Complementos de Programação em C

1ª edição: Março/2002

RICARDO J. MACHADO

Email: rmac@dsi.uminho.pt
URL: http://www.dsi.uminho.pt/~rmac

Universidade do Minho

Departamento de Sistemas de Informação

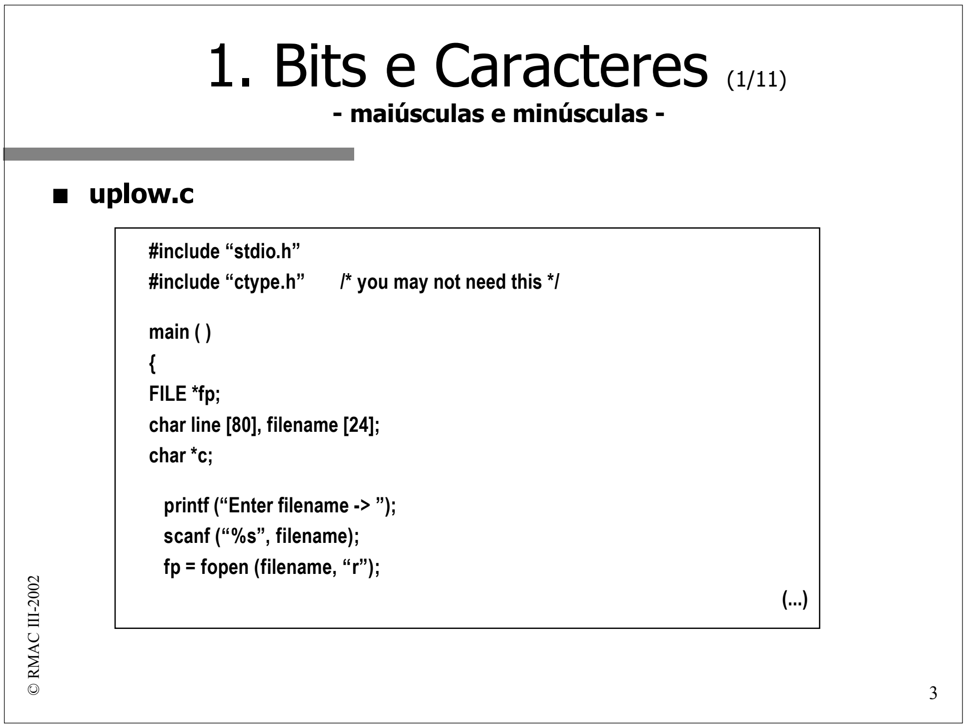
Sumário

1. Bits e Caracteres

2. ArgC e ArgV

3. Estruturas e Uniões

4. Alocação Dinâmica



1. Bits e Caracteres (1/11)

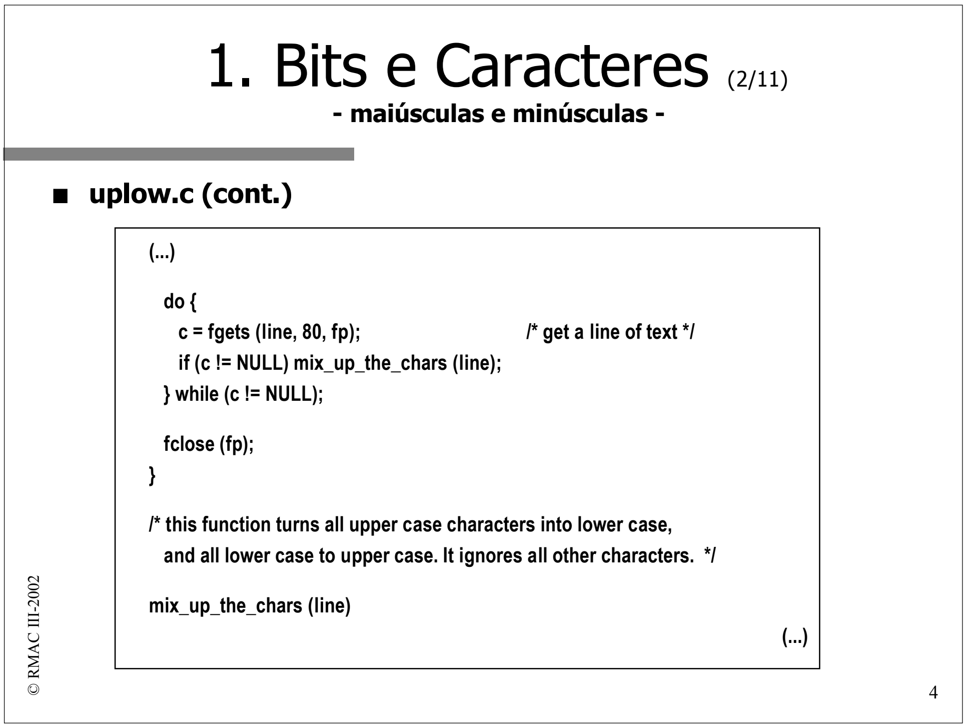
- maiúsculas e minúsculas -

- **uplow.c**

```
#include “stdio.h”
#include “ctype.h”       /* you may not need this */

main ( )
{
FILE *fp;
char line [80], filename [24];
char *c;

  printf (“Enter filename -> ”);
  scanf (“%s”, filename);
  fp = fopen (filename, “r”);
                                                        (...)
```



1. Bits e Caracteres (2/11)

- maiúsculas e minúsculas -

- **uplow.c (cont.)**

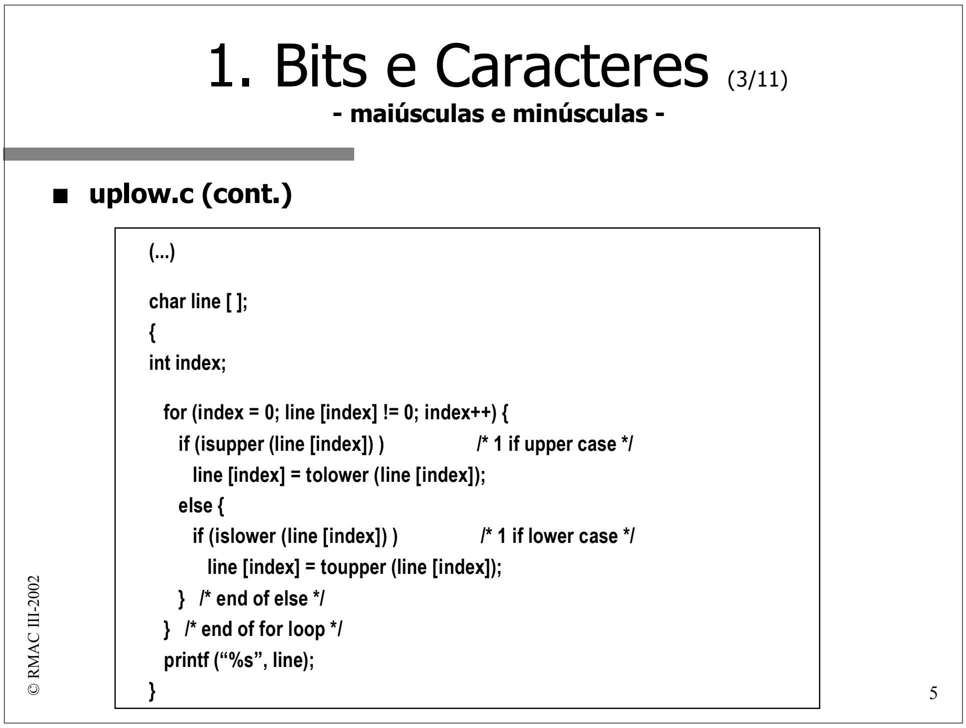
```
(...)

  do {
    c = fgets (line, 80, fp);                    /* get a line of text */
    if (c != NULL) mix_up_the_chars (line);
  } while (c != NULL);

  fclose (fp);
}

/* this function turns all upper case characters into lower case,
  and all lower case to upper case. It ignores all other characters.  */

mix_up_the_chars (line)
                                                        (...)
```



