Universidade Federal de Ouro Preto
Departamento de Computação

Boas Práticas para Implementação de Algoritmos em C

Prof. Haroldo Gambini Santos

decom
departamento de computação

Por que C ?

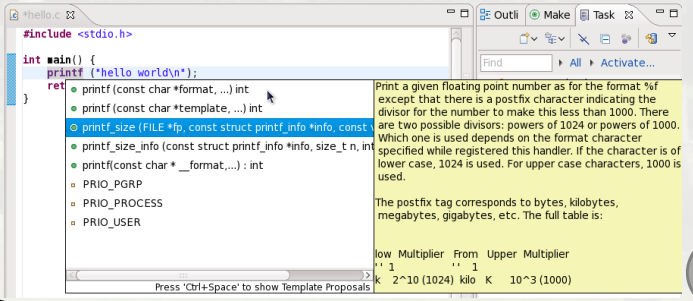
- Baixo nível: força contato com detalhes de implementação de estruturas de dados
- Linguagem com muitas ferramentas de alta qualidade disponível: compiladores, analisadores estáticos de código, ambientes de desenvolvimento, etc.

Padrões

- Implementações: ANSI C (C89 ou C99)
  - compilar com `gcc -std=c89 ou std=c99`
  - remover warnings `-Wall`
- Ferramentas sugeridas:
  - Compiladores que aderem ao padrão ANSI:
    - GCC – GNU Compiler Collection
    - CLANG

Ferramentas Recomendadas

- Ambientes de Desenvolvimento
  - Code::Blocks
  - Eclipse CTD
  - NetBeans



Ferramentas para checagem de código

- Análise estática: CLANG
- uso:

```
if (intervalLength == 0) {
  ② ← Assuming 'intervalLength' is not equal to 0 →
  ③ ← Taking false branch →
    // We have an Accessed event - otherwise, this wouldn't be 0
    score += d->timeFactor(QDateTime::fromTime_t(end)); // like it is oper
} else if (intervalLength >= 4) {
    ④ ← Assuming 'intervalLength' is >= 4 →
  ⑤ ← Taking true branch →
    // Ignoring stuff that was open for less than 4 seconds
    score += d->timeFactor(QDateTime::fromTime_t(end)) * intervalLength /
    ⑥ ← The left expression of the compound assignment is an uninitialized value. The computed
      garbage
}
}
```


Ferramentas para checagem de código

- Análise dinâmica: Valgrind (Linux e OSX) (Dr. Memory no Windows)
- uso: após comp) (-g) rodar valgrind

```
Terminal
File Edit View Terminal Tabs Help
[pwells2@newcell ~/junk]$ valgrind ./memleak
==16738== Memcheck, a memory error detector
==16738== Copyright (C) 2002-2010, and GNU GPL'd, by Julian Seward et al.
==16738== Using Valgrind-3.6.1 and LibVEX; rerun with -h for copyright info
==16738== Command: ./memleak
==16738==
==16738== Invalid write of size 4
==16738==    at 0x400589: main (mem_leak.c:32)
==16738==  Address 0x4c26068 is 0 bytes after a block of size 40 alloc'd
==16738==    at 0x4A0646F: malloc (vg_replace_malloc.c:236)
==16738==    by 0x400505: main (mem_leak.c:17)
==16738==
==16738== Invalid read of size 4
==16738==    at 0x400598: main (mem_leak.c:33)
==16738==  Address 0x4c26068 is 0 bytes after a block of size 40 alloc'd
==16738==    at 0x4A0646F: malloc (vg_replace_malloc.c:236)
==16738==    by 0x400505: main (mem_leak.c:17)
==16738==
==16738==
==16738== HEAP SUMMARY:
==16738==     in use at exit: 410 bytes in 8 blocks
==16738==   total heap usage: 11 allocs, 3 frees, 590 bytes allocated
==16738==
==16738== LEAK SUMMARY:
==16738==    definitely lost: 410 bytes in 8 blocks
==16738==    indirectly lost: 0 bytes in 0 blocks
==16738==      possibly lost: 0 bytes in 0 blocks
==16738==    still reachable: 0 bytes in 0 blocks
==16738==         suppressed: 0 bytes in 0 blocks
==16738== Rerun with --leak-check=full to see details of leaked memory
==16738==
==16738== For counts of detected and suppressed errors, rerun with: -v
==16738== ERROR SUMMARY: 36 errors from 2 contexts (suppressed: 4 from 4)
[pwells2@newcell ~/junk]$
```


