

Aula 04 - Repetição

Assuntos

1. incremento e decremento
2. for
3. while
4. do... while
5. recursão

incremento e decremento

```
int i = 0;  
printf("%d\n", i);  
i++;  
printf("%d\n", i);  
++i;  
printf("%d\n", i);  
i--;  
printf("%d\n", i);  
--i;  
printf("%d\n", i);
```

incremento e decremento

```
int i = 0;  
printf("%d\n", i++);  
printf("%d\n", ++i);  
printf("%d\n", i--);  
printf("%d\n", --i);  
printf("%d\n", i);
```

incremento e decremento

a ordem do operador altera a ordem em que a operação é executada:

- `i++` usa o valor e depois incrementa
- `++i` incrementa e depois usa o valor

o mesmo é válido para o decremento (`i--` e `--i`)

for

```
int i;  
  
for (i = 0; i < 10; i++){  
    printf("%d ", i);  
}
```

for

```
for (int i = 0; i < 10; i++)  
    printf("%d ", i);
```

Qual a diferença?

```
int i;  
  
for (i = 0; i < 10; i++)  
    printf("%d ", i);  
  
printf("\n%d", i);
```

```
for (int x = 0; x < 10; x++)  
    printf("%d ", x);  
  
printf("\n%d", x);
```


E o continue e break?

```
for (int i = 0; i < 10; i++){  
  
    if (i % 2 == 0)  
        continue;  
    else if (i >= 8)  
        break;  
  
    printf("%d ", i);  
}
```

- o `continue` "pula" a próxima execução do loop
- o `break` quebra a execução do loop

while

```
int i = 0;  
while (i < 10) {  
    printf("%d ", i);  
    i++;  
}
```

while e for

```
int i = 0;  
while (i < 10) {  
    printf("%d ", i);  
    i++;  
}
```

```
int i;  
for (i = 0; i < 10; i++)  
    printf("%d ", i);
```

do... while

```
int i = 0;  
do {  
    printf("%d ", i++);  
} while (i < 10);
```

usado quando queremos garantir que o loop será executado pelo menos uma vez

do...while vs while

```
int i = 0;
while (i < 10 && i > 0) {
    printf("%d ", i++);
}
```

```
int i = 0;
do {
    printf("%d ", i++);
} while (i < 10 && i > 0);
```

Recursão

Para entender recursão, primeiro você precisa entender recursão

Recursão

A recursão acontece quando uma função faz uma chamada para ela mesma.

```
int f(int i){  
    if(i <= 1)  
        return 1;  
    else  
        return f(i-1) + i;  
}
```

Cuidados

Para usar recursão sem grandes problemas, temos que tomar alguns cuidados:

1. Toda função recursiva tem que ter um critério de parada
2. O termo genérico tem que ser atualizado

Se não tivermos esses dois pontos, podemos ter um loop infinito

Cuidados

```
int f(int i){  
    if(i <= 1) // critério de parada  
        return 1;  
    else // chamada recursiva com valor atualizado  
        return f(i-1) + i;  
}
```

Problemas

Para cada chamada recursiva antes de critério de parada, o programa aloca memória para a chamada da próxima função. Portanto, uma função recursiva pode gastar mais memória do que uma função que usa laço de repetição.

Por que usar recursividade

- Em alguns casos, a recursão deixa o código mais simples e mais fácil de manter
- Em algumas linguagens, não existe a opção de laço de repetição e temos que usar recursão

Fatorial: exemplo clássico

```
#include <stdio.h>
int fact(int n){
    if(n <= 1) // critério de parada
        return 1;
    else // chamada recursiva com atualização do valor
        return fact(n-1) * n;
}
int main(){
    int i = 10;
    int x = fact(i);
    printf("%d %d\n", i, x);
}
```

Exemplo da soma

```
// laço de repetição  
int soma = 0;  
for(int i=1; i<n+1; i++)  
    soma += i;
```

```
// recursividade  
int soma(int n){  
    if (n == 0)  
        return 0;  
    else  
        return soma(n-1) + n;  
}
```