



Documentação das Mensagens de Erro Reportadas pelo SuSy-avalia

Equipe: Arthur J. Cançado, Felipe R. Jensen,
Gabriell Orisaka, Juliana L. G. Corá, Silvana
Trindade e Yusseli L. M. Mendoza

Instituto de Computação
Universidade Estadual de Campinas

Campinas, Junho de 2017.

Sumário

1	Introdução	2
2	Interpretação das Saídas	2
2.1	Variáveis não inicializadas	2
2.2	Função e Variável não Utilizada	4
2.3	Verificação de Tipo de Dados	6
2.4	Sintaxe Suspeita	10
2.5	Formatação de Caractere	11
2.6	Vazamento de Memória	16
2.7	Ponteiro Nulo	19
2.8	Verificação de Arquivo	21

1 Introdução

Ferramenta para avaliação de código escrito em linguagem C e C++, para ser integrado ao sistema SuSy. A avaliação feita pelo software é conhecida por análise estática, e é utilizada para dar *feedback* ao desenvolvedor sobre possíveis problemas em seu código, tais como vazamento de memória, acesso à posições inválidas de ponteiros e vetores, etc. Este *feedback* é importante para qualquer nível de programador, desde iniciantes até programadores experientes dado à vasta cobertura da análise.

2 Interpretação das Saídas

Nesta seção serão apresentadas exemplos arquivos *.c* e saídas de erros geradas a partir deles.

2.1 Variáveis não inicializadas

As Variáveis não inicializadas podem gerar resultados indesejados, pois a linguagem C não faz inicialização automática das variáveis, sendo assim, o que tem inicialmente é 'lixo'. Já as variáveis não utilizadas geram o desperdício do uso dos recursos.

Variável não inicializada

Mensagem de erro: '[exemplo.c:3]: (erro) Variável não inicializada'.

```
1 void f() {  
2     int a;  
3     a++;  
4 }
```

Pontos a serem percebidos:

- **exemplo.c** representa ao arquivo onde o erro foi encontrado e **3** representa a linha que o erro ocorreu.
- Na sequência é descrito o que motivou o erro a ocorrer, neste caso **Variável não inicializada**.

Memória foi alocada mas não foi inicializada

Mensagem de erro: '[exemplo.c:4]: (erro) Memória foi alocada mas não foi inicializada'.

```
1 void f() {  
2     char *strMsg = "This is a message";  
3     char *buffer=(char*) malloc (128*sizeof(char));  
4     strcpy(strMsg,buffer);  
5     free(buffer);  
6 }
```

Variável foi criada em uma *struct* mas não foi inicializada

Mensagem de erro: '[exemplo.c:2]: (erro) Variável foi criada em uma *struct* mas não foi inicializada'.

```
1 struct AB { int a; int b; };  
2 void f(void) {  
3     struct AB ab;  
4     int a = ab.a;  
5 }
```

2.2 Função e Variável não Utilizada

Variáveis e funções criadas mas nunca utilizadas demandam recursos e impactam na hora da execução, gerando assim o desperdício dos recursos utilizados.

Função não utilizada

Mensagem de erro: '[exemplo.c:1]: (erro) Função criada mas nunca foi utilizada'.

```
1 int f() { printf("Hello World!"); }
```

Variável nunca foi utilizada

Mensagem de erro: '[exemplo.c:8]: (erro) Variável nunca foi utilizada'.

```
1 void foo(int x) {  
2     int i;  
3     if (x) {  
4         int i;  
5     } else {  
6         int i;  
7     }  
8     i = 0;  
9 }
```

Recursos alocados mas nunca utilizados

Mensagem de erro: '[exemplo.c:2]: (erro) Recursos de memória alocados nunca foram utilizados'.

```
1 void foo() {  
2     void* ptr = malloc(16);
```

```
3     free(ptr);
4 }
```

Atribuição de valor à variável mas nunca utilizado

Mensagem de erro: '[exemplo.c:2]: (erro) Atribuição de valor à variável mas nunca utilizado'.

```
1 void foo() {
2     int i = 0;
3 }
```