Ferramentas para checagem de código

- Análise dinâmica: instrumentar código com AddressSanitizer
- No clang ou gcc incluir opção
  - -fsanitize=address
- Checagem automática de:
  - Acesso fora dos limites
  - Uso depois de free
  - Free duplo
  - Vazamentos de memória

Ferramentas para Análise de Desempenho

- GNU Profiler – gprof
- Usc ados com gpr

```
Flat Profile
File  Code Display  Utility  Help
  %time  cumulative seconds  self seconds  calls  self ms/call  total ms/call  name
   34.2        6.07     6.07       101      60.10         60.10  .xsolve [3]
   32.3       11.79     5.72       101      56.63         56.63  .ysolve [4]
   15.8       14.60     2.81       101      27.82         27.82  .yrhs [5]
   13.6       17.02     2.42       101      23.96         23.96  .xrhs [6]
    2.0       17.38     0.36         1     360.00        410.00  .setup_matrix [7]
    0.7       17.51     0.13                                     .__mcount [8]
    0.5       17.59     0.08     90000       0.00          0.00  .__exp [10]
    0.3       17.64     0.05    180000       0.00          0.00  .pot [11]
    0.2       17.67     0.03         1      30.00        110.00  .init_psi [9]
    0.1       17.68     0.01         4       2.50          2.50  .strcpn [16]
    0.1       17.69     0.01         1      10.00         10.00  .load1 [20]
    0.1       17.70     0.01         1      10.00      17550.00  .qmfd2 [1]
    0.1       17.71     0.01                                     .$SAVEF24 [21]
    0.1       17.72     0.01                                     ._xlfWriteLDChar [22]
    0.1       17.73     0.01                                     .qincrement [23]
    0.0       17.73     0.00        71       0.00          0.00  .xdrmem_getlong [25]
    0.0       17.73     0.00        68       0.00          0.00  .xdr_long [26]
    0.0       17.73     0.00        60       0.00          0.00  .__flsbuf [27]
    0.0       17.73     0.00        60       0.00          0.00  .__flsbuf [28]
    0.0       17.73     0.00        57       0.00          0.00  .leftmost [29]
Search Engine: (regular expressions supported)
```

Padronização da Formatação

- AStyle:
  - A Free, Fast, and Small Automatic Formatterfor C, C++, C++/CLI, Objective C, C#, and Java Source Code

Kernighan & Ritchie

```
int Foo(bool isBar)
{
    if (isBar) {
        bar();
        return 1;
    } else
        return 0;
}
```

Banner

```
int Foo(bool isBar) {
    if (isBar) {
        bar();
        return 1;
        }
else
    return 0;
}
```

GNU

```
int Foo(bool isBar)
{
    if (isBar)
        {
            bar();
            return 1;
        }
    else
        return 0;
}
```

Trabalhos Práticos

- Critérios Principais para Análise
  - Corretude
    - nem todo código que funciona para algumas instâncias está correto !
  - Desempenho
    - Preocupe-se em otimizar o gargalo:
      - Usualmente 99% do processamento fica em uma parte pequena do código (e.g. algum loop aninhado ou chamada recursiva)

Trabalhos Práticos: Material a Entregar

- Código
- Dados usados nos testes (ou gerador de dados)
- Relatório de execução do Valgrind
- Relatório de análise estática do CLang
  - Obs: tanto valgrind quanto clang analyzer podem dar falso positivos para erros
- Relatório técnico escrito em LaTeX descrevendo o(s) algoritmos implementado(s), tabelas com experimentos e gráficos de desempenho analisando tempo de execução e memória
- Gráficos em formato vetorial
- Detalhes sobre o ambiente de hardware e software utilizado no desenvolvimento e experimentos

