Introdução às técnicas básicas de depuração de código (**C**)

Este texto visa apresentar um resumo sobre procedimentos para ajudar na resolução de problemas gerais em tentativas de soluções algoritmicas.

1. Por que devemos saber depurar o código?

Se você já está envolvido com o aprendizado de *programação* há algum tempo, então muito provavelmente já recebeu mensagens de erro, algumas **sintáticas** (quando existe algum erro na *sintaxe* de seu código) mas outras, geralmente mais difíceis de resolver, na *lógica da execução*, neste caso um erro **semântico** (o programa não faz o que era esperado que ele fizesse).

Outra situação desagradável quando é utilizado um ambiente para teste automático de código (como é o caso do uso do *Moodle* com *VPL* ou com o *iTarefa/iVProg*), é o aprendiz conseguir (aparentemente) "executar" com sucesso em seu computador, mas ao passar para o ambiente com o avaliador automático, receber alguma mensagem de erro...

Então o que você faz? Provavelmente olhar todas as linhas do código, mas tudo parece estar como deveria. O código parece correto, "não tem motivo algum para a saída ser diferente da esperada, certamente o sistema do curso tem erro" é um pensamento frequente para o aprendiz.

Esta é uma reação natural, porém nem sempre "lemos" o que está codificado, mas o que pensamos ter codificado, talvez você já tenha passado por isso ao escrever uma redação (no estudo de *lingua portuguêsa*). Nestes casos, o problema é não darmos a devida atenção aos "detalhes", ou pelo código ser complexo, ou eventualmente não examinarmos algumas situações que os *casos-de-teste* estão preparados para testar.

Portanto, seja por falta de atenção, paciência ou complexidade do código, erros são comuns. Então o que devemos fazer para achar e corrigir estes erros?

Primeiro devemos entender a natureza do erro, existem duas grandes categorias, *erro sintático ou erro semântico.*

- **Erro Sintático**: É um erro na sintaxe da linguagem, em sua gramática, como por exemplo, usar letra maiúscula inicial para o *comando de seleção* (o sistema não reconhecerá como "comando de seleção", tentará como uma "atribuição", mas como "atribuição" também não estará correta). Neste caso, seu programa não poderá ser "rodado".
- **Erro semântico**: É um erro na "lógica de seu código", em sua *semântica*, o código está sintaticamente correto, porém não faz o que se esperava dele. Por isso, este tipo de erro é geralmente mais difícil de ser identificado e corrigido. Para isso algumas técnicas básicas, como as apresentadas na seção 3, podem ser de grande valia.

2. Como tratar erros sintáticos?

Como estes erros são acusados pelo próprio compilador são mais fáceis de identificar. Em geral, a mensagem de erro gerada indica a linha onde ocorreu o erro. Então é preciso calma e uma leitura cuidadosa para entender o significado da mensagem.

2.1. Leia as mensagens de erro geradas pelo compilador/interpretador

As mensagens de erro indicam qual o problema encontrado e porque a compilação/execução falhou. Por exemplo, considere o seguinte código errôneo em *C*:

*Tab. 1. Exemplo de código com **erro sintático** (nome errado para o comando de impressão).*

```
#include <stdio.h>
int main (void) {
  prit("Hello World\n"); // aqui tem erro sintatico...
  }
```

A mensagem de erro (ou advertência) recebida ao tentar compilar:

```
main.c:2:1: warning: implicit declaration of function ‘prit’ [-Wimplicit-
function-declaration]
  main.c: undefined reference to 'prit'
```

Examinando as mensagens recebidas, podemos perceber que a linha com a "tentativa" de comando `prit` está com problema. Se prestarmos atenção, notaremos que nessa linha está escrito `prit` ao invés de `printf` para a linguagem *C*.

Infelizmente nem sempre os erros são tão claros e simples como este. Como programação também é uma tarefa de exploração, as vezes utilizamos ferramentas que não estamos habituados, portanto apenas a leitura do erro não é suficiente.