1. Bits e Caracteres (3/11)

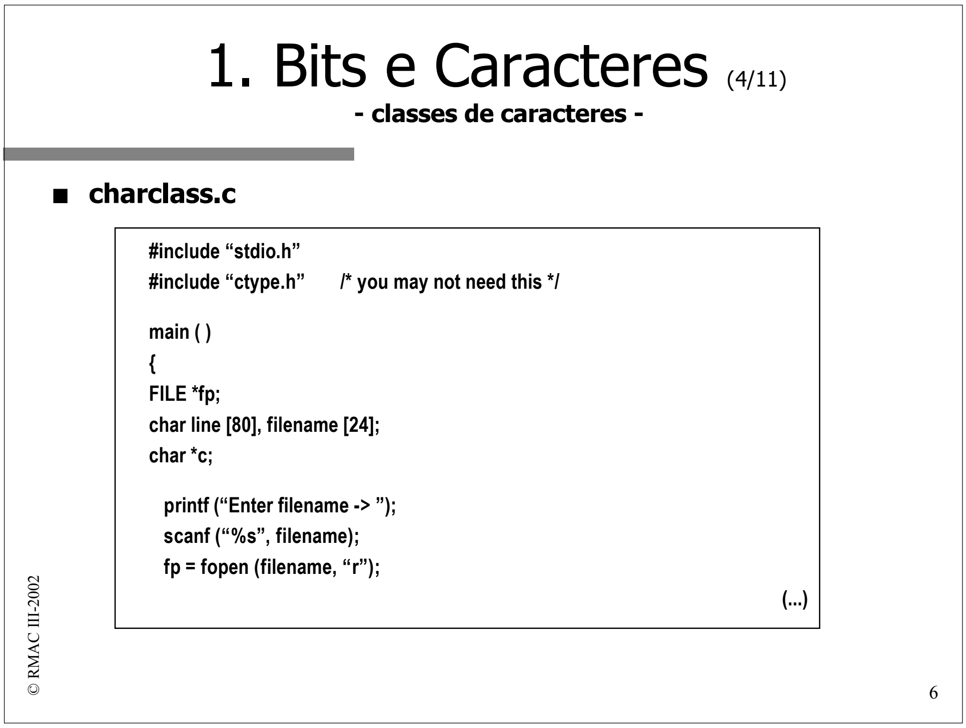
- maiúsculas e minúsculas -

- **uplow.c (cont.)**

```
(...)

char line [ ];
{
int index;

  for (index = 0; line [index] != 0; index++) {
    if (isupper (line [index]) )               /* 1 if upper case */
      line [index] = tolower (line [index]);
    else {
      if (islower (line [index]) )              /* 1 if lower case */
        line [index] = toupper (line [index]);
    }  /* end of else */
  }  /* end of for loop */
  printf (“%s”, line);
}
```



1. Bits e Caracteres (4/11)

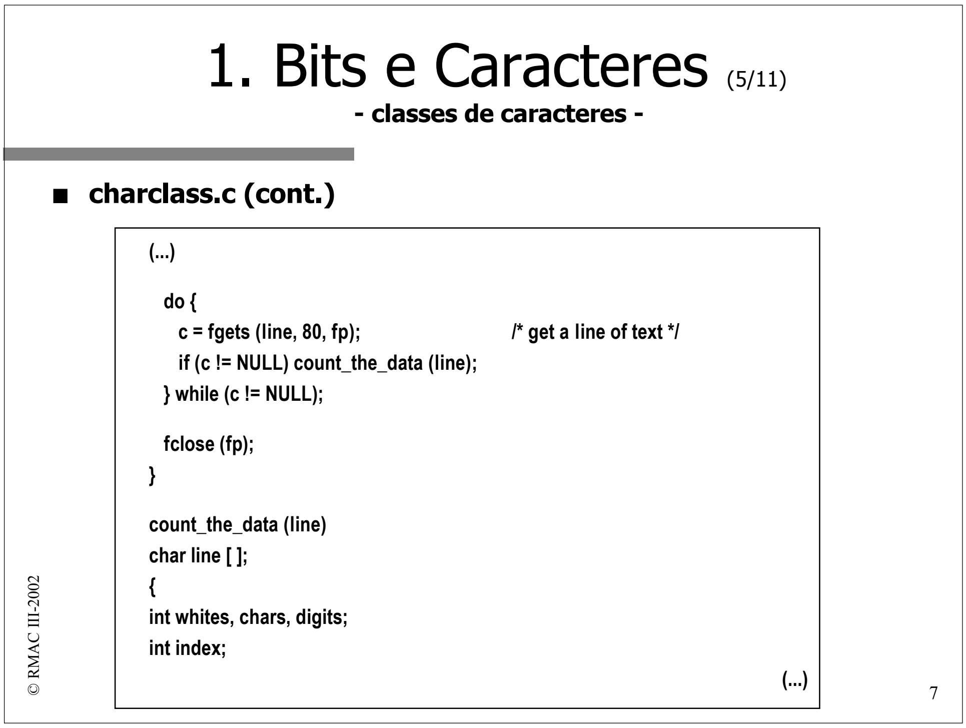
- classes de caracteres -

- **charclass.c**

```
#include “stdio.h”
#include “ctype.h”      /* you may not need this */

main ( )
{
FILE *fp;
char line [80], filename [24];
char *c;

  printf (“Enter filename -> ”);
  scanf (“%s”, filename);
  fp = fopen (filename, “r”);
                                                    (...)
```



