Wstęp do programowania w języku C

Grupa TDr

Lista 7.

1. *(15 punktów w trakcie pracowni, później 10 punktów)*
Zadanie polega na napisaniu siedmiu funkcji działających na poniższej reprezentacji list, gdzie `head` oznacza pierwszy element listy, a `tail` — jej dalszy ciąg. Pusta lista jest reprezentowana przez wskaźnik pusty.

```
typedef struct IntListNodeAux IntListNode;
typedef IntListNode* IntList;

struct IntListNodeAux {
    int head;
    IntList tail;
};
```

Do zachowania prawa do zdobycia 15 punktów wystarczy w czasie pracowni zaprezentować poprawnie napisane 4 z nich. Dłuższy przykład znajduje się na końcu treści zadania.

1. (2 punkty) Funkcja `IntList array_to_list_alloc(int* memory, int size);` powinna zaalokować i zwrócić listę o długości `size` (czyli `size` węzłów listy), której elementami będzie `size` pierwszych elementów tablicy `memory`.

2. (1,5 punktu) Funkcja o deklaracji `void print_list(IntList xs);` powinna wydrukować w jednej linii elementy listy `xs` oddzielone przecinkiem ze spacją (przecinek po ostatnim elemencie jest dozwolony), a całość otoczoną nawiasami kwadratowymi (czyli w pythonowym formacie list).

3. (1,5 punktu) Funkcja o deklaracji `int length(IntList xs);` powinna obliczyć i zwrócić długość listy `xs`.

4. (1,5 punktu) Funkcja o deklaracji `int sum(IntList xs);` powinna obliczyć i zwrócić sumę elementów listy `xs`.

5. (2,5 punktu) Funkcja `int fold(int op(int, int), int acc, IntList xs);` powinna obliczyć i zwrócić wynik łącznego i przemiennego działania `op` wykonywanego na elementach listy `xs` i liczbie `acc`. Zatem jeśli lista `xs` jest pusta, to powinna zwrócić `acc`. Łączność działania `op` oznacza, że `op`(`op`(a, b), c) = `op`(a, `op`(b, c)). Przykładowo dla listy `[1, 2, 3, ]` powinna zwrócić `op`(1, `op`(2, `op`(3, `acc`))).

6. (2 punkty) Funkcja `IntList list_plus(const IntList xs, const IntList ys);` powinna zaalokować odpowiednie węzły i zwrócić listę będącą sklejeniem list `xs` i `ys`, czyli listę, której elementami są najpierw elementy listy `xs`, a potem `ys`. Nie może przy tym modyfikować list `xs` i `ys`. Możemy założyć, że listy `xs` i `ys` nie będą w przyszłości modyfikowane i dzięki temu możemy współdzielić część wyniku z argumentem.

7. (4 punkty) Funkcja o deklaracji `IntList reverse_inplace(IntList xs);` powinna wykorzystać węzły listy `xs` do stworzenia listy o odwróconej kolejności elementów i ją zwrócić. Funkcja powinna działać w stałej pamięci, czyli może korzystać z dodatkowej, ale góry określonej ilości pamięci (nawet jeśli lista `xs` ma tysiąc czy milion elementów). Nie powinna wykonywać żadnej alokacji na stercie ani wywoływać się rekurencyjne (chyba że będą to wywołania ogonowe).

Przykład. Uruchomienie funkcji `main` z poniższego kodu:

```
int       addition(int a, int b) { return a + b; }
int multiplication(int a, int b) { return a * b; }

int main() {
    int arr[] = {3, 1, 4, 1, 5, 9, 2};
    IntList xs = array_to_list_alloc(arr + 1, 5);
    print_list(xs);
    printf("%d %d ", length(xs), sum(xs));
    printf("%d %d\n", fold(addition, 0, xs), fold(multiplication, 1, xs));
    print_list(list_plus(xs, xs));
    xs = reverse_inplace(xs);
    print_list(xs);
}
```

powinno wypisać np.:

```
[1, 4, 1, 5, 9, ]
5 20 20 180
[1, 4, 1, 5, 9, 1, 4, 1, 5, 9, ]
[9, 5, 1, 4, 1, ]
```

2. *(15 punktów)*
w serwisie SKOS