2.2. Procurar na *Internet*

Provavelmente você não é a primeira pessoa a receber esta mensagem de erro, nem será a última, então, se fizer uma busca pela *Web*, talvez encontre uma mensagem relatando erro semelhante e poderá estudar as proposta de correção.

Em geral basta copiar o *mensagem de erro* e colar na barra de pesquisa do seu "buscador" preferido (experimente o duckduckgo.org).

3. Como tratar erros semânticos?

Como são erros de "lógica de programação", então existe ao menos um *conjunto de entradas* para as quais a saída não é a esperada. Isto nos deixa em uma situação que requer mais atenção e nem sempre será uma solução fácil. Muitas vezes requerem reescrita parcial ou total do código.

3.1. Procure *simular* seu código

Se o seu algoritmo não é muito grande, pode-se fazer uma simulação dele para entender exatamente o que está fazendo e com isso identificar o momento que o primeiro erro aparece.

Para simular, construa uma tabela com todas variáveis de seu código (ou apenas aquelas que deseja rastrear), sendo que cada variável terá sua coluna. A cada instrução que altere o valor de determinada variáveis, deve-se registrar o novo valor na coluna correspondente, na linha seguinte à última linha que teve um valor registrado, ou seja, as linhas da tabela indicam a ordem de execução (uma entrada mais acima indica que a instrução que alterou a variável correspondente ocorreu "mais cedo").

ATENÇÃO, é essencial seguir precisamente a ordem de execução dos comandos e deve-se simular/executar **exatamente** o que está redigido (e não como "acha que deveria estar")!

Tab. 2. Exemplo de simulação de um código (sem erro): imprimir a somas dos naturais.

# Codigo	N	soma	i	Impressoes
Explicacoes (por linha)				
1 int N, soma=0, i=0;	?	0	0	
1 : valores iniciais (N desconhecido!)				
2 scanf("%d", &N); // ler e guardar em N	3			
2 : ler valor e guardar em N (supor 3)				
3 while (i < N) {				
3 : 0 < 3 verdadeiro => entra no laco				
4 soma = soma + i;		0		
4 : acumular i em soma				
5 i = i + 1;			1	
5 : acumular 1 em i				
6 }				
6 : final laco, voltar 'a linha 3				
7 printf("fim\n");				
3 : 1 < 3 verdadeiro => entra no laco				
		1		
4 : acumular i em soma				
			2	
5 : acumular 1 em i				
6 : final laco, voltar 'a linha 3				
3 : 2 < 3 verdadeiro => entra no laco				
		3		
4 : acumular i em soma				
			3	
5 : acumular 1 em i				
6 : final laco, voltar 'a linha 3				
3 : 3 < 3 falso => vai para final do laco				
				3
7 : escrever valor em soma				

3.2. Releia o código

Eventualmente existe um erro de lógica que "salte aos olhos", podendo deste modo encontrá-lo rapidamente. Vejamos um caso clássico de erro semântico (erro na "lógica") na linguagem *C*.

*Tab. 3. Um exemplo de **erro semântico** clássico em C.*

```
if (x = 0) // supondo que x chegue aqui com valor 0
  printf("O valor e' nulo\n");
```

Entretanto, a mensagem **não** é impressa! A razão é que em *C* uma *expressão lógica* é uma *expressão aritmética* que é comparada com o valor nulo, se for zero, então o resultado é **falso**, senão é **verdadeiro**. Como a *expressão lógica* que aparece é `x = 0`, na verdade é um *comando de atribuição*, significando que a variável `x` receberá (novamente) o valor nulo e o resultado da *expressão aritmética* é precisamente o zero! Ou seja, o resultado lógico será *falso* e o comando subordinado **não** será impresso!
Assim, correção é simples:

```
if (x == 0) // correcao: usar "=="
  printf("O valor e' nulo\n");
```

Porém, nem sempre é realista reler o código todo, então foque em partes críticas, aquelas partes em que o erro acontece. Geralmente o erro encontra-se em algum comando de **seleção** (`if`), em **condicional de laço** (`while`, `for` ou outros), em **atribuições** ou em **chamadas de função**.

