2 Permutacje, stos, kolejki

Zadanie 2.1. Losowanie, permutacje, sortowanie

Szablon programu należy uzupełnić o definicje 3 funkcji:

- Funkcja `int rand_from_interval(int a, int b)` – korzystając z bibliotecznej funkcji `rand()` oraz operacji dzielenia modulo – oblicza i zwraca liczbę należącą do domkniętego przedziału $[a, b]$. *Szczegóły w komentarzu szablonu programu.*
  Założenie: liczba elementów zbioru, z którego odbywa się losowanie, nie jest większa od `RAND_MAX`+1.

- Funkcja `void rand_permutation(int n, int tab[])` losowo wybiera jedną z permutacji n elementów zbioru liczb naturalnych. Elementy tego zbioru – liczby naturalne z przedziału $[0, n-1]$ – są wpisywane do tablicy `tab` w porządku rosnącym. Do losowania liczby z przedziału należy wykorzystać zdefiniowaną wcześniej funkcję `rand_from_interval()`.
  Zapisany w pseudokodzie algorytm wpisywania liczb do tablicy oraz wyboru permutacji:

  Require: $n \geqslant 0$
  for $i \leftarrow 0$ to $n-1$ **do**
  $a[i] \leftarrow i$
  end for
  for $i \leftarrow 0$ to $n-2$ **do**
  $k \leftarrow \text{random}(i, n-1)$ ▷ losowanie z przedzialu $[i, n-1]$
  $\text{swap}(a[i], a[k])$ ▷ zamiana elementow i i k tablicy a
  end for

- Funkcja `int bubble_sort(int n, int tab[])` metodą bąbelkową sortuje n elementów tablicy `tab` wg porządku od wartości najmniejszej do największej.
  Uwaga: Są dwa możliwe kierunki przeglądania elementów sortowanej tablicy - w kierunku rosnących albo malejących **indeksów** tej tablicy. Proszę zastosować ten pierwszy (od małych do dużych).
  Uwaga ta nie dotyczy kierunku uporządkowania elementów, lecz kierunku przeglądania tablicy, czyli najpierw porównujemy `t[i]` z `t[i+1]`, później `t[i+1]` z `t[i+2]` itd., a nie `t[i]` z `t[i-1]`, później `t[i-1]` z `t[i-2]` itd.
  Funkcja zwraca najmniejszą liczbę przeglądań tablicy (liczbę iteracji zewnętrznej pętli algorytmu sortowania), po której elementy tablicy są właściwie uporządkowane.

W segmencie głównym szablonu programu są zapisane trzy testy ww. funkcji.
Każdy test wymaga osobnego wykonania (egzekucji) programu.
Pierwszą daną wczytywaną przez program jest liczba – numer testu, który ma być zrealizowany.
Drugą wczytywaną daną jest zarodek (`seed`) generatora liczb pseudolosowych.
Wywoływanie funkcji `srand(seed)` ma na celu uzyskanie powtarzalności otrzymywanych wyników testów.

Test 2.1.1: Wypisanie losowo wygenerowanych 3 liczb z zadanego przedziału $[a, b]$

Test wczytuje granice przedziału i wypisuje trzy wygenerowane liczby w kolejności zgodnej z kolejnością ich generowania.

- **Wejście**
  1 `seed a b`
- **Wyjście**
  Trzy wylosowane liczby całkowite.
- **Przykład:**
  Wejście: 1 100 3 30
  Wyjście: 11 4 10

Test 2.1.2: Losowy wybór permutacji

Test wczytuje liczność zbioru (liczbę elementów zbioru), wywołuje funkcję `rand_permutation()` i wypisuje wylosowaną permutację.

- **Wejście**
  2 `seed n`
- **Wyjście**
  Wylosowana permutacja n liczb całkowitych.
- **Przykład:**
  Wejście: 2 20 10
  Wyjście: 1 0 3 4 6 2 8 9 5 7

Test 2.1.3: Sortowanie elementów tablicy metodą bąbelkową

Test wczytuje liczbę n elementów sortowanej tablicy, wywołuje funkcję generującą permutację `rand_permutation()` oraz funkcję sortującą permutację `bubble_sort()`.