Variável foi criada em uma *struct* mas não foi utilizada

Mensagem de erro: '[exemplo.c:2]: (erro) Variável foi criada em uma *struct* mas não foi utilizada'.

```
1 struct Point
2 {
3     int x;
4 };
```

Variável não foi atribuída

Mensagem de erro: '[exemplo.c:2]: (erro) Variável não foi atribuída'.

```
1 int foo() {
2     int i;
3     return i;
4 }
```

Label não utilizado

Mensagem de erro: '[exemplo.c:3]: (erro) Label não utilizado'.

```
1 int f(char art) {  
2     switch (art) {  
3         caseZERO:  
4             return 0;  
5         case1:  
6             return 1;  
7         case 2:  
8             return 2;  
9     }  
10 }  
11 // ...
```

2.3 Verificação de Tipo de Dados

Quando o tipo do dado está incorreto, consequentemente o resultado final será incorreto ou indefinido, como por exemplo $\log 0$ que é indefinido.

Acesso à posição inválida

Mensagem de erro: '[exemplo.c:3]: (erro) Acesso à posição inválida de vetor'.

```
1 int i = 0;  
2 int a[i];  
3 a[--i] = 3;
```

Resultado indefinido

Mensagem de erro: '[exemplo.c:3]: (erro) Passando valor que pode gerar resultado indefinido'.

```
1 void foo() {  
2     print(exp(x) - 1);  
3     print(log(1 + x));  
4     print(1 - erf(x));  
5 }
```

Resultado inteiro é atribuído a valor longo

Mensagem de erro: '[exemplo.c:2]: (erro) Resultado inteiro é atribuído a valor longo, podendo ocasionalmente haver perda de informação'.

```
1 long f(int x, int y) {  
2     const long ret = x * y;  
3     return ret;  
4 }
```

Resultado inteiro é retornado como valor longo

Mensagem de erro: '[exemplo.c:2]: (erro) Resultado inteiro é retornado como valor longo, podendo ocasionalmente haver perda de informação'.

```
1 long f(int x, int y) {  
2     return x * y;  
3 }
```


Overflow de variável do tipo *float*

Mensagem de erro: '[exemplo.c:2]: (erro) Overflow de variável do tipo *float*'.

```
1 void f(void) {  
2     return (int)1E100;  
3 }
```

Valor booleano atribuído a ponteiro

Mensagem de erro: '[exemplo.c:2]: (erro) Valor booleano atribuído a ponteiro'.

```
1 void f(char *p) {  
2     if (p+1){}  
3 }
```

Conversão literal de variável para booleano sempre é verdadeiro

Mensagem de erro: '[exemplo.c:2]: (erro) Conversão literal de variável tipo caractere para booleano sempre é verdadeiro'.

```
1 int f() {  
2     if ("Hello") { }  
3 }
```

Vetor acessado em índice inválido

Mensagem de erro: '[exemplo.c:4]: (erro) Vetor acessado em índice inválido, portanto fora do seu limite'.

```
1 int buffer[9];  
2 long int i;
```

```
3 for(i = 10; i--;) {  
4     buffer[i] = i;  
5 }
```

Acesso inválido

Mensagem de erro: '[exemplo.c:3]: (erro) Acesso inválido'.

```
1 void f(unsigned char n) {  
2     int a[n];  
3     a[-1] = 0;  
4     a[256] = 0;  
5 }
```

Acesso a posição inválida do ponteiro

Mensagem de erro: '[exemplo.c:4]: (erro) Acesso à posição inválida do ponteiro'.

```
1 void f() {  
2     char a[10];  
3     char *p = a + 100;  
4 }
```

Possível acesso inválido

Mensagem de erro: '[exemplo.c:3]: (erro) Possível acesso inválido'.

```
1 void foo(char src[100]) {  
2     char * data = alloca(50);  
3     strcat(data, src);  
4 }
```

2.4 Sintaxe Suspeita

Os erros reportados de sintaxe suspeita podem gerar resultados incorretos, entretanto o que deve ser analisado é se realmente o que é gerado um resultado incorreto, sendo assim, cabe ao aluno executar o programa e verificar se de fato a saída gerada sofre impacto pelos erros reportados.

