ESTGOH
Escola Superior de Tecnologia e Gestão de Oliveira do Hospital
Instituto Politécnico de Coimbra

Engenharia informática

Estruturas de Dados

2019/2020

Relatório

Ficha de ponteiros

David Ferreira 2019141281

António Pereira 2019141051

Índice

1 Introdução

Neste relatório, iremos apresentar a resolução da ficha de trabalho sobre ponteiros, com código e lógica utilizados por nós e devida explicação.

2 Resolução da ficha

2.1 Considere o seguinte conjunto de instruções de um programa:

```
int a = 2, b = 3;
int *p, *q;
p = &a;
q = &b;
```

a) Indique o valor das seguintes expressões apresentadas:

i. **p==&a ->** verifica se o valor de p é o endereço de a. Confirma-se, pois como podemos ver nos dados, p = &a, logo, é verdade.

ii. ***p - *q ->** faz a subtração do valor apontado por p (2) com o valor apontado por q (3), logo, o resultado dessa subtração será -1.

iii. ****&p ->** *&p indica o endereço apontado pelo endereço de p. **&p indica o valor apontado pelo endereço que &p está a apontar, logo, será 2. Caso fosse apenas *&p, daria o valor do endereço de a.

iv. ***p+1 ->** soma ao valor apontado por p, 1, logo, como o valor apontado por p é 2, 2+1=3.

v. ***(q-2) ->** o resultado é 1, pois é o valor apontado por q (3) menos 2.

b) Teste e analise o resultado da seguinte instrução:

```
printf("%p %u %u %d %d %d %d\n", p, p, &p,*p+4,**&p,5**p,**&p+6);
```

Como pedido pela professora, fomos testar no nosso IDE o resultado da instrução de cima. O resultado foi o seguinte:

```
0061FEB8 6422200 6422192 6 2 10 8
```

Como podemos ver, temos vários resultados diferentes. Começando pelo primeiro, que é o valor de **p** mas apresentado de forma diferente. Como podemos ver na instrução, para esse p, temos **%p**, que irá apresentar o valor do p em hexadecimal. O segundo resultado apresenta-nos o valor de **p** na forma unsigned int, ou seja, sem números negativos a representarem o valor. O terceiro resultado apresenta o valor de **p** na forma **%d**, ou seja, no seu valor decimal. O quarto resultado faz a soma do valor apontado por **p**, que já se sabe que é 2, com 4, logo, 4+2=6, está explicado o resultado. O quinto resultado, é o valor apontado pelo endereço de **p.** Ou seja, ***&p** aponta para o endereço de **p.** Este, por sua vez, aponta para o valor do endereço de a. Logo, como a=2, **&p = 2. O sexto resultado, faz a multiplicação de 5 pelo valor apontado por **p**. Como ***p = 2**, 5 * 2 = 10, estando assim explicado o resultado. Por fim, já se sabe que ****&p** = 2, logo, ****&p** + 6 = 8. Estão assim explicados os 7 resultados diferentes daquela instrução.

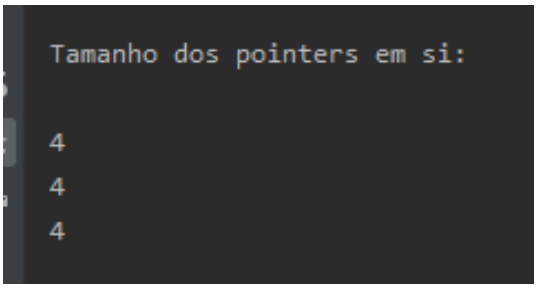
2.2 Elabore um programa que mostre o número de bytes ocupados por cada tipo de variável (char, int, float), assim como o tamanho (em bytes) de um ponteiro para cada um desses tipos. NOTA: Use o operador sizeof().

Para este exercício, o programa elaborado foi o seguinte:

```c
#include <stdio.h>

int main()
{
 int *p;
 char *m;
 float *j;
 printf("\n\nTamanho dos valores apontados pelos pointers: \n\n");
 printf("%d\n",sizeof(*p));
 printf("%d\n",sizeof(*j));
 printf("%d\n\n",sizeof(*m));
 printf("\n\nTamanho dos pointers em si: \n\n");
 printf("%d\n",sizeof(p));
 printf("%d\n",sizeof(j));
 printf("%d\n",sizeof(m));

}
```

Para este programa, o respetivo output (dividido em 2 partes) é o seguinte:

```
Tamanho dos valores apontados pelos pointers:

4
4
1
```



Assim, podemos concluir que o tamanho dos valores apontados pelos pointers variam de acordo com o tipo de pointer que é (char, int, float etc..), enquanto os pointers em si, têm sempre 4 bytes de tamanho.