- **Wejście**
  3 `seed n`
- **Wyjście**
  Numer iteracji pętli zewnętrznej (liczony od 1), po której tablica była już uporządkowana, np.:
  dla 0 1 2 3 7 4 5 6 wynik = 1,
  dla 1 2 3 7 4 5 6 0 wynik = 7,
  dla 0 1 2 3 4 5 6 7 wynik = 0.
- **Przykład:**
  Wejście: 3 20 10
  Wyjście: 3

Zadanie 2.2. Stos, kolejka w tablicy z przesunięciami, kolejka z buforem cyklicznym

W programie są zdefiniowane tablice `stack, queue, cbuff`. Ich rozmiary są takie same i równe 10.

2.2.1: Stos

Stos jest realizowany za pomocą tablicy `stack` i zmiennej `top` zdefinowanymi poza blokami funkcji. Szablon programu należy uzupełnić o definicję funkcji obsługujących stos `stack_push(), stack_pop(), stack_state()`.

- Funkcja `stack_push(double x)` kładzie na stosie wartość parametru i zwraca zero, a w przypadku przepełnienia stosu - nie zmienia zawartości stosu i zwraca stałą `INFINITY` (zdefiniowaną w `math.h`).
- Funkcja `stack_pop(void)` zdejmuje ze stosu jeden element i zwraca jego wartość. W przypadku stosu pustego (pustego przed próbą zdjęcia elementu) zwraca stałą `NAN` (też zdefiniowaną w `math.h`).
- Funkcja `stack_state(void)` zwraca liczbę elementów leżących na stosie.

Test 2.2.1 (stosu)
Test pozwala zapisać na stosie dodatnie liczby typu `double`. Wczytuje numer testu oraz ciąg liczb rzeczywistych reprezentujących operacje na stosie:

- Wpisanie dodatniej liczby `x` powoduje wywołanie funkcji `stack_push(x)` i w przypadku przepełnienia stosu wypisuje wartość stałej `INFINITY`.
- Wpisanie ujemnej liczby powoduje wywołanie funkcji `stack_pop()` i wypisanie zwracanej przez nią wartości.
- Wpisanie zera powoduje wywołanie funkcji `stack_state()`, wypisanie zwracanej przez nią wartości i zakończenie testu.
- **Przykład:**
  Wejście:
  1
  2. 4. 5. 7. 1. -2. -1. 9. -1. 5. 0.
  Wyjście:
  1.00 7.00 9.00
  4

2.2.2: Kolejka w tablicy z przesunięciami

Obsługa kolejki (typu FIFO) jest realizowana z zastosowaniem tablicy `queue` i zmiennej `in` zdefiniowanymi poza blokami funkcji. Wartością zmiennej `in` jest liczba klientów oczekujących w kolejce. Kolejny pojawiający się klient otrzymuje kolejny numer począwszy od 1. Klient, który zastaje pełną kolejkę, rezygnuje

z oczekiwania, ale zachowuje swój numer (kolejny klient otrzyma następny numer). Numery klientów czekających w kolejce są pamiętane w kolejnych elementach tablicy `queue` w taki sposób, że numer klienta najdłużej czekającego jest pamiętany w `queue[0]`.

Szablon programu należy uzupełnić o definicję funkcji obsługujących kolejkę `queue_push(), queue_pop(), queue_state(), queue_print()`.

- Funkcja `queue_push(int how_many)` powiększa kolejkę o `how_many` klientów. Numer bieżącego klienta jest pamiętany w zmiennej globalnej `curr_nr`. Zwraca 0.0.
  W przypadku, gdy liczba wchodzących do kolejki jest większa niż liczba wolnych miejsc w kolejce, miejsca w kolejce są zajmowane do zapełnienia miejsc, a pozostali "niedoszli klienci" rezygnują (zachowując swoje numery). W takiej sytuacji funkcja zwraca stałą `INFINITY`.