1. Bits e Caracteres (5/11)

- classes de caracteres -

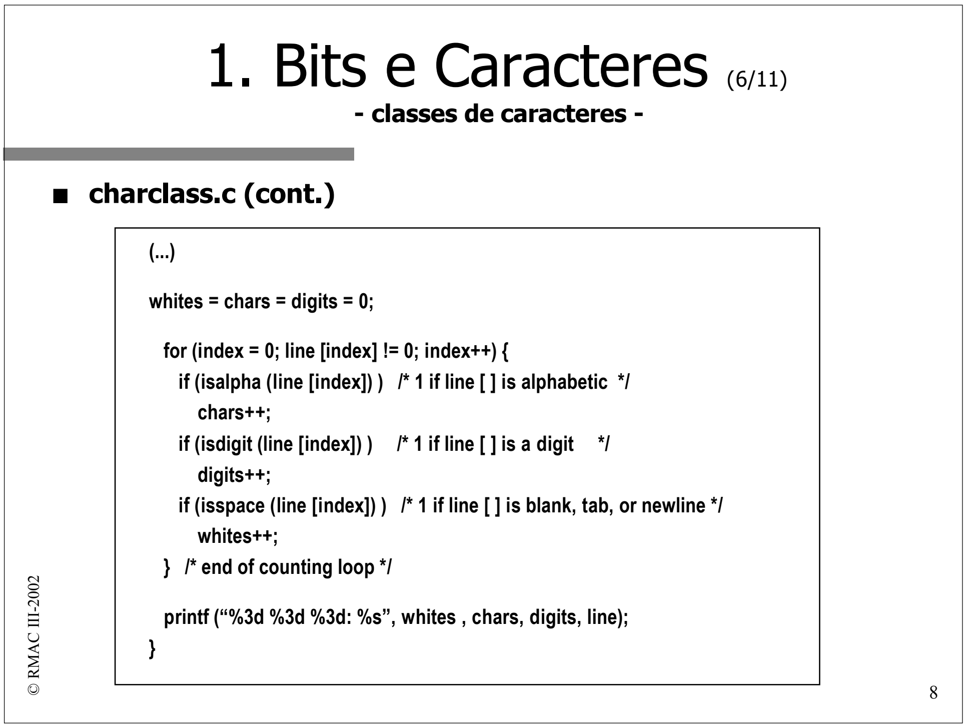
- **charclass.c (cont.)**

```
(...)

   do {
      c = fgets (line, 80, fp);                    /* get a line of text */
      if (c != NULL) count_the_data (line);
   } while (c != NULL);

   fclose (fp);
}

count_the_data (line)
char line [ ];
{
int whites, chars, digits;
int index;
                                                                    (...)
```



1. Bits e Caracteres (6/11)

- classes de caracteres -

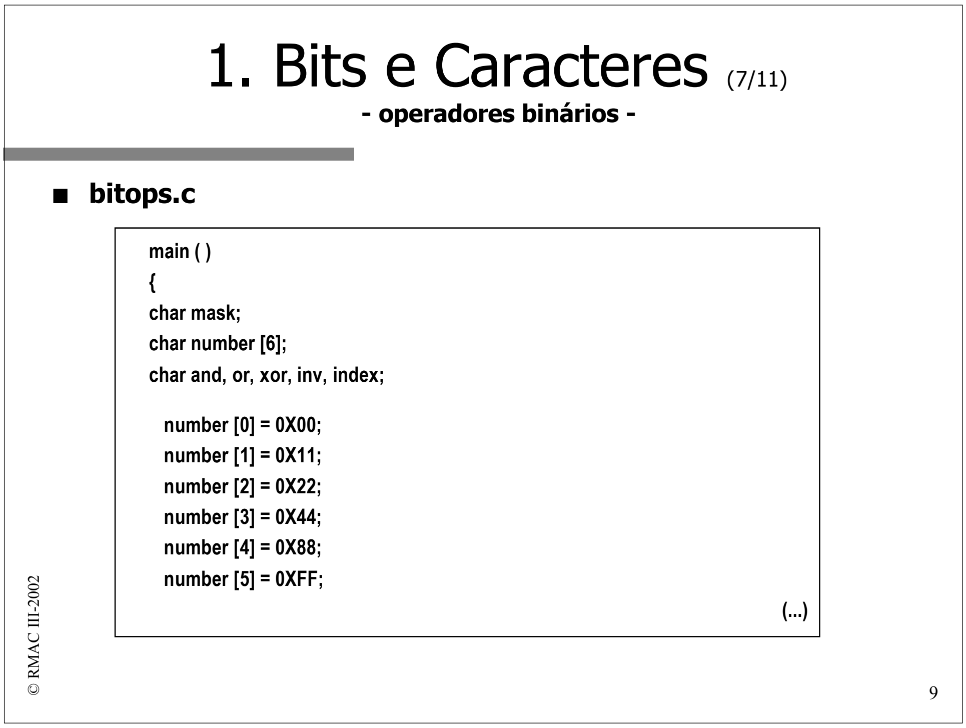
- **charclass.c (cont.)**

```
(...)

whites = chars = digits = 0;

   for (index = 0; line [index] != 0; index++) {
      if (isalpha (line [index]) )   /* 1 if line [ ] is alphabetic  */
         chars++;
      if (isdigit (line [index]) )    /* 1 if line [ ] is a digit     */
         digits++;
      if (isspace (line [index]) )   /* 1 if line [ ] is blank, tab, or newline */
         whites++;
   }   /* end of counting loop */

   printf ("%3d %3d %3d: %s", whites , chars, digits, line);
}
```



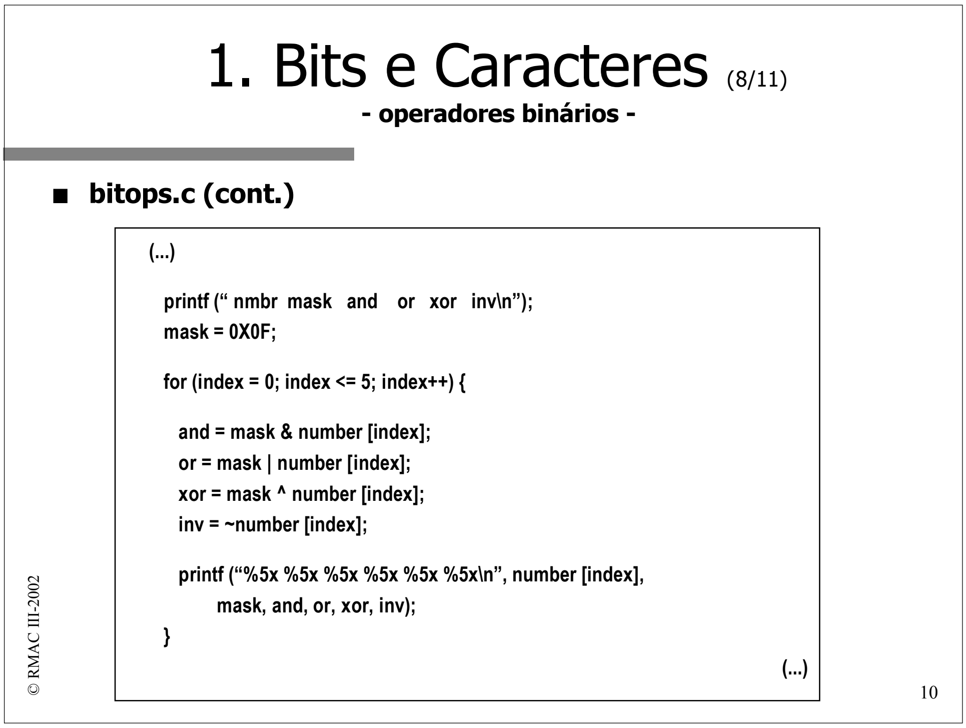
1. Bits e Caracteres (7/11)

