CTT003

THI CUỐI KỲ (80')

November 21, 2018

Quy định thi cuối kỳ

- KHÔNG SỬ DỤNG INTERNET (trừ trang web Moodel môn học - để download và nộp bài)
- Đây là bài thi cá nhân, tất cả các hình thức gian lận đều được chuyển về Khoa với hình thức xử lý "RỚT MÔN". Các hình thức gian lận gồm:
  - Trao đổi, thảo luận trong lúc làm bài thi
  - Sao chép bài người khác không phải của mình (của bạn, trên Internet, ...)
  - Cho bạn sao chép bài của mình
  - Sử dụng INTERNET không theo quy định

Quy định nộp bài

- **Lặp vô tận = 0 điểm, lỗi không chạy được chương trình = 0 điểm (lỗi ở 1 hàm có thể làm cả chương trình không chạy được)**
- Sinh viên tuỳ chỉnh hàm `mainStudent` để debug.
  Sinh viên tuỳ chỉnh giá trị `volatile int flag = 1` để chạy bộ FINAL TEST (giá trị khởi tạo là `flag = 0`).
- Nộp tại Moodle môn học, phần Final
- Nộp file MSSV.zip/functions.c – chỉ nộp file FUNCTIONS.C

Bài 1: Đệ quy (4)

Dãy số `Hailstone` bắt đầu từ một số tự nhiên `n` cho trước, các số tiếp theo sẽ được tạo theo quy tắc

i) Nếu số hiện tại là chẵn thì số tiếp theo sẽ bằng số đó chia cho 2

ii) Nếu số hiện tại là lẻ thì số tiếp theo sẽ được nhân lên 3 lần rồi cộng 1

iii) Khi gặp số 1 thì dãy số kết thúc

Viết hàm đệ quy tính tổng các số trong dãy `Hailstone` khi biết `n`? Lưu ý không viết đệ quy thì trừ 2/3 số điểm.

- Input: `n = 3`
- Khi đó, dãy Hailstone tương ứng là 3, 10, 5, 16, 8, 4, 2, 1.
- Ouput: Tổng = `49`
- Prototype: `int hailstone(int n)`
- Statement: `hailstone(3) = 49`

Bài 2: Mảng (Max_score = 7)

2 câu tiếp theo là phần kiểm tra về mảng. Chọn làm 1 trong 2 câu (hoặc làm cả 2 đều được). Hãy đọc đề cả 2 trước khi chọn làm.

- Làm câu 2.1 (câu dễ - 4 điểm)
  - Ưu:
    - dễ
    - nhanh
  - Nhược: điểm final tối đa = 8 điểm
- Làm câu 2.2 (câu khó - 7 điểm)
  - Nhược: khó nên tốn thời gian
  - Ưu:
    - điểm final tối đa = 11 điểm
    - có gợi ý cách giải (và bạn có thể nghĩ ra cách làm khác hay hơn gợi ý)

Bài 2. 1: (4)

Viết hàm:

- Hàm tạo mảng gồm `n` phần tử là các số liên tiếp trong dãy `Fibonacci`, với `n>=1` ($F_0 = 0; F_1 = 1; F_i = F_{i-1} + F_{i-2}$)
  - Input: `a[]` và `n = 6`
  - Ouput: `a = {0, 1, 1, 2, 3, 5}`
  - Prototype: `void setFibArray(int a[], int n)`
  - Statement: `setFibArray(a, 6) => a = {0, 1, 1, 2, 3, 5}`
- Hàm đếm số cặp phần tử liền kề mà cả 2 cùng lẻ
  - Input: `a[]` kết quả của câu trên
  - Ouput: `2` vì có 2 cặp (1,1) (3,5)
  - Prototype: `int countPairsAdjacentOddNumber(int a[], int n)`
  - Statement: `countPairsAdjacentOddNumber(a, 6) = 1`

Bài 2. 2: (7)

Một nhóm người muốn đứng xếp hàng tại công viên theo thứ tự chiều cao tăng dần. Tuy nhiên có rất nhiều cây cao mọc rải rác trong công viên. Khi xếp hàng, do bị cây che khuất nên hầu hết mọi người đều không thể nhìn thấy chiều cao của nhau và tự sắp xếp được.

Nhiệm vụ của bạn là sắp xếp hàng người này theo thứ tự chiều cao tăng dần mà không làm thay đổi vị trí hàng cây.

Lưu ý: chiều cao của người tại công viên này chỉ mang tính chất minh hoạ

- Input: mảng ban đầu `a = {-1, 180, 150, 170, -1, -1, 160, 180}`, chiều dài mảng `len = 8`, số lượng cây (-1) `tree = 3`
- Ouput: `b = {-1, 150, 160, 170, -1, -1, 180, 180}`
- Prototype: `void sortByHeight(int a[], int len, int tree, int b[])`
- Statement: `sortByHeight(a, 8, 3, b) => b = {-1, 150, 160, 170, -1, -1, 180, 180}`

Gợi ý giải

(1) Dùng mảng mới lưu chiều cao người

(2) Sắp xếp mảng mới này: dùng hàm sort có sẵn tại thư viện std::iostream + algorithm: `sort(startaddress, endaddress)`
=> VD: dùng hàm sort cho mảng `people`: `sort(people, people + chiều dài của people)`

(3) Trong mảng kết quả, tại các vị trí không phải là cây, đưa lần lượt giá trị của mảng mới đã được sắp xếp vào