2.3

2.4 Usando ponteiros, elabore subprogramas que manipulem uma matriz de reais com dimensão 20*20:

a) Inicializar a matriz com o –1:

Função utilizada para resolver esta alínea:

```
//alinea a
void InicializaMatrizA(float * pm)
{
  for(int i=1;i<=400;i++)
  {
    *pm = -1;
    ++pm;
  }
}
```

Chamada da função no main:

```
int main()
{
  int i, j;
  float * pm1;
  float * pm2;
  float matriz1[20][20];
  float matriz2[20][20];
  pm1 = matriz1[0];
  pm2 = matriz2[0];
  InicializaMatrizA(pm1);
  for(int k=0;k<20;k++)
  {
    printf("\n");
    for(int x=0;x<20;x++)
    printf("%.0f ", matriz1[k][x]);
  }
```

O que a função irá fazer, é apresentar a matriz de modo a que todos os elementos da matriz sejam -1. Como podemos ver no main, o pointer * pm1 é inicializado, e, mais em baixo, está a apontar para o primeiro elemento da matriz (matriz1[0]). Na função, há um ciclo for que percorre os elementos todos da matriz, de 1 a 400 (inclusive). Durante esse ciclo, ao valor apontado por pm (*pm) é dado o valor de -1. Com o ++pm, o pointer irá avançar para o próximo elemento e dar-lhe o respetivo valor de -1, fazendo isto para os 400 elementos do vetor.

Output da matriz após esta função é o seguinte:

```
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
```

b) Inicializar a matriz com valores de 1 a 400 por ordem crescente das linhas e das colunas

Esta alínea é bastante parecida com a alínea a), a diferença é que em vez de -1, os valores serão os números de 1 a 400 por ordem crescente dos elementos. Assim, a única mudança relativamente à função da alínea anterior, é o valor apontado por **pm**:

```
//alinea b
void InicializaMatrizB(float * pm)
{
  for(int i=1;i<=400;i++)
  {
    *pm = i;
    ++pm;
  }
}
```

Output desta função é o seguinte:

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40
41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60
61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80
81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100
101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120
121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140
141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160
161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180
181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200
201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220
221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240
241 242 243 244 245 246 247 248 249 250 251 252 253 254 255 256 257 258 259 260
261 262 263 264 265 266 267 268 269 270 271 272 273 274 275 276 277 278 279 280
281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296 297 298 299 300
301 302 303 304 305 306 307 308 309 310 311 312 313 314 315 316 317 318 319 320
321 322 323 324 325 326 327 328 329 330 331 332 333 334 335 336 337 338 339 340
341 342 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359 360
361 362 363 364 365 366 367 368 369 370 371 372 373 374 375 376 377 378 379 380
381 382 383 384 385 386 387 388 389 390 391 392 393 394 395 396 397 398 399 400
```

c) Colocar numa posição da matriz (i , j) o valor indicado pelo utilizador

Nesta função, o objetivo era o utilizador meter um valor à sua escolha para trocar por um valor qualquer em qualquer posição da matriz. Para isso, definimos a seguinte função:

```c
void InserirValor(float * pm, int i, int j)
{
    float numeroUtilizador;
    scanf("%f", &numeroUtilizador);
    for(int k=0;k<20*i+j;k++)
    ++pm;
    *pm = numeroUtilizador;
}
```

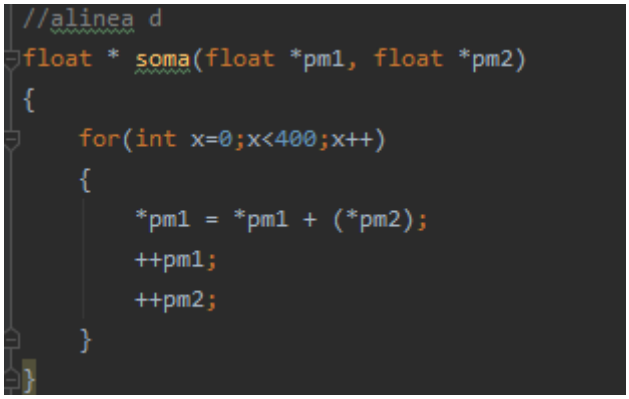
Esta função tinha a mesma lógica que as duas anteriores, no entanto, n seria a percorrer o ciclo inteiro, mas sim alterar em apenas uma posição específica. Assim, o utilizador mete os índices que pretende e o ponteiro será incrementado até chegar à posição desejada. A partir daí, o valor apontado pelo ponteiro **pm,** será introduzido nesse lugar. Exemplo de output desta função:

```
Escolha a linha e a coluna respetivamente:
18
17
Insira o numero que pretende inserir:
111
```

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40
41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60
61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80
81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100
101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120
121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140
141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160
161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180
181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200
201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220
221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240
241 242 243 244 245 246 247 248 249 250 251 252 253 254 255 256 257 258 259 260
261 262 263 264 265 266 267 268 269 270 271 272 273 274 275 276 277 278 279 280
281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296 297 298 299 300
301 302 303 304 305 306 307 308 309 310 311 312 313 314 315 316 317 318 319 320
321 322 323 324 325 326 327 328 329 330 331 332 333 334 335 336 337 338 339 340
341 342 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359 360
361 362 363 364 365 366 367 368 369 370 371 372 373 374 375 376 377 111 379 380
381 382 383 384 385 386 387 388 389 390 391 392 393 394 395 396 397 398 399 400
Process finished with exit code 0
```

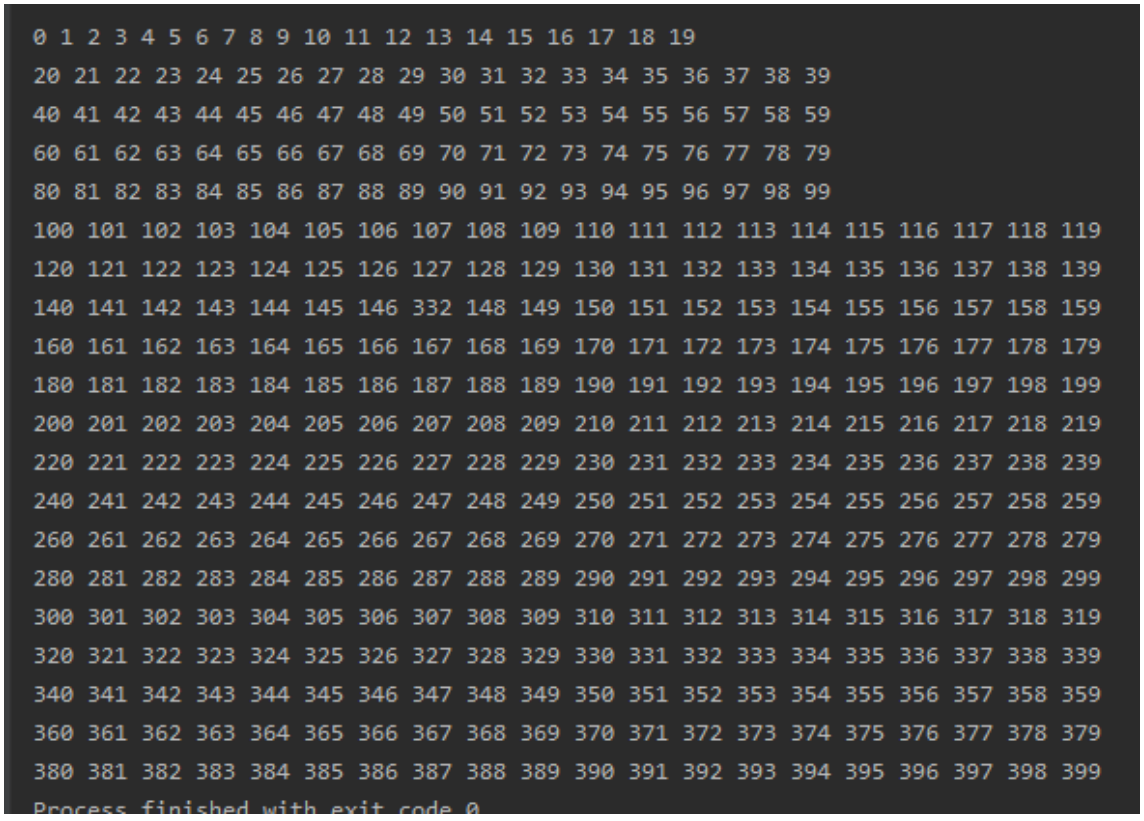
d) Adicionar duas matrizes

A ideia desta função é fazer a soma de duas matrizes. Seguimos a recomendação da professora e decidimos somar as matrizes que já tínhamos. Para tal, seguindo a mesma lógica que temos utilizado, fizemos a seguinte função:



```
//alinea d
float * soma(float *pm1, float *pm2)
{
    for(int x=0;x<400;x++)
    {
        *pm1 = *pm1 + (*pm2);
        ++pm1;
        ++pm2;
    }
}
```