- operadores binários -

- **bitops.c**

```
main ( )
{
char mask;
char number [6];
char and, or, xor, inv, index;

  number [0] = 0X00;
  number [1] = 0X11;
  number [2] = 0X22;
  number [3] = 0X44;
  number [4] = 0X88;
  number [5] = 0XFF;
                                                    (...)
```



1. Bits e Caracteres (8/11)

- operadores binários -

- **bitops.c (cont.)**

```
(...)

  printf (“ nmbr  mask   and    or   xor   inv\n”);
  mask = 0X0F;

  for (index = 0; index <= 5; index++) {

    and = mask & number [index];
    or = mask | number [index];
    xor = mask ^ number [index];
    inv = ~number [index];

    printf (“%5x %5x %5x %5x %5x %5x\n”, number [index],
           mask, and, or, xor, inv);
  }
                                                    (...)
```

1. Bits e Caracteres (9/11)

- operadores binários -

- **bitops.c (cont.)**

```
(...)

  printf (“\n”);
  mask = 0X22;

  for (index = 0; index <= 5; index++) {

    and = mask & number [index];
    or = mask | number [index];
    xor = mask ^ number [index];
    inv = ~number [index];

    printf (“%5x %5x %5x %5x %5x %5x\n”, number [index],
          mask, and, or, xor, inv);
  }  /* end of for loop */
}  /* end of main */
```

1. Bits e Caracteres (10/11)

- deslocamentos -

- **shifter.c**

```
main ( )
{
int small, big, index, count;

  printf (“ shift left      shift right\n\n”);
  small = 1;
  big = 0x4000;

  for (index = 0; index < 17; index++) {
    printf (“%8d %8x %8d %8x\n”, small, small, big, big);
    small = small << 1;
    big = big >> 1;
  }
                                                        (...)
```

1. Bits e Caracteres (11/11)

- deslocamentos -

- **shifter.c (cont.)**

```
(...)

  printf (“\n”);
  count = 2;
  small = 1;
  big = 0x4000;

  for (index = 0; index < 9; index++) {
    printf (“%8d %8x %8d %8x\n”, small, small, big, big);
    small = small << count;
    big = big >> count;
  }  /* end of for loop */
}  /* end of main */
```

tente responder

exercícios

1. **Implemente em C as funções “isupper ( )”, “islower ( )”, “toupper ( )”, “tolower ( )”, “isalpha ( )”, “isdigit ( )”, “isspace ()”.**
2. **Reproduza, no papel, o conteúdo dos écrans produzidos pelos programas “bitops.c” e “shifter.c”.**

2. ArgC e ArgV (1/3)

- argumentos em linha de comando -

- **list.c**

```
/* This program will read in any text file and list it on the      */
/* monitor with line numbers and with page numbers.                 */

#include “stdio.h”                 /* standard I/O header file */
#include “io.h”                    /* file I/O prototypes      */

void open_file (int no, char *name);
void open_print_file (void);
void print_a_line (void);
void top_of_page (void);

#define MAXCHARS 255           /* maximum size of a line */
FILE *file_point;              /* pointer to file to be read */
FILE *print_file_point;        /* pointer to printer */
char oneline [256];            /* input string buffer area */
                                                              (...)
```

2. ArgC e ArgV (2/3)

- argumentos em linha de comando -

- **list.c (cont.)**

```
(...)

main (number, name)
int number;                 /* argc: number of arguments on command line */
char *name [ ];             /* argv: arguments on the command line          */
{
char *c;                    /* variable to indicate end of file */
char *point;

  point = name [1];
  open_file (number, point);      /* open the file to read and print */
  open_print_file ( );
                                                              (...)
```

2. ArgC e ArgV (3/3)

- argumentos em linha de comando -

- **list.c (cont.)**

```
(...)

  do {
    c = fgets (oneline, MAXCHARS, file_point);     /* read one line */
    if (c != NULL) print_a_line ( );               /* print the line */
  } while (c != NULL);                             /* continue until EOF */

  top_of_page ( );                  /* move paper to top of page */
  fclose (file_point);              /* close read file */
  fclose (print_file_point);        /* close printer file */
}
```

3. Estruturas e Uniões (1/19)

- estruturas -

- **struct1.c**

```
main ( )
{
struct {
  char initial;    /* last name initial        */
  int age;         /* childs age               */
  int grade;       /* childs grade in school */
  } boy, girl;

  boy.initial = ‘R’;
  boy.age = 15;
  boy.grade = 75;
                                                    (...)
```

3. Estruturas e Uniões (2/19)

- estruturas -

- **struct1.c (cont.)**

```
(...)

   girl.age = boy.age - 1;  /* she is one year younger */
   girl.grade = 82;
   girl.initial = ‘H’;

   printf (“%c is %d years old and got a grade of %d\n”,
           girl.initial, girl.age, girl.grade);

   printf (“%c is %d years old and got a grade of %d\n”,
           boy.initial, boy.age, boy.grade);
}
```

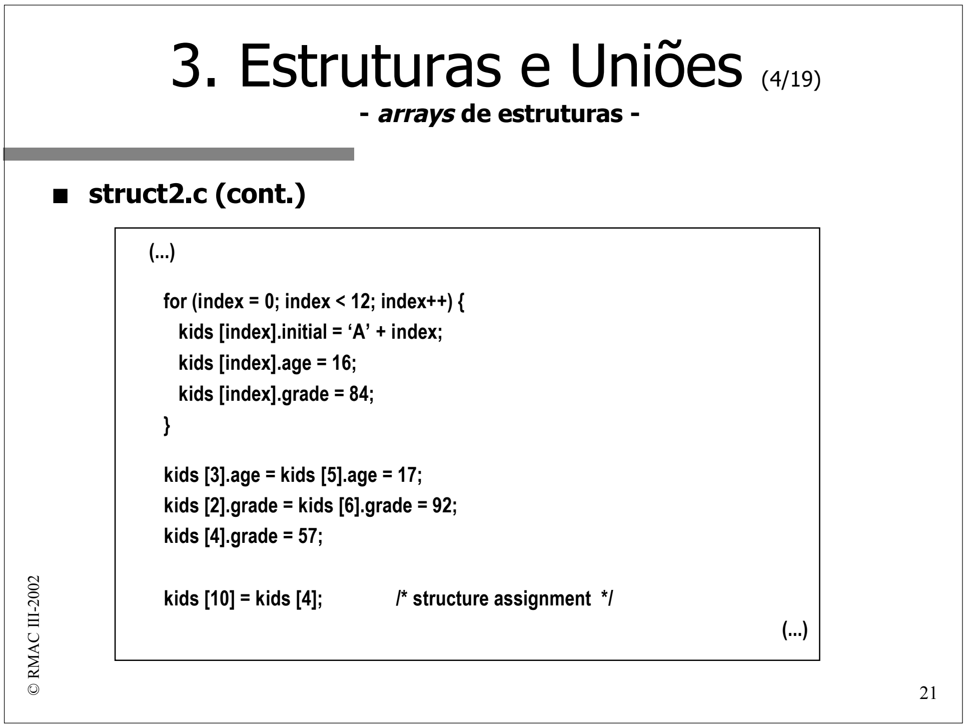
3. Estruturas e Uniões (3/19)

