tornar uma variável constante. Segue exemplo:

```
const int CONSTANTE = 123;
```

Constantes globais podem ser definidas usando o comando *define*. Por exemplo:

```
#define PI 3.14159265;
```

Variáveis

```
int number = 0;                 // inteiro
char letter = 'c';              // char (1 caracter)
char *string = "hello";          // cadeia de caracteres
char buffer[50];                // buffer de 50 bytes
int array[3] = {1,2,3};         // vetor de 3 elementos inteiros
float fraction = 3.14;          // número real de 4 bytes
double bigfrac = 1.41;          // número real de 8 bytes
unsigned long num = 22;          // número sem sinal de 4 bytes
signed short num2 = -17;        // número com sinal (positivo/negativo) de 2 bytes
int multi_arr[5][2];            // exemplo de matriz 5x2: 5 ponteiros para vetores de 2 inteiros cada
```

Expressões

Operação	Operador	Exemplo de expressão	Valor de a	Valor da expressão
Multiplicação	*	a * 2	4	8
Divisão	/	a / 2	4	2
Adição	+	a + 2	4	6
Subtração	-	a -2	4	2
Incremento	++	a++	4	5
Decremento	--	a--	4	3
Resto da dsivisão	%	a % 3	4	1
AND bit-a-bit	&	a & 3	6	2
OR bit-a-bit	|	a | 3	6	7
XOR bit-a-bit	^	a ^ 3	6	5
Deslocamento para a esquerda	<<	a << 3	6	48
Deslocamento para a direita	>>	a >> 3	6	0

Atribuições de valor

Toda atribuição de valor é da forma *variavel = expressão;*. Exemplo:

```
auxiliar = 1;
```

Seqüências

Uma sequência de código em C é executada em um bloco do tipo:

```
{
        int a = 5;
        int b = 2;
        a = a/b;
}
```

Comandos condicionais

C suporta os comandos *if* e *switch* para programação de desvios condicionais.

O uso do **if** segue a seguinte estrutura:

```
if (expressao_verdadeira) {
        sequência de comandos;
}
```

O uso do **switch** segue a seguinte estrutura:

```
switch (expressao) {
        case valor_1:
                sequência de comandos;
                break;
        case valor_n:
                sequência de comandos;
                break;
        default:
                sequência de comandos;
                break;
}
```

A tabela abaixo representa os operadores condicionais presentes em C:

Operador	**Significado**
==	*igual*
!=	*diferente*
<	*menor*
<=	*menor ou igual*
>	*maior*
>=	*maior ou igual*

É possível aplicar o operador condicional para o cálculo de uma expressão. Para isto, deve-se utilizar a seguinte construção:

condition ? expression1 : expression2

Exemplo de uso de operador condicional para cálculo de uma expressão:

*s = (x < 0) ? -1 : x * x;*

Se x é menor que zero, então s = -1

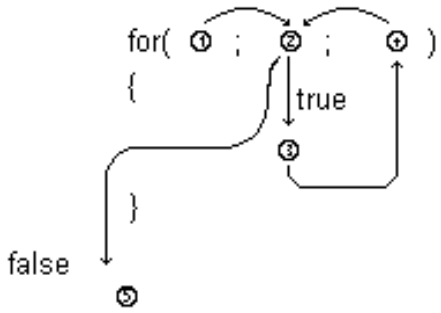
*Senão, então s = x * x*

Comandos iterativos

Em C, é possível usar os comandos *for*, *while* e *do-while*. Eles são usados em laços.

Estrutura do **for**:

```
for (condicao_inicial; condicao_para_continuar; comando_pos_iteracao) {
        sequência de comandos;
}
```



Estrutura do **while**:

```
while (condicao_para_continuar) {
        sequência de comandos;
}
```

Estrutura do **do-while**:

```
do {
        sequência de comandos;
} while (condicao_para_continuar);
```

Teorema de Böhm-Jacopini

Böhm e Jacopini demonstraram em 1966 que qualquer programa pode ser escrito sem a instrução **goto**. Um ano depois, Dijkstra publicou o artigo "Goto statement considered harmful". Em C, a instrução **goto** existe, mas não é aconselhável usá-la. Segue exemplo de como ela é usada:

```
rotulo_qualquer:
        if (condicao) {
                comandos;
                goto rotulo_qualquer;
        }
```

Estruturas de dados homogêneas

Para usar estruturas de dados homogêneas em C, pode-se usar vetores. Veja exemplo:

```
int valores_homogeneos_inteiros[5] = {1, 2, 3, 4, 5};
```

