Primeiros passos em C

O objetivo da aula é aprender:

1. a estrutura básica do programa.
2. Declarar variáveis inteiras e reais.
3. Escrever os valor de variáveis inteiras e reais.
4. Ler os valores de variáveis inteiras e reais.
5. Comando de desvio condicional
6. Blocos de comandos.
7. Declarando lista de variáveis.
8. Comando de repetição

1 Primeiros passos

A estrutura básica de um programa em linguagem C é a seguinte:

```
1 #include <stdio.h>
2 int main() {
3 }
```

Na linha 1, utilizamos a biblioteca de funções `stdio.h`[1]. Essa biblioteca fornece funções para realizar operações de leitura (entrada) e escrita (saída).

```
1 #include <stdio.h>
2 int main() {
3     printf("Hello World!!!\n");
4 }
```

O programa acima exibe a mensagem `"Hello World!!!"` na tela. Para executar um programa em C, precisamos seguir os seguintes passos:

Compilação

```
1 $ gcc exemplo.c -o exemplo
```

O comando acima transforma um arquivo fonte (`exemplo.c`) em um arquivo executável (`exemplo`). No Linux, o programa `exemplo` pode ser executado da seguinte maneira:

```
1 $ ./exemplo
```

Em programas, muitas vezes precisamos armazenar valores que são utilizados para realizar cálculos. Inicialmente, vamos trabalhar com variáveis para representar números inteiros.

```
1 #include <stdio.h>
2 int main() {
3     x = 2;
4 }
```

Ao tentar compilar o código acima, obtemos a seguinte mensagem de erro:

```
1 exemplo1.c: In function ‘main’:
2 exemplo1.c:3:9: error: ‘x’ undeclared (first use in this function)
3     3 |         x = 2;
4       |         ^
5 exemplo1.c:3:9: note: each undeclared identifier is reported only once
6 for each function it appears in
```

A mensagem de erro indica que a variável `x` não foi declarada. Para que uma variável possa ser utilizada, precisamos fornecer mais informações além de seu nome.

Declarando variáveis inteiras

```
1 #include <stdio.h>
2 int main() {
3     int x = 2;
4 }
```

Ao adicionar a palavra `int` antes do nome da variável `x`, estamos informando ao compilador que `x` é do tipo inteiro.

[1] https://cplusplus.com/reference/cstdio/

Escrevendo variável inteira

Para exibir o valor de uma variável inteira, utilizamos o comando `printf`. O `%d` é um formato específico que indica a exibição de uma variável do tipo inteiro.

```c
#include <stdio.h>
int main(){
    int x = 2;
    printf("o valor de %d", x);
}
```

O que acontece quando escrevemos o seguinte programa?

```c
#include <stdio.h>
int main(){
    int x = 2;
    printf("o valor de %d", x);
    x = 3;
    printf("o valor de %d", x);
}
```

A saída do programa será a seguinte:

```
o valor de 2o valor de 3
```

`\n` é um caractere especial que representa uma nova linha. Toda vez que utilizamos esse caractere ele vai pular a linha na saída.

```c
#include <stdio.h>
int main(){
    int x = 2;
    printf("o valor de %d\n", x);
    x = 3;
    printf("o valor de %d\n", x);
}
```

Agora, a saída vai ficar da maneira como queremos.

```
o valor de 2
o valor de 3

```

Agora, o que acontece quando escrevemos o seguinte programa:

```c
#include <stdio.h>
int main(){
    int x = 2;
    printf("o valor de %d\n", x);
    x = 2.5;
    printf("o valor de %d\n", x);
}
```

Perguntas:

1. Você acha que vai dar algum erro?
2. Se não der erro, qual seria a saída do programa?

Neste caso, o compilador não vai dar nenhum erro e o valor 2.5 será convertido para 2 (Por que?).

Agora, vamos aprender a declarar e escrever variáveis reais.

Declarando variável real

```c
#include <stdio.h>
int main(){
    int x;
    float y;
    x = 2;
    printf("o valor de %d\n", x);
    y = 2.5;
    printf("o valor de %f\n", y);
}
/*
Saída:
o valor de 2
o valor de 2.500000
*/
```

Note que o formato '%f' está sendo usado para escrever uma variável do tipo float usando para representar números reais.