Estruturas de Dados Básicas - Vetor

- Alocação automática

```
int v[n];
```

C89
#define n 1000
C99
int n;

- Alocação manual

```
int *v = malloc(sizeof(int)*n);
…
free(v);
```

- Redimensionamento

```
int *v = realloc(v,sizeof(int)*n*2);
```

Vetor

- Vantagens
  - Localidade de memória
  - Espaço – somente dados salvos
  - $O(1)$:
    - acesso a uma posição
    - inserção/remoção no final
    - troca de elementos
- Devantagens
  - $O(n)$:
    - inserção/remoção sem ser no final

Matriz

- Alocação automática

```
int m[rows][cols];
```

- Alocação manual

```
int **m = malloc(sizeof(int*)*rows);
for ( int i=0 ; (i<rows) ; ++i )
   m[i] = malloc(sizeof(int)*cols);
…
for ( int i=0 ; (i<rows) ; ++i )
   free(m[i]);
free(m);
```


Matriz

- Alocação manual evitando fragmentação

```
int **m = malloc(sizeof(int*)*rows);
m[0] = malloc(sizeof(int)*rows*cols));
for ( int i=1 ; (i<rows) ; ++i )
   m[i] = m[i-1]+cols;
…
free(m[0]);
free(m);
```


Listas Ligadas

- Elementos com Dados e um ponteiro para o próximo

```
typedef struct list {
   ItemType item;
   struct list *next;
} list;
```

Listas Ligadas

- Inserção no início

```
void insert_list(list **l, ItemType x)
{
   list *p = malloc(sizeof(list));
   p->item = x;
   p->next = *l;
   *l = p;
}
```


Listas Ligadas

- Busca pode ser iterativa ou recursiva

```
list *search_list(list *l, ItemType x)
{
   if (l == NULL) return NULL;
   if (l->item == x)
      return(l);
   else
      return( search_list(l->next, x) );
}
```


Listas

- Vantagens
  - inserção/remoção em qualquer posição mais fácil
  - com registros grandes mover ponteiros é mais fácil que mover os dados
- Desvantagem
  - acesso rápido somente quando sequencial
  - fragmentação de memória

Tipos Abstratos de Dados - TAD

- Coleção de dados
- Especificações TAD dizem o que as operações fazem mas não como fazem
- Implementação de um TAD envolve escolher uma estrutura de dados específica

TAD: Pilha (*Stack*)

- Coleção de dados onde elementos são removidos de acordo com a regra *Last In First Out* (LIFO)
- TAD de pilha (stack)
  - Construtor

```
Stack *stack_create();    // creates a new stack
```

  - Acesso

```
Element stack_top( Stack * );
int stack_size( Stack * );
```

  - Modificação

```
stack_push( Stack *stack, Element element );
Element element stack_pop( Stack *stack );
```

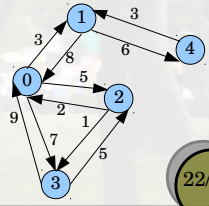

TAD: Grafo

- Conjunto de Vértices e Arestas/Arcos, com ou sem pesos associados
- TAD de Grafo (stack)
  - Construtor

```
Graph *graph_create( int nodes, int arcs, Arc arcs[] );
```

  - Consulta

```
int graph_distance( Graph *graph, int i, int j );
int graph_n_neighbors( Graph *graph, int i );
int *graph_neighbors( Graph *graph );
```



Grafos – Estruturas de Dados

- Matriz de Adjacência

0	3	5	7	INT_MAX
8	0	INT_MAX	INT_MAX	6
2	INT_MAX	0	1	INT_MAX
9	INT_MAX	5	0	INT_MAX
INT_MAX	3	INT_MAX	INT_MAX	0

-

3	1	3	2	5	3	7
2	0	8	4	6		
2	0	2	3	1		
2	0	9	2	5		
1	1	3