- *arrays* de estruturas -

- **struct2.c**

```
main ( )
{
struct {
   char initial;
   int age;
   int grade;
   } kids [12];

int index;
                                                        (...)
```



3. Estruturas e Uniões (4/19)

- *arrays* de estruturas -

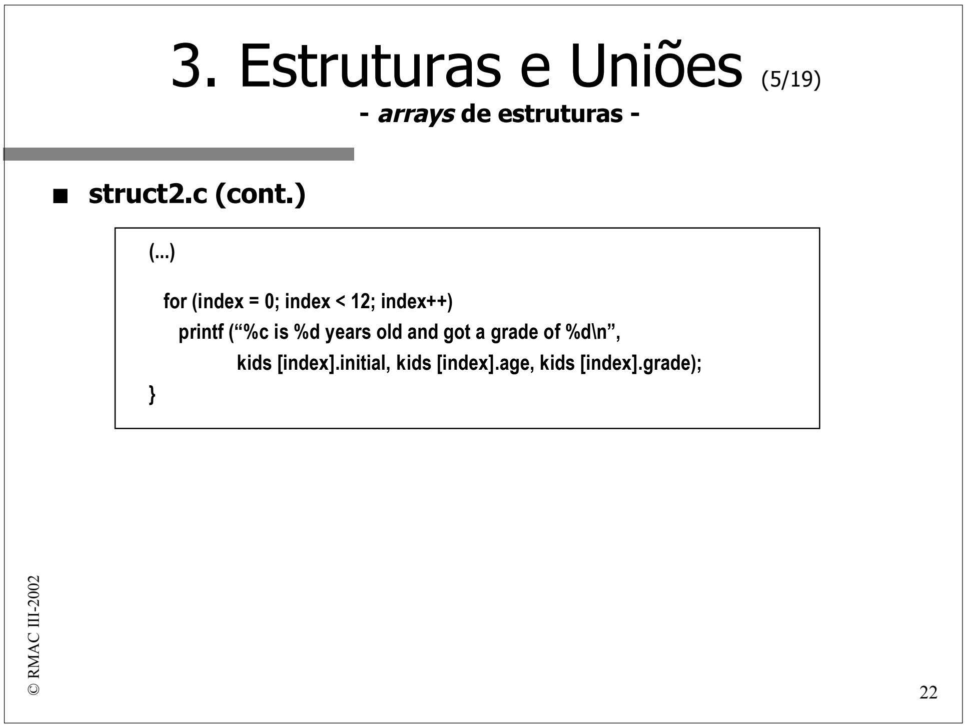
- **struct2.c (cont.)**

```
(...)

  for (index = 0; index < 12; index++) {
     kids [index].initial = 'A' + index;
     kids [index].age = 16;
     kids [index].grade = 84;
  }

  kids [3].age = kids [5].age = 17;
  kids [2].grade = kids [6].grade = 92;
  kids [4].grade = 57;

  kids [10] = kids [4];             /* structure assignment  */
                                                                    (...)
```



3. Estruturas e Uniões (5/19)

- *arrays* de estruturas -

- **struct2.c (cont.)**

```
(...)

  for (index = 0; index < 12; index++)
     printf ("%c is %d years old and got a grade of %d\n",
              kids [index].initial, kids [index].age, kids [index].grade);
}
```

3. Estruturas e Uniões (6/19)

- apontadores para estruturas -

- **struct3.c**

```
main ( )
{
struct {
  char initial;
  int age;
  int grade;
  } kids [12], *point, extra;

int index;
                                                    (...)
```

3. Estruturas e Uniões (7/19)

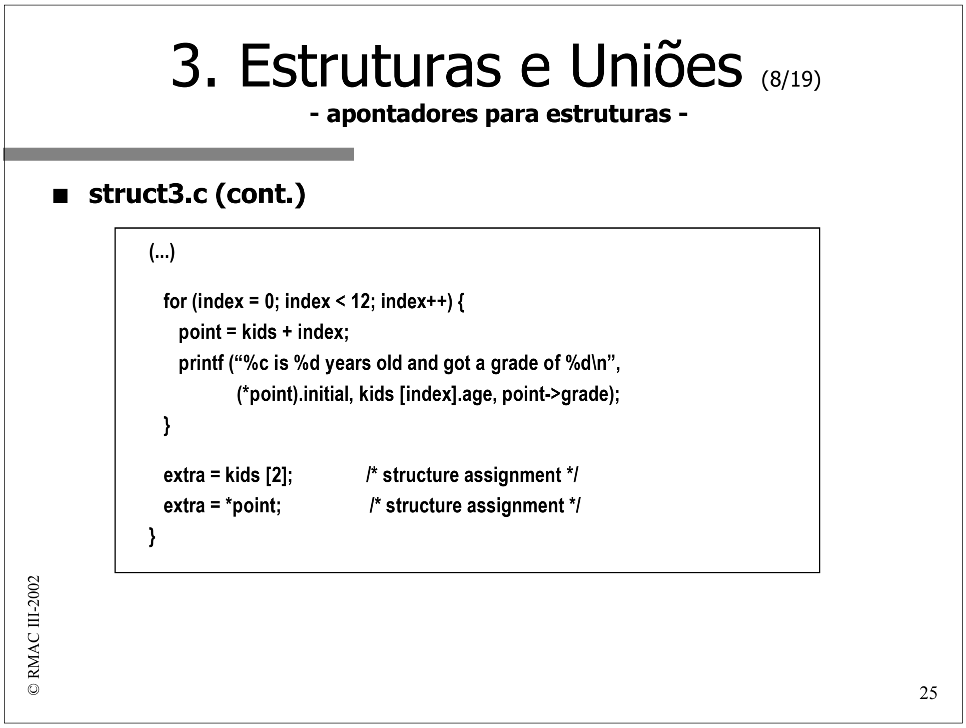
- apontadores para estruturas -

- **struct3.c (cont.)**

```
(...)

  for (index = 0; index < 12; index++) {
    point = kids + index;
    point->initial = ‘A’ + index;
    point->age = 16;
    point->grade = 84;
  }

  kids [3].age = kids [5].age = 17;
  kids [2].grade = kids [6].grade = 92;
  kids [4].grade = 57;
                                                    (...)
```