Podemos escrever mais de uma variável com o mesmo comando.

Escrevendo duas variáveis

```c
#include <stdio.h>
int main(){
    int x;
    float y;
    x = 2;
    y = 2.5;
    printf("x = %d, y = %f\n", x, y);
}
/*
Saída:
x = 2, y = 2.500000
*/
```

O que acontece se mudarmos a ordem das variáveis?

```c
#include <stdio.h>
int main(){
    int x;
    float y;
    x = 2;
    y = 2.5;
    printf("x = %d, y = %f\n", y, x);
}
```

O compilador gera um executável contudo ele dar uma mensagem de warning.

```
exemplo1.c: In function ‘main’:
exemplo1.c:7:22: warning: format ‘%d’ expects argument of type ‘int’, but
argument 2 has type ‘double’ [-Wformat=]
    7 |         printf("x = %d, y = %f\n", y, x);
      |                      ~^             ~
      |                       |             |
      |                       int           double
      |                      %f
```

```
exemplo1.c:7:30: warning: format ‘%f’ expects argument of type ‘double’, but
argument 3 has type ‘int’ [-Wformat=]
    7 |         printf("x = %d, y = %f\n", y, x);
      |                              ~^         ~
      |                               |         |
      |                               double    int
      |                              %d
```

Note que o compilador está informando que está esperando um valor inteiro para substituir o formato %d contudo um valor float está sendo passado no segundo argumento da função.

Neste caso, o programa tem um comportamento indefinido (ou undefined behavior em inglês). Comportamento indefinido ocorre quando o código executa operações que não têm um resultado bem definido de acordo com a especificação da linguagem C.

Leitura de números inteiros

Agora, precisamos aprender como ler uma variável inteira. Para ler um variável, vamos utilizar a função scanf. Essa função faz uma leitura formatada. Logo, vamos utilizar novamente os formatos %d para leitura de inteiros e %f para a leitura de números reais. Para a leitura de um inteiro, a função recebe dois parâmetros: o primeiro parâmetro é o formato da leitura e o segundo é a variável que vai receber o valor lido.

```c
#include <stdio.h>
int main(){
    int x;
    scanf("%d", &x);
    printf("x = %d\n",x);
}
/*
Saída:
78
x = 78

*/
```

Classificação da nota

Seu programa deve ler dois números reais entre 0 e 10, representando as notas de um aluno, e calcular a média dessas notas. Considere os seguintes casos:

- Se média < 4: exibir a mensagem de que o aluno está **reprovado**.
- Se $4 \leq$ média < 7: exibir a mensagem de que o aluno precisa fazer uma **avaliação final**.
- Se média ≥ 7: exibir a mensagem de que o aluno está **aprovado**.

```c
#include <stdio.h>
int main(){
    float n1, n2, media;

    printf("Nota 1:");
    scanf("%f", &n1);
    printf("Nota 2:");
    scanf("%f", &n2);
    /*
    O que acontece quando calculamos
    a media dessa maneira?
```

```
    media = n1+n2/2;
    */
    media = (n1+n2)/2;

    if(media<4)
        printf("reprovado\n");
    if(media>=7)
        printf("aprovado\n");
    if(media >=4 && media < 7)
        printf("avaliação final\n");
}
```

Podemos reescrever esse programa evitando a utilização do operador lógico E (&&) usando o comando else.

```
#include <stdio.h>
int main(){
    float n1, n2, media;
    printf("Nota 1:");
    scanf("%f", &n1);
    printf("Nota 2:");
    scanf("%f", &n2);
    media = (n1+n2)/2;
    printf("media: %f\n", media);
    if(media<4)
        printf("reprovado\n");
    else // media >= 4
        if(media>=7)
            printf("aprovado\n");
        else // media >= 4 && media < 7
            printf("avaliação final\n");
}
```

Na hipótese da aluno ir para a AF, o aluno deverá obter nota igual ou superior a 4,0 (quatro) na avaliação final que somada à média das AP deverá resultar numa média igual ou superior a 5,0 (cinco).