- Funkcja `queue_pop(int how_many)` symuluje wyjście z kolejki (obsługę) `how_many` najdłużej czekających klientów. Funkcja zwraca długość pozostałej kolejki.
  W przypadku gdy `how_many` jest większa od długości kolejki, kolejka jest opróżniana, a funkcja zwraca -1.

- Funkcja `queue_state()` zwraca liczbę czekających w kolejce.

- Funkcja `queue_print()` wypisuje numery czekających klientów (w kolejności wejścia do kolejki).

Test 2.2.2 (kolejki z przesunięciami)

- **Wejście**
  Test wczytuje numer testu oraz ciąg liczb całkowitych reprezentujących operacje na kolejce:
  - Liczba dodatnia jest liczbą klientów dochodzących do kolejki.
  - Liczba ujemna jest liczbą obsłużonych klientów opuszczających kolejkę.
  - Zero powoduje wywołanie funkcji `queue_state()` i wypisanie zwracanej przez nią wartości, wywołanie funkcji `queue_print()` oraz zakończenie testu.

- **Wyjście**
  Wartości stałych:
  - `INFINITY`, gdy wystąpiła próba przepełnienia kolejki,
  - -1 w przypadku próby wyjścia z kolejki klientów, których nie było w kolejce.

  Po wpisaniu na wejściu liczby 0 są wypisywane: liczba czekających klientów oraz ich numery wg kolejności w kolejce.

- **Przykład:**
  Wejście:
  2
  1 3 5 -2 7 -3 -9 3 -1 0
  Wyjście:
  inf -1
  2
  18 19

2.2.3: Kolejka w buforze cyklicznym

Obsługa kolejki (typu FIFO) jest realizowana z zastosowaniem tablicy `cbuff` służącej jako bufor cykliczny i zmiennych `out` i `len` zdefiniowanymi poza blokami funkcji. Wartością zmiennej `len` jest liczba klientów oczekujących w kolejce, a zmiennej `out` – indeks tablicy `cbuff`, w której jest pamiętany numer klienta najdłużej czekającego (o ile długość kolejki `len` > 0).

Szablon programu należy uzupełnić o definicję funkcji obsługujących kolejkę `cbuff_push(), cbuff_pop(), cbuff_state(), cbuff_printf()`.

- Funkcja `cbuff_push(int cli_nr)` powiększa kolejkę o jednego klienta o numerze `cli_nr` i zwraca 0.0. W przypadku braku miejsca w kolejce zwraca stałą `INFINITY`.
- Funkcja `cbuff_pop()` symuluje obsługę i wyjście z kolejki najdłużej czekającego klienta. Funkcja zwraca numer klienta wychodzącego z kolejki, a w przypadku, gdy kolejka była pusta, zwraca -1.
- Funkcja `cbuff_state()` zwraca liczbę czekających klientów.
- Funkcja `cbuff_state()` wypisuje numery czekających klientów (wg kolejności w kolejce).

Test 2.2.3 (kolejki w buforze cyklicznym)
Test jest symulacją kolejki z wykorzystaniem bufora cyklicznego. Wczytywane liczby są kodami operacji na kolejce:

1. Liczba dodatnia oznacza przyjście nowego klienta. Pierwszy klient otrzymuje numer 1. Kolejny klient otrzymuje kolejny numer. Klient, który zastaje pełną kolejkę, rezygnuje z oczekiwania, ale zachowuje swój numer (kolejny klient otrzyma następny numer).
   Klient (jego numer) jest umieszczany w kolejce przez wywołanie funkcji `cbuff_push(nr_klienta)`. Funkcja ta zapisuje przesłany numer w elemencie tablicy (bufora) o indeksie `out + len` (z uwzględnieniem „cykliczności” bufora).
2. Liczba ujemna wywołuje funkcję `cbuff_pop()`, która symuluje obsługę i opuszczenie kolejki przez jednego klienta.

