Depuração & Testes

Algoritmos e Programação de Computadores

Guilherme N. Ramos
gnramos@unb.br

2015/2

Introdução

"*Há duas maneiras de se escrever programas sem erros; apenas a terceira funciona.*"

Alan Perlis

Erros de programação *sempre* existirão [pelo menos enquanto o processo de geração de código for o que conhecemos]...

O desenvolvimento de código [bem feito] segue etapas fases:

1. entendimento/análise do problema,
2. elaboração de um algoritmo,
3. implementação do algoritmo.
4. *depuração*
5. *testes*

Introdução

Origens de erros: especificação, algoritmo, codificação.

"*Se depurar é o processo de remover bugs, então programar deve ser o processo de inserí-los.*"

Edsger W. Dijkstra

Depuração

Depuração

Feita quando se sabe que o programa não funciona (erros de execução, de segmentação de memória), não tem o desempenho desejado, ou simplesmente não tem o comportamento esperado.

Uma das melhores práticas de programação é *realizar pequenas alterações no código e testá-las adequadamente a medida que são feitas.*

Depuração

1. *Teste* o código para descobrir quais problemas existem.
2. *Defina* as condições que o erro pode ser reproduzido.
3. *Encontre* onde no código está a instrução que causa o erro.
4. *Corrija* a instrução;
5. *Verifique* que a correção funciona (com testes).

The GNU Project Debugger

A *depuração* é inevitável... Há diferentes formas de avaliar a execução:

- *pensar* a respeito;
- o bom e velho `printf`;
- depuradores;
- etc.

Antes de consertar um *bug*, é preciso encontrá-lo:

- ao manipular qual variável?
- ao chamar qual função?
- em que linha?

O depurador é um programa que facilita este processo!

The GNU Project Debugger

0-gdb_sigsev.c

```
1 #include <stdio.h>
2
3 int main() {
4     int *ptr = NULL, i = 5;
5
6     ++i;
7     printf("\n i = %d\n", *ptr);
8
9     return 0;
10 }
```

"Tradicional"

```
Segmentation fault (core dumped)
```

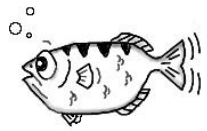
"Depurável"

```
Program received signal SIGSEGV, Segmentation fault.
0x0000000000400548 in main () at sigsegv.c:5
7     printf("\n i = %d\n", *ptr);
```


gdb

The GNU Project Debugger

Permite que se veja o que ocorre "dentro" de um programa durante sua execução – ou o que o programa estava fazendo até o momento que falhou.



O `gdb` oferece várias facilidades para a depuração de programas [compilados com o `gcc`], permitindo:

1. iniciar o programa especificando qualquer coisa que possa afetar seu comportamento;
2. interromper o programa conforme condições específicas;
3. examinar o que aconteceu (quando o programa for interrompido);
4. alterar coisas no programa (para avaliar os efeitos).

gdb

`gdb` é um depurador para diversas linguagens de programação.[1]

- gera informação para depuração [conforme o sistema operacional] para o `gdb` (pode funcionar com outros depuradores. ou não)
- `gdb` aceita otimização (`-O`), mas lembre-se que a isso é coisa do ~~tinhoso~~ compilador.
- `http://www.gnu.org/software/gdb/gdb.html`

Depuração:

```
$ gcc [flags] -g <arquivo> -o <saída>
```

[1] C, C++, D, Go, Objective-C, OpenCL, Fortran, Pascal, Modula-2, Ada

gdb

`gdb` tem uma interface interativa (com histórico, *auto-complete*, etc.)

`help`	é inestimável...
`file`	define o arquivo [executável, compilado com a opção `-g`] a ser depurado
`run`	executa o programa [em depuração]
`kill`	finaliza a execução do programa
`break`	interrompe a execução na linha ou função especificada
`print`	imprime o resultado da expressão
`step/next`	avança a execução (passo a passo)
`continue`	continua a execução
`watch`	interrompe a execução quando o valor da expressão muda
`set`	"avalia expressão e atribui variável"
`backtrace`	mostra o traço de cada elemento na pilha de execução
`quit`	termina o `gdb`

Valgrind

`http://valgrind.org/`

Software livre (GPL2) para depuração. É, na verdade, uma máquina virtual que possibilita a análise dinâmica (*checker*/*profiler*) da execução de programa.

Testes

"*Testes de programas podem ser usados para revelar a existência de erros, mas nunca para mostrar sua ausência!*"

Edsger W. Dijkstra

Testes buscam investigar a qualidade do programa no contexto em que ele deve operar.

"*Em um típico projeto de programação, 50% do tempo e mais de 50% do custo total são gastos em testes do programa ou sistema em desenvolvimento.*"

Myers, Badgett & Sandler

Testes

Idealmente, toda possível execução do programa deveria ser testada, mas como isso é inviável, a qualidade do testes depende da qualidade dos profissionais que definem *o que testar*.

Origem de erros?

- especificação incompleta, errada ou impossível;
- falha(s) na implementação.

Testes de Caixas

Teste de Caixa-Preta

Testar a funcionalidade do programa sem analisar a implementação: análise de pares entrada/saída.Quanto mais abrangentes as entradas, em função das especificações, melhor a qualidade do teste.

```
1 assert(min(1, 2, 3) == 1);
2 assert(min(1, 3, 2) == 1);
3 assert(min(2, 1, 3) == 1);
4 assert(min(2, 3, 1) == 1);
5 assert(min(3, 1, 2) == 1);
6 assert(min(3, 2, 1) == 1);
```

```
1 assert(min(1, 1, 1) == 1);
2 assert(min(1, 1, 2) == 1);
3 assert(min(1, 2, 2) == 1);
4 assert(min(-1, 2, 3) == -1);
5 assert(min(1, -2, 3) == -2);
6 assert(min(1, 2, -3) == -3);
```


Testes de Caixas

Teste de Caixa-Branca

Testar a implementação do sistema: análise de fluxo (processos, decisões e condições).

```
1 /* Implementação */
2 if(A > 1 && B == 0)
3     x /= A;
4 if(A == 2 || x > 1)
5     ++x;
```

```
1 A=2, B=0; /* todos os processos */
2
3 A=3, B=0, X=0; /* todas decisões e */
4 A=2, B=1, X=1; /* todas as condições */
```


Test Driven Development