Note que temos uma tentação de fazer o seguinte:

```
printf("media: %f\n", media);
if(media<4)
    printf("reprovado\n");
else // media >= 4
    if(media>=7)
        printf("aprovado\n");
    else // media >= 4 && media < 7
        printf("avaliação final\n");
        float n3, media2;
        scanf("%f", &n3);
        ...
```

O trecho acima não tem nenhum erro estrutural contudo queremos que quando o aluno for para fazer a avaliação final queremos que o programa execute várias instruções, então precisamos informar isso para o compilador. Para informar que o programa deve executar várias instruções podemos criar um bloco de instruções usando as chaves { comando1; comando2; }. Agora, podemos resolver o problema acima.

Criando blocos

```c
#include <stdio.h>
int main(){
    float n1, n2, mediaAP;
    printf("Nota 1:");
    scanf("%f", &n1);
    printf("Nota 2:");
    scanf("%f", &n2);
    mediaAP = (n1+n2)/2;
    printf("mediaAP =  %f\n", mediaAP);
    if(mediaAP<4)
        printf("reprovado\n");
    else // media >= 4
        if(mediaAP>=7)
            printf("aprovado\n");
        else {
            // media >= 4 && media < 7
            float n3, mediaF;
            printf("Avaliação Final:");
            scanf("%f", &n3);
            if(n3 < 4)
                printf("reprovado\n");
            else{
                mediaF = (mediaAP + n3)/2;
                if(mediaF < 5)
                    printf("reprovado\n");
                else
                    printf("aprovado\n");
            }
        }
}
```

Uma coisa interessante é que dentro de um bloco podemos declarar novas variáveis. Essas variáveis declaradas em bloco só podem ser usadas dentro do bloco em que elas são declaradas. Podemos criar novos blocos de instruções toda vez que for necessário.

```c
int main(){
    {
        int x;
        x = 3;
        printf("%d\n", x);
    }
    printf("%d\n", x);
}
```

O compilador vai dar um erro um erro na Linha 7 indicando que a variável x não está declarada.

Declarando listas de inteiros

A linguagem C permite que você possa criar várias variáveis que podem ser acessadas por um nome e um valor. Por exemplo,

```c
int L[8]; //L[0],...,L[7]
```

Na linha acima, estamos criando 8 variáveis inteiras que podem ser acessadas usando o nome da variável L e os valores entre 0 e 7.

O programa abaixo declara três variáveis, realiza a leitura e a escrita dessas três variáveis

```c
#include <stdio.h>
int main(){
    int L[3];
    printf("L[0]:");
    scanf("%d", &L[0]);
    printf("L[1]:");
    scanf("%d", &L[1]);
    printf("L[2]:");
    scanf("%d", &L[2]);

    printf("L[0] = %d\n", L[0]);
    printf("L[1] = %d\n", L[1]);
    printf("L[2] = %d\n", L[2]);
}
/*
L[0]:7
L[1]:9
L[2]:6
L[0] = 7
L[1] = 9
L[2] = 6

*/
```

Podemos reescrever o programa acima usando o comando de repetição for.

```c
#include <stdio.h>
int main(){
    int i;
    int L[3];
    for(i=0;i<3;i++){
        printf("L[%d]:", i);
        scanf("%d", &L[i]);
    }

    for(i=0;i<3;i++){
        printf("L[%d] = %d\n", i, L[i]);
    }

}
```

Vamos imaginar que o comando for representa o dedo de uma pessoa que está apontado para regiões de memória, então começamos apontando para a posição 0, 3 representa que só podemos apontar até o 2 e i++ representa que passamos para a próxima posição depois que executamos o bloco de instruções.

A partir de agora, vamos nos concentrar no processamento de lista de números inteiros.

Imprimindo uma lista

```c
#include <stdio.h>
int main(){
    int i;
    int L[7] = {10,12,45,74,78,86,12};

    for(i=0;i<7;i++){
        printf("L[%d] = %d\n", i, L[i]);
```

```
    }

}
```

Perguntas

1. O que acontece quando trocamos 0 por 2?
2. O que acontece quando trocamos 7 por 5?
3. O que acontece quando trocamos 7 por 10?
4. O que acontece quando trocamos i++ por i=i+2?