3. Estruturas e Uniões (8/19)

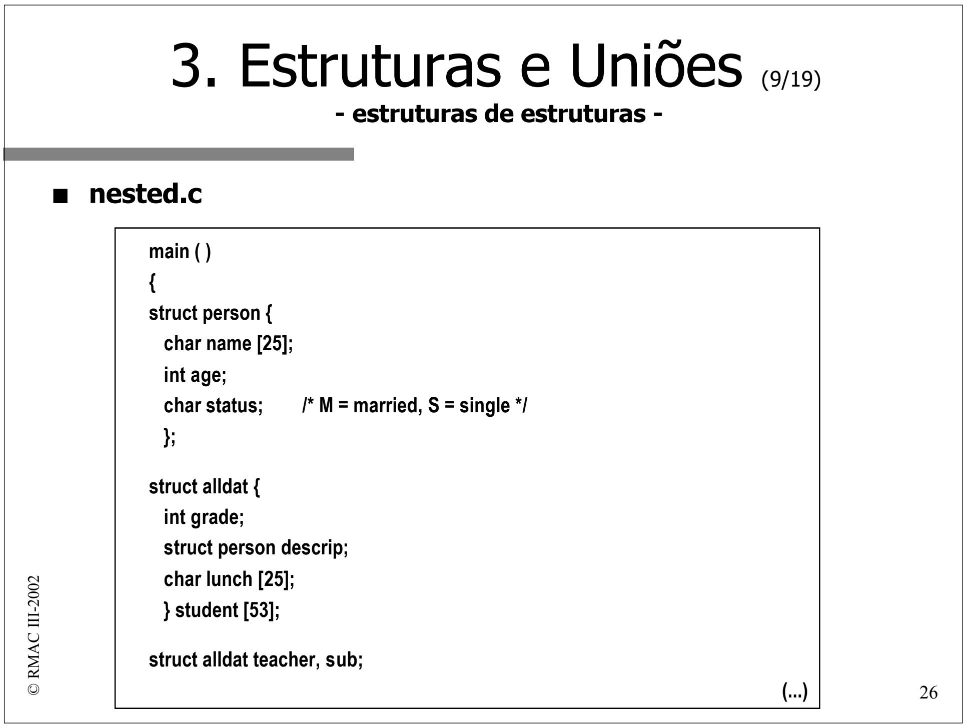
- apontadores para estruturas -

- **struct3.c (cont.)**

```
(...)

  for (index = 0; index < 12; index++) {
    point = kids + index;
    printf (“%c is %d years old and got a grade of %d\n”,
            (*point).initial, kids [index].age, point->grade);
  }

  extra = kids [2];              /* structure assignment */
  extra = *point;                /* structure assignment */
}
```



3. Estruturas e Uniões (9/19)

- estruturas de estruturas -

- **nested.c**

```
main ( )
{
struct person {
  char name [25];
  int age;
  char status;       /* M = married, S = single */
  };

struct alldat {
  int grade;
  struct person descrip;
  char lunch [25];
  } student [53];

struct alldat teacher, sub;
                                                    (...)
```

3. Estruturas e Uniões (10/19)

- estruturas de estruturas -

- **nested.c (cont.)**

```
(...)

  teacher.grade = 94;
  teacher.descrip.age = 34;
  teacher.descrip.status = ‘M’;
  strcpy (teacher.descrip.name, “Mary Smith”);
  strcpy (teacher.lunch, “Baloney sandwich”);

  sub.descrip.age = 87;
  sub.descrip.status = ‘M’;
  strcpy (sub.descrip.name, “Old Lady Brown”);
  sub.grade = 73;
  strcpy (sub.lunch, “Yogurt and toast”);
                                                  (...)
```

3. Estruturas e Uniões (11/19)

- estruturas de estruturas -

- **nested.c (cont.)**

```
(...)

  student [1].descrip.age = 15;
  student [1].descrip.status = ‘S’;
  strcpy (student [1].descrip.name, “Billy Boston”);
  strcpy (student [1].lunch, “Peanut Butter”);
  student [1].grade = 77;

  student [7].descrip.age = 14;
  student [12].grade = 87;
}
```

3. Estruturas e Uniões (12/19)

- lista ligada -

- **struct.def**

```
struct vars {              /* variable storage              */
   char varname [7];    /* variable name A-F & I-N     */
   char outtype;          /* output format for variable   */
   double value;          /* value of the variable           */
   };

struct lines {              /* dynamic structure for transcripts     */
   struct lines *dn;       /* next transcript line                       */
   struct lines *up;       /* last transcript line                        */
   char *lineloc;           /* point to dynamic location of line     */
   int linelngt;             /* length of line stored here               */
   char isvalue;            /* 1 = calculated value, 0 = none          */
   char marked;            /* 1 = line marked, 0 = not marked       */
   char strval [13];        /* string representation of variable      */
   };
```

3. Estruturas e Uniões (13/19)

- uniões -

- **union1.c**

```
main ( )
{
union {
   int value;      /* this is the first part of the union */
   struct {
      char first;   /* these two values are the second */
      char second;
      } half;
   } number;

long index;
                                                                    (...)
```

3. Estruturas e Uniões (14/19)

- uniões -

- **union1.c (cont.)**

```
(...)

  for (index = 12; index < 300000; index += 35231) {
    number.value = index;
    printf ("%8x %6x %6x\n", number.value, number.half.first,
              number.half.second);
  }
}
```

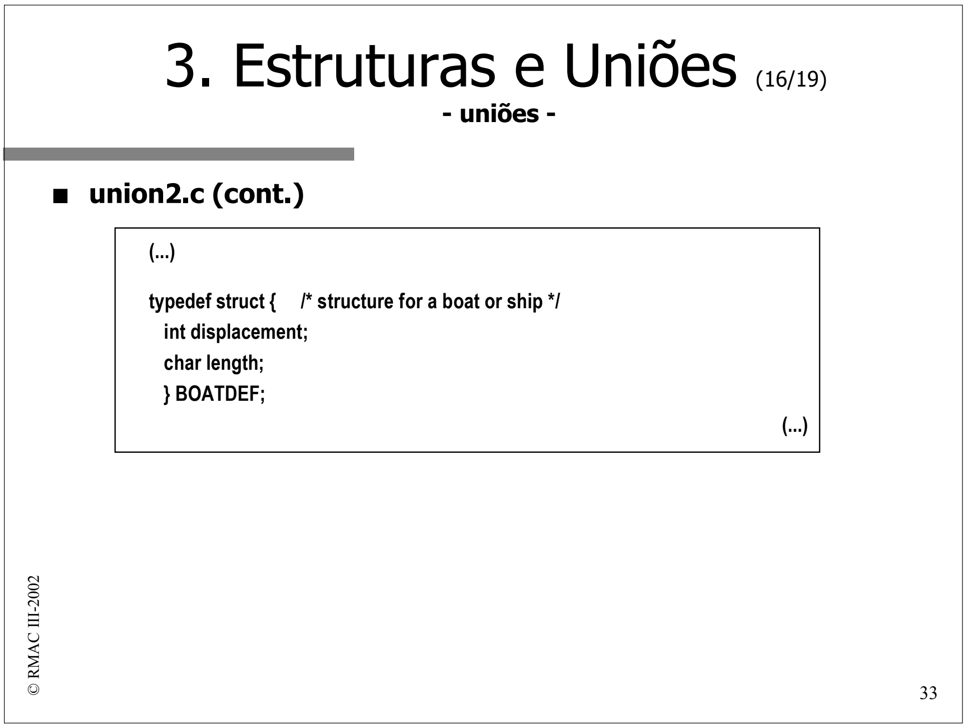
3. Estruturas e Uniões (15/19)