Atribuição redundante

Mensagem de erro: '[exemplo.c:4]: (erro) Atribuição redundante'.

```
1 void f() {  
2     int i[10];  
3     i[2] = 1;  
4     i[2] = 1;  
5 }
```

Condição indefinida

Mensagem de erro: '[exemplo.c:2]: (erro) Condição não está claramente definida'.

```
1 int f() {  
2     if (a & b == c){}  
3 }
```

Comparação de tipo booleano com tipo inteiro

Mensagem de erro: '[exemplo.c:2]: (erro) Comparação de tipo booleano com tipo inteiro'.

```
1 void f(int x, bool y) { if ( x != y ) {} }
```

Condição sempre verdadeira ou falsa

Mensagem de erro: '[exemplo.c:2]: (erro) Condição sempre verdadeira ou falsa'.

```
1 int x = 123;
2 if ( sizeof(char) != x) {}
```

Divisão por zero

Mensagem de erro: '[exemplo.c:4]: (erro) Divisão por zero'.

```
1 int a = 2;
2 int b = 2;
3 // ...
4 b = a / (a-b);
```

2.5 Formatação de Caractere

A formatação incorreta das saídas ou entradas do programa pode gerar a busca por erros não existentes no código por resultar em saídas ou entradas no formato incorreto.

Número de argumentos inválidos - *printf*

Mensagem de erro: '[exemplo.c:4]: (erro) Número de variáveis utilizadas não corresponde com o número de formatadores'.

```
1 void foo() {
2     scanf("%5s", bar);
3     scanf("%5[^~]", bar);
4     scanf("aa%s", bar);
5     scanf("aa%d", &a);
6     scanf("aa%ld", &a);
```

```
7 scanf("%*[^~]");
8 }
```

Scanf com argumento do tipo *string/char** inválido

Mensagem de erro: '[exemplo.c:3]: (erro) Scanf com argumento do tipo *string/char** inválido'.

```
1 void g() {
2     const char c[]="42";
3     scanf("%s", c);
4 }
```

Scanf com argumento do tipo *int* inválido

Mensagem de erro: '[exemplo.c:5]: (erro) Scanf com argumento do tipo *int* inválido'.

```
1 void foo() {
2     int i;
3     unsigned int u;
4     char str[10];
5     scanf("%d", str);
6 }
```

Scanf com argumento do tipo *float* inválido

Mensagem de erro: '[exemplo.c:5]: (erro) Scanf com argumento do tipo *float* inválido'.

```
1 void foo() {
2     int i;
3     unsigned int u;
```

```
4 char str[10];
5 scanf("%f", str);
6 }
```

Argumento do tipo *string/char** no formato apresentado é inválido

Mensagem de erro: '[exemplo.c:3]: (erro) Argumento do tipo *string/char** no formato apresentado é inválido'.

```
1 void foo(char* s, const char* s2, std::string s3, int i) {
2     printf("%s%s", s, s2);
3     printf("%s", i);
4     printf("%i%s", i, i);
5     printf("%s", s3);
6     printf("%s", "s4");
7     printf("%u", s);
8 }
```

Argumento do tipo *number* no formato apresentado é inválido

Mensagem de erro: '[exemplo.c:2]: (erro) Argumento do tipo *number* no formato apresentado é inválido'.

```
1 void foo(const int ci) {
2     printf("%n", ci);
3 }
```

Argumento do tipo ponteiro no formato apresentado é inválido

Mensagem de erro: '[exemplo.c:3]: (erro) Argumento do tipo ponteiro no formato apresentado é inválido'.

```
1 double f() { return 0; }
```

```
2 void foo() {  
3     printf("%p", f());  
4 }
```