Este método requer maior conhecimento do programador para saber qual a parte crítica do código, elas podem ser diferentes para cada problema. Além disto é necessário um entendimento maior do problema. Então a técnica seguinte pode ajudar.

3.3. Utilizar "bandeiras" (flags)

Nem sempre é possível identificar tudo apenas relendo o código, principalmente em códigos mais complexos. Portanto precisamos de mais ferramentas. Esta técnica ajuda a encontrar as *partes críticas* de seu código e consiste em imprimir algumas variáveis ao longo do código.

Se você não tem a mínima ideia de onde o erro esteja, pode colocar uma impressão de uma lista de variáveis como primeira instrução dentro de cada comando de *seleção* ou em cada comando de *repetição*. No caso de laço infinito (um erro muito comum!), o uso de uma "bandeira" como primeira instrução do *comando de repetição* lhe indicará claramente o problema, *laço infinito*!

Considere o seguinte problema: *Escreva um programa que imprima a frase "hello world" 10 vezes*. Suponha que um colega tenha apresentado a solução *C* da tabela 4.

*Tab. 4. Um exemplo de **erro semântico** clássico: laço infinito.*

```
#include <stdio.h>
int main (void) {
```

```
int i=0;
while (i < 10) {
  printf("hello world\n");
  }
}
```

Ao executar o código, a frase fica sendo impressa indefinidamente (portanto *laço infinito*). Neste caso, qual o melhor local para inserirmos uma bandeira?

Como o problema está relacionado à frase ser impressa muitas vezes, fica natural colocar a *bandeira* como primeira instrução do *comando de repetição*.

Mas qual variável imprimir?

No exemplo acima (tab. 4), não é difícil deduzir, pois o laço usa a variável `i` como controle (e não tem outra...).

Ao fazer isto e executarmos o código percebemos que além da frase, o valor de `i` está sempre com o valor nulo. Portanto o que falta é o incremento na variável de controle, antes de testar a *condição de entrada*.
Ou seja, identificamos que não existe uma atribuição para incrementar a variável de controle.

O que fazer em códigos maiores?

Quando seu código for grande, será necessário identificar cada "bandeira", por exemplo, use algo parecido com: `printf("1: alguma informacao daqui\n");... printf("2: algo daqui\n");...printf("10: algo daqui\n");....`

Mas vale a pena fazer uma análise geral de seu código e da resposta obtida, isso pode indicar um provável local de erro, neste caso, concentre-se neste trecho, colocando uma "bandeira" em cada comando de *seleção* e em todos os inícios de *laços*. Neste caso vale a pena diferenciar as mensagens, por exemplo, com: `printf("se 1: i=%d\n", i); ... printf("laco 1: j=%d\n", j); ... .`

Esta técnica é muito utilizada por ser rápida e efetiva. Mas lembre-se: sempre apague/comente as "bandeiras" depois de utilizá-las, você não quer que a execução do seu código final fique poluída com a impressão de várias "bandeiras".

3.4. Explique o código para alguém:

Esta técnica é conhecida como "Rubber duck debugging" (RDD - "depuração pato de borracha"), que está associada a ideias bastante antigas (como "tente ensinar para aprender") e outras nem tanto, como a técnica "pensamento em voz alta" (*think aloud*). A RDD consiste em explicar, linha por linha, o código para uma outra pessoa, ou na falta de uma pessoa explique para um objeto (como o "pato de borracha").

Pode-se adotar uma abordagem hierárquica, procurando explicar de modo mais geral o que seu programa deveria fazer e depois ir detalhando. Primeiro explique os objetivos do problema, o que você deseja fazer e quais ideias tem para resolvê-lo. Eventualmente pode haver um problema de entendimento de enunciado.
Se o problema estiver nos detalhes, tente explicar cada trecho de seu código e, em cada um, o que que cada linha dele faz.

Ao explicar o trabalho para outra pessoa que não tem a mesma familiaridade com o problema ou com seu código, eventualmente você poderá compreender melhor o fez ou o que deveria fazer.

Leônidas de Oliveira Brandão
http://line.ime.usp.br

Alterações