Encontrando o maior

Imaginemos que usamos um dedo para apontar os elementos da lista, um por um. Inicialmente, criamos uma variável chamada maior para armazenar o maior valor encontrado até o momento. Em seguida, para cada elemento apontado pelo dedo, verificamos se ele é maior que o valor armazenado em maior. Se for, atualizamos maior para esse novo valor. No final do processo, maior conterá o maior elemento da lista.

```c
#include <stdio.h>
int main(){
    int i, maior;
    int L[7] = {10,12,45,74,78,86,12};
    maior = 0;
    for(i=0;i<7;i++){
        if(L[i]>maior){
            maior = L[i];
        }
        printf("maior ate o momento %d\n", maior);
    }
        printf("maior %d\n", maior);
}
```

Como será a saída desse programa? Você acertou se você pensou o seguinte:

```
maior ate o momento 10
maior ate o momento 12
maior ate o momento 45
maior ate o momento 74
maior ate o momento 78
maior ate o momento 86
maior ate o momento 86
maior 86

```

Note que obtemos o valor correto para a lista acima. Contudo, o programa acima não vai funcionar quando utilizamos listas de números negativos.

```c
int L[7] = {-10,-12,-45,-74,-78,-86,-12};
```

Por exemplo, com essa lista acima, o maior elemento será -10. Para evitar esse erro, vamos fazer o seguinte:

```c
maior = L[0];
```

Encontrando o maior e o segundo maior

Para resolver este problema, podemos usar a seguinte estratégia:

- Encontre um valor inicial para maior elemento.
- Encontre o maior elemento da lista e armazene-o na variável `maior`.
- Encontre um valor inicial para o segundo maior.
- Em seguida, identifique o maior elemento da lista que seja diferente de `maior` e armazene-o na variável `maior2`.

```c
#include <stdio.h>
int main(){
    int i, maior, maior2;
    int L[7] = {10,12,45,74,78,86,12};
    //Valor inicial para maior
    maior = L[0];
    //Encontrando o maior elemento
    for(i=0;i<7;i++){
        if(L[i]>maior){
            maior = L[i];
        }
    }
    //Encontrando um valor inicial para maior2
    if(L[0] != maior)
        maior2 = L[0];
    else
        maior2 = L[1];
    //Encontrando o segundo maior
    for(i=0;i<7;i++){
        if(L[i] != maior && L[i] > maior2){
            maior2 = L[i];
        }
    }
    printf("maior %d segundo maior %d\n", maior, maior2);
}
```

Perguntas

1. O valor inicial do maior poderia ser L[4]?
2. Em algum momento, o programa assume que o vetor tem pelo menos dois elementos?
3. O valor inicial do maior2 pode ser qualquer elemento diferente do maior?
4. O que acontece quando a lista possui elementos iguais?

Considere o seguinte programa:

```c
#include <stdio.h>
int main(){
    int i, maior, maior2;
    int L[7] = {10,12,45,74,78,86,12};

    if(L[0]>L[1]){
        maior = L[0];
        maior2 = L[1];
    }else{
```

```
        maior = L[1];
        maior2 = L[0];
    }
    printf("maior %d segundo maior %d\n", maior, maior2);

    for(i=0;i<7;i++){
        if(L[i]>maior){
            maior2 = maior;
            maior = L[i];
        }else if(L[i]>maior2){
            maior2 = L[i];
        }
        printf("maior %d segundo maior %d\n", maior, maior2);
    }

    printf("maior %d segundo maior %d\n", maior, maior2);
}
```

Durante a sua execução, o maior e o segundo maior estão com o mesmo valor. Esse é um comportamento que não queremos que aconteça.

```
maior 12 segundo maior 10
maior 12 segundo maior 10
maior 12 segundo maior 12
maior 45 segundo maior 12
maior 74 segundo maior 45
maior 78 segundo maior 74
maior 86 segundo maior 78
maior 86 segundo maior 78
maior 86 segundo maior 78

```

Perguntas

1. Por que esse problema acontece?
2. Encontre duas maneiras de consertar esse problema?