Argumento de tipo *int* é inválido neste caso

Mensagem de erro: '[exemplo.c:3]: (erro) Argumento de tipo *int* é inválido neste caso'.

```
1 double f() { return 0; }  
2 void foo() {  
3     printf("%f %d", f(), f());  
4 }
```

Argumento de tipo *unsigned int* é inválido neste caso

Mensagem de erro: '[exemplo.c:3]: (erro) Argumento de tipo *unsigned int* é inválido neste caso'.

```
1 class foo {};  
2 void foo(const int* cpi, foo f, bar b, bar* bp, double d, int i, bool bo) {  
3     printf("%u", f);  
4     printf("%u", "s4");  
5     printf("%u", d);  
6     printf("%u", i);  
7     printf("%u", cpi);  
8     printf("%u", b);  
9     printf("%u", bp);  
10    printf("%u", bo);  
11 }
```

Argumento de tipo *short* inválido

Mensagem de erro: '[exemplo.c:2]: (erro) Argumento de tipo *short* é inválido neste caso'.

```
1 void foo(char c, unsigned char uc, short s, unsigned short us, int i, unsigned int ui,  
    long l, unsigned long ul) {  
2     printf("%hhu %hhu %hhu %hhu %hhu %hhu %hhu %hhu", c, uc, s, us, i, ui, l, ul);  
3 }
```

Argumento de tipo *float* inválido

Mensagem de erro: '[exemplo.c:3]: (erro) Argumento de tipo *float* é inválido neste caso'.

```
1 class foo {};  
2     // ...  
3     printf("%f", (float)cpi);  
4 }
```

Formato de *string/char** modificado não pode ser utilizado

Mensagem de erro: '[exemplo.c:2]: (erro) Formato de *string/char** modificado não pode ser utilizado sem conversão específica'.

```
1 void foo(unsigned int i) {  
2     printf("%I64%i", s, i);  
3 }
```

Tamanho da variável no formato *string/char** é inválida

Mensagem de erro: '[exemplo.c:2]: (erro) Tamanho da variável no formato *string/char** é inválida'.


```
1 void f()  
2 {  
3     char str [8];  
4     scanf ("%70s",str);  
5 }
```

Referenciamento de parâmetro foi realizado incorretamente

Mensagem de erro: '[exemplo.c:3]: (erro) Referenciamento de parâmetro foi realizado incorretamente'.

```
1 void foo() {  
2     int bar;  
3     scanf("%2$d", &bar);  
4     printf("%0$f", 0.0);  
5 }
```

2.6 Vazamento de Memória

Os erros apontados pelo vazamento de memória estão relacionados a alocação e desalocação que podem gerar resultados indesejados ou ambiguidade no código.

Vazamento de recursos

Mensagem de erro: '[exemplo.c:4]: (erro) Vazamento de recursos'.

```
1 void f() {  
2     FILE *fp = fopen("name", "r");  
3     if (!fp) {  
4         fp = fopen("name", "w");  
5         fclose(fp);
```

```
6 }  
7 }
```

Vazamento de memória

Mensagem de erro: '[exemplo.c:3]: (erro) Vazamento de memória'.

```
1 void f() {  
2     int *p = malloc (sizeof(int)*10);  
3     return 1;  
4 }
```

Ponteiro liberado duas vezes

Mensagem de erro: '[exemplo.c:4]: (erro) Ponteiro foi liberado duas vezes'.

```
1 void foo()  
2 {  
3     char *str = malloc(100);  
4     free(str);  
5     free(str);  
6 }
```

Acesso a variável já desalocada

Mensagem de erro: '[exemplo.c:2]: (erro) Acesso a variável já desalocada'.

```
1 void f(char *p) {  
2     free(p);  
3     *p = 0;  
4 }
```

Falta de alocação ou desalocação de memória

Mensagem de erro: '[exemplo.c:2]: (erro) Falta de alocação ou desalocação de memória'.

```
1 void f() {  
2     FILE*f=fopen(fname,a);  
3     free(f);  
4 }
```