- uniões -

- **union2.c**

```
#define AUTO 1
#define BOAT 2
#define PLANE 3
#define SHIP 4

main ( )
{
struct automobile {     /* structure for an automobile */
  int tires;
  int fenders;
  int doors;
  };
                                                    (...)
```



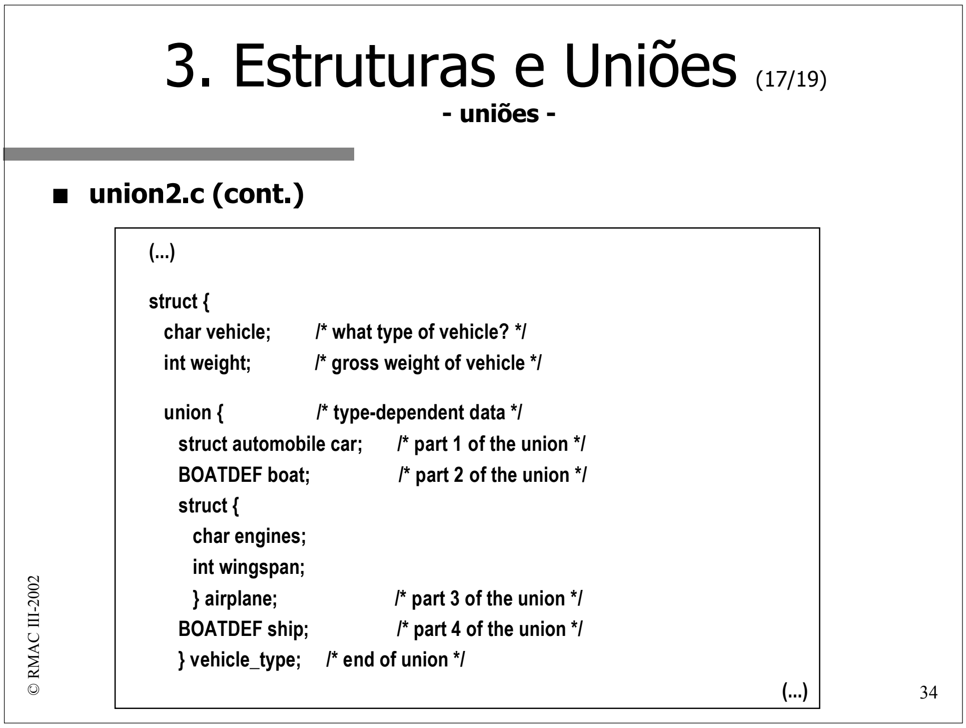
3. Estruturas e Uniões (16/19)

- uniões -

- **union2.c (cont.)**

```
(...)

typedef struct {     /* structure for a boat or ship */
   int displacement;
   char length;
   } BOATDEF;
                                                        (...)
```



3. Estruturas e Uniões (17/19)

- uniões -

- **union2.c (cont.)**

```
(...)

struct {
   char vehicle;         /* what type of vehicle? */
   int weight;           /* gross weight of vehicle */

   union {               /* type-dependent data */
      struct automobile car;      /* part 1 of the union */
      BOATDEF boat;               /* part 2 of the union */
      struct {
         char engines;
         int wingspan;
         } airplane;              /* part 3 of the union */
      BOATDEF ship;               /* part 4 of the union */
      } vehicle_type;    /* end of union */
                                                        (...)
```

3. Estruturas e Uniões (18/19)

- uniões -

- **union2.c (cont.)**

```
(...)

  int value;              /* value of vehicle in dollars */
  char owner [32];    /* owners name */
  } ford, sun_fish, piper_cub;  /* three variable structures */

     /* define a few of the fields as an illustration    */

  ford.vehicle = AUTO;
  ford.weight = 2742;              /* with a full gas tank */
  ford.vehicle_type.car.tires = 5;  /* including the spare */
  ford.vehicle_type.car.doors = 2;

  sun_fish.value = 3742;          /* trailer not included */
  sun_fish.vehicle_type.boat.length = 20;
                                                                                    (...)
```

3. Estruturas e Uniões (19/19)

- uniões -

- **union2.c (cont.)**

```
(...)

  piper_cub.vehicle = PLANE;
  piper_cub.vehicle_type.airplane.wingspan = 27;

  if (ford.vehicle == AUTO) /* which it is in this case */
     printf ("The ford has %d tires.\n", ford.vehicle_type.car.tires);

  if (piper_cub.vehicle == AUTO) /* which it is not in this case */
     printf ("The plane has %d tires.\n", piper_cub.vehicle_type.car.tires);
}
```

tente responder

exercícios

1. Defina uma estrutura que contenha um campo do tipo *string* para um nome e dois campos do tipo *integer*, um para o nº de pernas e outro para o nº de braços. Dê um nome a essa estrutura. Utilize este novo tipo de dados para declarar um *array* de 6 elementos. Preencha o *array* com valores de forma que, ao enviar para o monitor, a informação apareça, tal e qual como, se mostra de seguida:

```
Um humano tem 2 pernas e 2 braços.

Um cão tem 4 pernas e 0 braços.

Uma televisão tem 4 pernas e 0 braços.

Uma cadeira tem 4 pernas e 2 braços.

...
```

2. Re-escreva o exercício anterior, de forma a utilizar um apontador para os dados.

4. Alocação Dinâmica (1/11)

- apontadores para estruturas -

- **dynlist.c**

```
main ( )
{
struct animal {
   char name [25];
   char breed [25];
   int age;
   } *pet1, *pet2, *pet3;

   pet1 = (struct animal *) malloc (sizeof (struct animal) );
   strcpy (pet1->name, “General”);
   strcpy (pet1->breed, “Mixed Breed”);
   pet1->age = 1;

   pet2 = pet1;   /* pet2 now points to the above data structure */
                                                                    (...)
```

4. Alocação Dinâmica (2/11)

- apontadores para estruturas -

- **dynlist.c**

```
(...)

  pet1 = (struct animal *) malloc (sizeof (struct animal) );
  strcpy (pet1->name, “Frank”);
  strcpy (pet1->breed, “Labrador Retriever”);
  pet1->age = 3;

  pet3 = (struct animal *) malloc (sizeof (struct animal) );
  strcpy (pet3->name, “Krystal”);
  strcpy (pet3->breed, “German Shepherd”);
  pet3->age = 4;
                                                                    (...)
```