Estruturas de dados heterogêneas

Para usar estruturas de dados heterogêneas em C, pode-se usar *structs*. Veja exemplo:

```
struct data_nascimento {
        int dia;
        int mes;
        int ano;
        double tempo_em_milissegundos;
}
```

Ponteiros e estruturas dinâmicas

Declaração de um ponteiro:

```
int * pointer;
```

Atribuição de endereço para o valor que o ponteiro aponta:

```
pointer = &variavel;
```

Acesso do valor apontado por esse ponteiro:

```
valor = *pointer;
```

Procedimentos e Funções

Em C, pode-se declarar funções com ou sem retorno. A estrutura geral de uma função é:

```
tipo_de_dado_de_retorno   nome_da_funcao   (tipo_de_dado param1, tipo_de_dado param2, ...) {
        corpo_da_funcao;
}
```

Parâmetros e Argumentos

Para a estrutura:

```
tipo_de_dado_de_retorno   nome_da_funcao   (tipo_de_dado param1, tipo_de_dado param2, ...) {
        corpo_da_funcao;
}
```

Os parâmetros seriam as variáveis declaradas na função (*param1*, *param2*, ...).
Os argumentos são encontrados na chamada da função, por exemplo:

nome_da_funcao(arg1, arg2, ...);

Os valores de *arg1* e *arg2* são os argumentos da função.

Programa principal

O programa principal é declarado na função *main*. Exemplo:

```
#include <stdio.h>
main(){
        printf("Programar em C é relativamente fácil.\n");
}
```

2. Procure, através de exemplos, determinar as diversas formas que cada um dos comandos podem assumir. Se possível estabeleça, ainda que informalmente, uma forma geral para cada comando, de modo que as suas diversas variantes sejam todas casos particulares dessa forma geral.

Declaração e atribuição de valor a uma variável

Exemplos:

```
unsigned long number = 30;
char number = 'c';
```

Forma geral:

```
[atributo_modificador_da_variável]   tipo_de_variável   nome_variavel   =   valor;
```

Comandos condicionais

if:

```
if (condicao1) {
        sequência de comandos;
} else if(condicao2) {
        sequência de comandos;
} else {
        sequência de comandos;
}
```

switch:

```
switch (condicao) {
        case valor_1:
                sequência de comandos;
                break;
        case valor_n:
                sequência de comandos;
                break;
        default:
                sequência de comandos;
                break;
}
```

Forma geral:

nome_do_comando_condicional (condição) { sequência de comandos }

Comandos iterativos

for:

```
for (condicao_inicial; condicao_para_continuar; comando_pos_iteracao) {
        sequência de comandos;
}
```

Variação:

```
for(;;){
        sequência de comandos;
}
```

Neste caso, cria-se um loop infinito, já que a *condicao_inicial* e o *comando_pos_iteracao* são ignorados e o *condicao_para_continuar* é sempre considerado como *true*.

while:

```
while (condicao_para_continuar) {
        sequência de comandos;
}
```

do-while:

```
do {
        sequência de comandos;
} while (condicao_para_continuar);
```

Forma geral:

nome_do_comando_iterativo (*condição_para_continuar*) { sequência *de comandos* }

3. Considerando que uma linguagem de montagem será usada como código-objeto, estabeleça correspondência entre os diversos comandos da linguagem de alto nível e a correspondente seqüência de comandos em linguagem de montagem que lhe seja equivalente.

Para realizar este exercício, utilizamos a linguagem de montagem simbólica da disciplina *PCS2024 - Laboratório de Fundamentos de Engenharia de Computação*.

Linguagem C	***Linguagem de Montagem***
int a; *a = 12;*	`DOZE  K  /000C ; declaração de constante 12` `A  K  /0000 ; declaração de variável A` `LD  DOZE ; carrega 12 no acumulador (AC)` `MM  A  ; move AC para endereço de A`
if (a >= 0) { sequência *de comandos;* }	`LD  A  ; carrega A em AC` `JN  END  ; se AC < 0 desvia para END` `;` `; sequência de comandos` `;` `END #`
for (a = 12; a >= 0; *a = a -1*) { sequência *de comandos;* }	`UM  K  /0001  ; constante 1` `LD  DOZE  ; carrega 12 em AC` `MM  A  ; move AC para endereço de A` `LOOP  JN  END  ; se AC < 0 desvia para END` `;` `; sequência de comandos` `;` `LD  A  ; carrega A em AC` `-  UM  ; subtrai 1 de AC` `MM  A  ; move AC para endereço de A` `JP  LOOP  ; desvia para LOOP` `END #`
rotulo: *sequência de comandos;* *goto rotulo;*	`ROTULO  ; cria um rótulo` `;` `; sequência de comandos` `;` `JP  ROTULO  ; desvia para ROTULO`