Vazamento de memória no processo de realocação

Mensagem de erro: '[exemplo.c:6]: (erro) Vazamento de memória no processo de realocação'.

```
1 static void f() {  
2     char *buf = malloc(10);  
3     if (aa)  
4         // ...  
5     else if (buf = realloc(buf, 100))  
6         // ...  
7     free(buf);  
8 }
```

Liberação Dupla de Memória

Mensagem de erro: '[exemplo.c:6]: (erro) Ponteiro foi liberado duas vezes'.

```
1 void f(char *p) {  
2     if (x) {  
3         free(p);  
4         printf("Hello");  
5     }  
6     free(p);
```

```
7 }
```

Retorno/acesso de variável já desalocada

Mensagem de erro: '[exemplo.c:3]: (erro) Retorno/acesso de variável já desalocada'.

```
1 void f(char *p) {  
2     free(p);  
3     return p;  
4 }
```

2.7 Ponteiro Nulo

Verificação de ocorrência de erros na manipulação de ponteiros irá auxiliar em relação ao acesso de ponteiros que são nulos, e neste o SuSy-avalia faz a cobertura de três erros possíveis, que são os seguintes: aritmética de ponteiros e acesso a ponteiro nulo.

Possível acesso a ponteiro nulo

Mensagem de erro: '[exemplo.c:2]: (erro) Possível acesso a ponteiro nulo se o valor padrão foi utilizado'.

```
1 void f(int *p = 0) {  
2     *p = 0;  
3 }
```

Possível valor de ponteiro ser nulo

Mensagem de erro: '[exemplo.c:3]: (erro) Existe uma possibilidade do valor acessado do ponteiro ser nulo'.

```
1 void foo(char *p) {  
2     // ...  
3     if (!p) { }  
4 }
```

Overflow na aritmética de ponteiro

Mensagem de erro: '[exemplo.c:2]: (erro) Overflow na aritmética de ponteiro, ponteiro nulo é subtraído'.

```
1 void foo(char *s) {  
2     p = s - 20;  
3 }  
4 void bar() { foo(0); }
```

Desreferenciamento de ponteiro nulo

Mensagem de erro: '[exemplo.c:2]: (erro) Desreferenciamento de ponteiro nulo'.

```
1 int foo(int *p) {  
2     *p = 0;  
3     if (!p) {  
4         return 1;  
5     }  
6     return 0;  
7 }
```

2.8 Verificação de Arquivo

Os erros apontados de verificação de arquivo podem garantir a manipulação correta de um arquivo, como por exemplo, o objetivo do arquivo é de escrita e leitura e o aluno abriu para leitura, portanto não irá conseguir fazer nenhuma escrita no arquivo.

Operação de escrita em um arquivo que foi aberto somente para leitura

Mensagem de erro: '[exemplo.c:5]: (erro) Operação de escrita em um arquivo que foi aberto somente para leitura'.

```
1 void foo(FILE*& f) {  
2     f = fopen(name, "r");  
3     fread(buffer, 5, 6, f);  
4     rewind(f);  
5     fwrite(buffer, 5, 6, f);  
6 }
```

Operação de leitura em um arquivo que foi aberto somente para escrita

Mensagem de erro: '[exemplo.c:5]: (erro) Operação de leitura em um arquivo que foi aberto somente para escrita'.

```
1 void foo(FILE*& f) {  
2     f = fopen(name, "w");  
3     fwrite(buffer, 5, 6, f);  
4     rewind(f);  
5     fread(buffer, 5, 6, f);  
6 }
```

Arquivo utilizado que não foi aberto

Mensagem de erro: '[exemplo.c:3]: (erro) Arquivo utilizado que não foi aberto'.

```
1 void foo() {  
2     FILE* f;  
3     fwrite(buffer, 5, 6, f);  
4 }
```

Chamada de função fflush() no stream de entrada

Mensagem de erro: '[exemplo.c:2]: (erro) Chamada de função fflush() no stream de entrada, podendo resultar em comportamento indefinido em sistemas não-linux'.

```
1 void foo() {  
2     fflush(stdin);  
3 }
```