© RMAC III-2002

4. Alocação Dinâmica (3/11)

- apontadores para estruturas -

- **dynlist.c**

```
(...)

  /* now print out the data described above */
  printf (“%s is a %s, and is %d years old.\n”, pet1->name,
         pet1->breed, pet1->age);
  printf (“%s is a %s, and is %d years old.\n”, pet2->name,
         pet2->breed, pet2->age);
  printf (“%s is a %s, and is %d years old.\n”, pet3->name,
         pet3->breed, pet3->age);

  pet1 = pet3;   /* pet1 now points to the same structure that pet3 points to */

  free (pet3);   /* this frees up one structure            */
  free (pet2);   /* this frees up one more structure       */
  /* free (pet1);    this cannot be done */
}
```

© RMAC III-2002

4. Alocação Dinâmica (4/11)

- *arrays* de apontadores (para estruturas) -

- **bigdynl.c**

```
main ( )
{
struct animal {
  char name [25];
  char breed [25];
  int age;
  } *pet [12], *point;   /* this defines 13 pointers, no variables */

int index;
                                                        (...)
```

4. Alocação Dinâmica (5/11)

- *arrays* de apontadores (para estruturas) -

- **bigdynl.c**

```
(...)

  /* first, fill the dynamic structures with nonsense */

  for (index = 0; index < 12; index++) {
    pet [index] = (struct animal *) malloc (sizeof (struct animal) );
    strcpy (pet [index]->name, “General”);
    strcpy (pet [index]->breed, “Mixed Breed”);
    pet [index]->age = 4;
  }

  pet [4]->age = 12;      /* these lines are simply to      */
  pet [5]->age = 15;      /*     put some nonsense data into */
  pet [6]->age = 10;      /*         a few of the fields.  */
                                                        (...)
```

4. Alocação Dinâmica (6/11)

- *arrays* de apontadores (para estruturas) -

- **bigdynl.c**

```
(...)

  /* now print out the data described above */

  for (index = 0; index < 12; index++) {
     point = pet [index];
     printf (“%s is a %s, and is %d years old.\n”, point->name,
             point->breed, point->age);
  }

  /* good programming practice dictates that we free up the */
  /* dynamically allocated space before we quit.            */

  for (index = 0; index < 12; index++) free (pet [index]);
}
```

4. Alocação Dinâmica (7/11)

- listas ligadas -

- **dynlink.c**

```
#include “stdio.h”   /* this is needed only to define the NULL    */
#define RECORDS 6

main ( )
{
struct animal {
   char name [25];          /* the animals name   */
   char breed [25];         /* the type of animal  */
   int age;                 /* the animals age     */
   struct animal *next;     /* a pointer to another record of this type */
   } *point, *start, *prior;   /* this defines 3 pointers, no variables  */

int index;
                                                              (...)
```

4. Alocação Dinâmica (8/11)

- listas ligadas -

- **dynlink.c**

```
(...)

  /* the first record is always a special case */

  start = (struct animal *) malloc (sizeof (struct animal) );
  strcpy (start->name, “General”);
  strcpy (start->breed, “Mixed Breed”);
  start->age = 4;
  start->next = NULL;
  prior = start;
                                                              (...)
```

4. Alocação Dinâmica (9/11)

- listas ligadas -

- **dynlink.c**

```
(...)

  /* a loop can be used to fill in the rest once it is started */

  for (index = 0; index < RECORDS; index++) {
    point = (struct animal *) malloc (sizeof (struct animal) );
    strcpy (point->name, “Frank”);
    strcpy (point->breed, “Laborador Retriever”);
    point->age = 3;
    prior->next = point;   /* point last “next” to this record */
    point->next = NULL;    /* point this “next” to NULL        */
    prior = point;         /* this is now the prior record     */
  }
                                                              (...)
```

4. Alocação Dinâmica (10/11)

- listas ligadas -

- **dynlink.c**

```
(...)

  /* now print out the data described above */

  point = start;

  do {
    prior = point->next;
    printf ("%s is a %s, and is %d years old.\n", point->name,
            point->breed, point->age);
    point = point->next;
  } while (prior != NULL);


                                                              (...)
```

4. Alocação Dinâmica (11/11)

- listas ligadas -

- **dynlink.c**

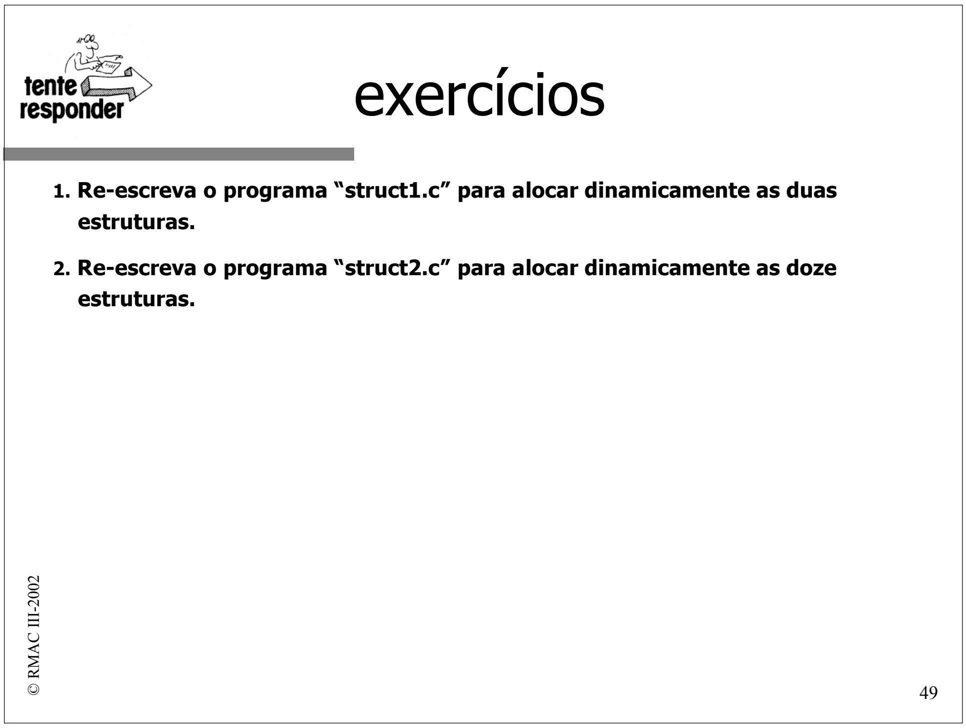
```
(...)

  /* good programming practice dictates that we free up the */
  /* dynamically allocated space before we quit              */

  point = start;              /* first block of group     */

  do {
    prior = point->next;      /* next block of data    */
    free (point);             /* free present block    */
    point = prior;            /* point to next         */
  } while (prior != NULL);    /* quit when next is NULL    */

}
```


tente
responder

exercícios

1. Re-escreva o programa "struct1.c" para alocar dinamicamente as duas estruturas.
2. Re-escreva o programa "struct2.c" para alocar dinamicamente as doze estruturas.