AULA 2

Lições

The $3n + 1$ Problem

Enunciado

- leitura cuidadosa (das entrelinhas . . .)
- (entrada) supor apenas o que está escrito
  - i não era necessariamente menor que j
  - terminar a leitura com fim de arquivo

```
1    while (scanf("%d %d", &i, &j) == 2)
2            {
3                    [. . . ]
4            }
```

  - várias instâncias
- (saída) seguir as instrução literalmente (número de espaços entre . . .)

"*output* i, j *in the same order in which they appeared*"

Compilação

Colabore: Diga NÃO aos Warnings!

```
gcc -o prog prog.c -Wall -ansi -pedantic -O2
```

```
1   int main(){
2       int i, j;
3       printf("%d", j);
4       return 0;
5   }
```

In function ‘main’:
warning: unused variable ‘i’
warning: ‘j’ might be used uninitialized in this function

Testes

- fazer testes do tipo que sabemos “resolver na mão”

Testes

- fazer testes do tipo que sabemos “resolver na mão”
- testar com o exemplo do enunciado

Testes

- fazer testes do tipo que sabemos “resolver na mão”
- testar com o exemplo do enunciado
- não se iluda com os teste do enunciado

Testes

- fazer testes do tipo que sabemos “resolver na mão”
- testar com o exemplo do enunciado
- não se iluda com os teste do enunciado
- não se iluda com os teste do enunciado

Testes

- fazer testes do tipo que sabemos “resolver na mão”
- testar com o exemplo do enunciado
- não se iluda com os teste do enunciado
- não se iluda com os teste do enunciado
- testar com “casos extremos”

Testes

- fazer testes do tipo que sabemos “resolver na mão”
- testar com o exemplo do enunciado
- não se iluda com os teste do enunciado
- não se iluda com os teste do enunciado
- testar com “casos extremos”
- testar = tentar “quebrar” o programa

Erros

"*Errar é humano, mas colocar a culpa dos próprios erros no outros é mais humano ainda.*"

Anônimo

Erros

"*Errar é humano, mas colocar a culpa dos próprios erros no outros é mais humano ainda.*"

Anônimo

"*Testes mostram a presença de erros, mas não a ausência.*"

Edsger W. Dijkstra

Erros

"*Errar é humano, mas colocar a culpa dos próprios erros no outros é mais humano ainda.*"

Anônimo

"*Testes mostram a presença de erros, mas não a ausência.*"

Edsger W. Dijkstra

"*Iniciantes tendem a colocar a culpa dos erros no compilador, na biblioteca, no mau tempo, . . . Programadores experientes gostariam de ser iniciantes para a ter quem culpar, além deles mesmos . . .*"

The Practice of Programming
Kerningham e Pike

Erros

```
#define DEBUG 1

#if DEBUG
#define debug(x)  x
#else
#define debug(x)
#endif

[...]

debug(printf("Opsss \n ", m));
```

$3n + 1$

Problema: O número de chamadas recursivas da função abaixo é finito para todo inteiro positivo n?

```
int f(n)
1   if (n == 1) return 1;
2   if (n % 2 == 0) return f(n % 2);
3   return f(3 * n + 1);
```

$3n + 1$

Problema: O número de chamadas recursivas da função abaixo é finito para todo inteiro positivo n?

```
int f(n)
1   if (n == 1) return 1;
2   if (n % 2 == 0) return f(n % 2);
3   return f(3 * n + 1);
```

Não se sabe a resposta!

$3n + 1$

n	1	3	15	23	31	510	511	871	937
comp	1	8	18	16	107	49	62	179	174
max	1	16	160	9232	9232	13120	39364	190996	250504

$3n + 1$

n	1	3	15	23	31	510	511	871	937
comp	1	8	18	16	107	49	62	179	174
max	1	16	160	9232	9232	13120	39364	190996	250504

n	1819	10087	100123	100167	100414
comp	162	224	235	341	107
max	1.276.936	2.484.916	19.486.168	41.163.712	50.143.264

Mais testes . . .

limite = (LONG_MAX - 1)/3;

[. . .]

if ($n >$ limite)

[Erro]

else $n = 3 * n + 1$

Mais testes . . .

$\text{limite} =$ (LONG_MAX - 1)/3;

[. . .]

if ($n > \text{limite}$)

[Erro]

else $n = 3 * n + 1$

$\text{limite} = 715.827.882$

Com 432 valores entre 1 e 1 milhão ocorre *overflow*:

113383, 134379, 138367, 151177, . . . , 997601, 997823, 999167

Mais testes . . .

limite = (ULONG_MAX - 1)/3;

[. . .]

if ($n >$ limite)

[Erro]

else $n = 3 * n + 1$

Mais testes . . .

$\text{limite} =$ (ULONG_MAX - 1)/3;

[. . .]

if ($n > \text{limite}$)

[Erro]

else $n = 3 * n + 1$

$\text{limite} = 1.431.655.764$

Com 109 valores entre 1 e 1 milhão ocorre *overflow*:

159487, 212649, 239231, 270271, . . . , 983039, 984623, 997823

Mais testes . . .

$\text{limite} = 2*$(ULONG_MAX/3)$+$ (ULONG_MAX%3)-1;

[. . .]

if ($n > \text{limite}$)

[Erro]

else $n = 3 * (n/2) + 2$

Mais testes . . .

$\text{limite} = 2*$(ULONG_MAX/3)$+$ (ULONG_MAX%3)-1;
[. . .]
if ($n > \text{limite}$)
    [Erro]
else $n = 3 * (n/2) + 2$

$\text{limite} = 2.863.311.529$

Com 77 valores entre 1 e 1 milhão ocorre *overflow*:

159487, 212649, 239231, 270271, . . . , 974079, 984623, 997823,

Inteiros longos

```
#define MAXDIGITS 2
#define BASE 0x8000000

long num[MAXDIGITS];
```

Inteiros longos

```
#define MAXDIGITS 2
#define BASE 0x8000000

long num[MAXDIGITS];
```

TIME LIMIT EXCEEDED . . .

Inteiros longos

```
#define MAXDIGITS 2
#define BASE 0x8000000

long num[MAXDIGITS];
```

TIME LIMIT EXCEEDED . . .

Com tabela para valores pequenos passou.

liiéééééésssss!

Conclusão

Fazer código para testar código parece ser bom.