PUC-Rio – Programação 2 – INF1007
Prova 2 – Soluções – Turma 33C

- Questão 1a: (Leandro)

```
int buscaFaixaIdade (Bio **v, int n, int idade, int *lim1, int *lim2) {
  int ini, fim, meio, x;
  ini = 0;
  fim = n-1;

  if (n == 0) return -1;

  /* busca binária para achar alguem com a idade especificada */
  do {
    meio = (ini+fim)/2;
    if (v[meio] -> idade == idade) { /* achou */

      /* procura o primeiro desta faixa, a partir do que achou */
      for (x = meio; x >= 0; x--)
        if (v[x]->idade != idade) break;

      /* parou no anterior do primeiro */
      *lim1 = x+1;

      /* procura o último desta faixa, a partir do que achou */
      for (x = meio; x < n; x++)
        if (v[x]->idade != idade) break;

      /* parou no seguinte ao último */
      *lim2 = x-1;

      return 1;
    }
    else
      if (v[k] -> idade > idade)
        fim = meio-1;
      else
        ini = meio + 1;
  } while (ini <= fim);

  return -1; /* não há ninguém com esta idade */
}
```

- Questão 1b (usando `qsort`): (Maria Beatriz)

```
#include <stdlib.h>

/* Função de comparação */
int comparaQ(const void *p1, const void *p2) {
  Bio *b1, *b2;
  b1 = *((Bio **)p1);
  b2 = *((Bio **)p2);

  if (b1->peso < b2->peso) return -1;
  if (b1->peso > b2->peso) return 1;
  if (b1->altura < b2->altura) return -1;
  if (b1->altura > b2->altura) return 1;
  return 0;
}
```

```
/* Função para ordenação */
void ordenaFaixaIdade(Bio **v, int n, int idade) {
  int ind1, ind2, situ, tam;
  situ = buscaFaixaIdade(v, n, idade, &ind1, &ind2);
  if (situ == 1) {

    /* tamanho da faixa */
    tam = ind2 - ind1 + 1;

    /* ordena a partir do início da faixa */
    qsort(&v[ind1], tam, sizeof(Bio *), comparaQ);
  }
}
```

- Questão 1b (sem usar `qsort`):

  Várias das soluções da turma ordenam (ou tentam ordenar) apenas a faixa de idade especificada. Porém todas essas soluções tem um problema de eficiência, porque a busca pela faixa é feita **dentro** da função de ordenação. Assim, a busca é realizada novamente, sem necessidade, a cada chamada da função de ordenação. Apenas **uma** busca é necessária, para identificar que parte do vetor deve ser ordenada. Por isso, essa busca deve ser realizada **antes** de fazer a ordenação. Uma solução possível é:

```
int compara(Bio *b1, Bio *b2) {
  if (b1->peso < b2->peso) return -1;
  if (b1->peso > b2->peso) return 1;
  if (b1->altura < b2->altura) return -1;
  if (b1->altura > b2->altura) return 1;
  return 0;
}

void ordena(Bio **v, int n) {
  int a = 0;
  int b = n-1;
  Bio *temp, *pivo = v[0];

  if (n <= 1) return;
  do {
    while ((a < n) && compara(v[a], pivo) <= 0) a++;
    while (compara(v[b], pivo) > 0) b--;
    if (a < b) {
      temp = v[a];
      v[a] = v[b];
      v[b] = temp;
      a++; b--;
    }
  } while (a <= b);
  v[0] = v[b];
  v[b] = pivo;
  ordena(v,b);
  ordena(&v[a], n-a);
}
void ordenaFaixaIdade(Bio **v, int n, int idade) {
  int ind1, ind2;
  if (buscaFaixaIdade(v, n, idade, &ind1, &ind2) == 1)
    ordena(&v[ind1], ind2-ind1+1);
}
```

- Questão 2: (Isabel)

  Observação: esta solução foi adequada ao uso do `typedef` fornecido no enunciado.

```
/* Interface: circulo.h */
typedef struct circulo *Circulo;
Circulo cria_circ(float x, float y, float raio);
void libera_circ(Circulo c);
float obtemy(Circulo c);
float obtemx(Circulo c);
float obtemraio(Circulo c);


/* Implementação: circulo.c */
#include <stdlib.h>
#include "circulo.h"

struct circulo {
  float x, y, raio;
};

Circulo cria_circ(float x, float y, float r) {
  Circulo c = (Circulo) malloc(sizeof (struct circulo));
  c->x = x;
  c->x = y;
  c->raio = r;
  return c;
}

void libera_circ(Circulo c) {
  free(c);
}

float obtemx(Circulo c) {
  return c->x;
}

float obtemy(Circulo c) {
  return c->y;
}

float obtemraioCirculo c) {
  return c->raio;
}

/* Usuário do TAD */
#include <stdio.h>
#include "circulo.h"

int circuloAcima(Circulo c) {
  float y, raio;
  y = obtemy(c);
  raio = obtemraio(c);

  if (y- raio < 0)
    return 0;
  return 1;
}
```