Output desta função, na soma das matrizes “matriz1” e “matriz2”, é o seguinte:



```
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39
40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59
60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79
80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99
100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119
120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139
140 141 142 143 144 145 146 332 148 149 150 151 152 153 154 155 156 157 158 159
160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179
180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199
200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219
220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239
240 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255 256 257 258 259
260 261 262 263 264 265 266 267 268 269 270 271 272 273 274 275 276 277 278 279
280 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296 297 298 299
300 301 302 303 304 305 306 307 308 309 310 311 312 313 314 315 316 317 318 319
320 321 322 323 324 325 326 327 328 329 330 331 332 333 334 335 336 337 338 339
340 341 342 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359
360 361 362 363 364 365 366 367 368 369 370 371 372 373 374 375 376 377 378 379
380 381 382 383 384 385 386 387 388 389 390 391 392 393 394 395 396 397 398 399
Process finished with exit code 0
```

2.5 Usando ponteiros, elabore novas versões para as funções

a) strlen

A função strlen, como se sabe, indica ao utilizador o comprimento de certa string. O nosso objetivo, é refazer esta função, utilizando os seguintes parâmetros: **int tamanho_str(char *ps)**.

Assim, a função que nós fizemos foi a seguinte:

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
int tamanho_str(char *ps)
{
    int count = 0;
    while(*ps!='\0')
    {
        ++ps;
        count ++;
    }
    return count;
}
```

O seu funcionamento é simples de entender. Criámos uma variável count com o valor de 0 inicial. De seguida, utilizámos um ciclo while, enquanto o valor (neste caso, caratere) apontado por **ps** for diferente de **'\0'** (valor no final de todas as strings), o ponteiro anda uma posição para a frente e enquanto andar, incrementa a variável count. Depois, para não perdermos o valor de count, retornamos o próprio. A chamada da função no main funciona da seguinte maneira:

```c
int main()
{
    char ps[200];
    int i, len;
    printf("Insira a palavra que quer contar: ");
    scanf("%s", ps);
    *ps = ps[0]; //inicializar o ponteiro no primeiro elemento da string
    int count = tamanho_str(ps); //igualar a variável count ao valor retornado na função, em cima
    tamanho_str(ps);
    printf("Tamanho da palavra %s: %d", ps, count);
```

Exemplo de output para a string “estgoh”, que contém 6 carateres:

```
Insira a palavra que quer contar:estgoh
 Tamanho da palavra estgoh: 6
```

b) strncpy (usado para copiar parte de uma string)

Para esta alínea, o objetivo era também recriar uma função, neste caso, a função “strncpy”, com os seguintes parâmetros: **char * copia_substr(char * src, int i, int len)**.

A função que definimos foi esta:

```
char * copia_substr(char * src, int i, int len)
{
    char ps[200];
    char * aux = malloc(len);
     src = &src[i];
     int x=0;
     while(x<len)
     {
         ps[x] = *src;
         ++src;
         x++;
     }
     ps[x]='\0';
     strcpy(aux, ps);
    return aux;
}
```

Começámos por criar uma variável **ps** do tipo char, uma string. Após isso, criámos um ponteiro aux, que servirá de auxílio e abrimos um espaço na memória de acordo com o tamanho da string que o utilizador decidir colocar. Depois, simplesmente apontámos o ponteiro **src** para o índice que o utilizador indicou para começar a copiar. Após isso, fizemos um ciclo while para armazenar no vetor **ps**, o valor apontado por **src,** incrementando este para ir armazenando nos seguintes elementos de **ps**. No fim do ciclo, como é uma string, definimos o valor do último elemento do **ps**, como **‘\0’**. Depois, foi só armazenar o valor de **ps** na variável auxiliar, utilizando a função “strcpy” e retornar a string **aux**.

3 Conclusão

Este trabalho foi bastante útil para nos familiarizarmos com os pointers, um "bicho-papão" do C para muita gente, mas que acho que conseguimos entender bem a lógica. Achamos que fizemos um trabalho bastante decente, completámos todos os exercícios propostos, portanto, podemos dizer que foi um bom trabalho e esperemos que estes conhecimentos cimentados sejam úteis para o nosso futuro.

4 Referências

PDFs disponibilizados pela professora e dúvidas tiradas à mesma.