3. Zero powoduje wywołanie funkcji `cbuff_state()` i wypisanie zwracanej przez nią wartości, wywołanie funkcji `cbuff_print()` oraz zakończenie testu.

- **Wejście**
  Test wczytuje numer testu oraz ciąg liczb całkowitych reprezentujących operacje na kolejce.

- **Wyjście**
  Pierwsza linia zawiera:

  – numery klientów wychodzących z kolejki,

  – stałą `INFINITY` w przypadku próby przepełnienia,

  – -1 w przypadku próby symulacji wyjścia klienta nieobecnego w kolejce

  w kolejności pojawiania się ww. zdarzeń.
  W drugiej linii wypisywana jest liczba klientów pozostających w kolejce po zakończeniu symulacji, a w trzeciej linii - ich numery.

- **Przykład:**
  Wejście:
  3
  1 1 -1 -1 -1 1 1 1 1 1 1 1 1 1 1 1 -1 1 0
  Wyjście:
  1 2 -1 inf inf 3
  10
  4 5 6 7 8 9 10 11 12 15

Zadanie 2.3. Symulacja gry Wojna

Zadanie polega na napisaniu programu symulującego grę w karty.

Ogólne zasady gry przyjmujemy za Wikipedią `https://pl.wikipedia.org/wiki/Wojna_(gra_karciana)`. Występuje tam pojęcie "wojna" jako "spotkanie się" kart o takim samym poziomie starszeństwa. Dodajmy termin "konflikt" na określenie bardziej elementarnego zdarzenia - spotkania się dwóch kart, przy którym konieczne jest rozstrzygnięcie relacji starszeństwa między nimi.

- **Uwaga:**
  W czasie wojny zachodzą dwa konflikty, tj. spotkanie pierwszych i trzecich kart (drugie karty nie są ze sobą porównywane), każde przedłużenie wojny to dodatkowy konflikt.

  Rozstrzygnięcie jednej wojny może nastąpić po jednym lub wielu konfliktach. Np. Gracz A wykłada karty: 2 5 8 4 K Q, a równocześnie gracz B: 3 5 9 4 3 A. Liczba konfliktów jest równa 4.

- **Wymagania:**
  Karty posiadane przez uczestnika gry (a dokładniej - ich kody) tworzą

kolejkę zapisaną w buforze cyklicznym (kołowym, pierścieniowym) o rozmiarze równym liczbie kart w talii (ewentualnie o 1 większym).
Należy rozważyć możliwość skorzystania z funkcji zdefiniowanych w zadaniu 2.2.

- Dla jednoznaczności otrzymywanych wyników konieczne jest dodanie kilku ograniczeń, które MUSZĄ być w programie symulującym grę uwzględnione.

  1. Liczba graczy = 2, liczba kart = 52, wg starszeństwa: (2,3,4,5,6,7,8,9,10,J,Q,K,A)*4 kolory, (choć dodatkową zaletą programu byłaby jego elastyczność - zadawana liczba kart i kolorów).
  2. Kodowanie kart. Kolory kart (pik, kier, karo, trefl) nie mają znaczenia przy ustalaniu relacji starszeństwa między nimi. Dla zbliżenia symulacji do rzeczywistości, każdej karcie jest przypisany unikalny kod - liczby naturalne z zakresu [0, liczba kart-1]. Dwójki mają kody 0 - 3, trójki 4 - 7,..., asy 48 - 51.
     Wskazówka: Która (pojedyncza) operacja arytmetyczna (lub bitowa) wykonana na wartości kodu pozwala na "wyrównanie starszeństwa" tych samych figur (lub blotek) różnych kolorów?
  3. Algorytmy
     - tasowania kart: Wg algorytmu funkcji `rand_permutation()`,
     - rozdawania kart graczom:
       Gracz A otrzymuje pierwszą połowę potasowanej talii, a gracz B drugą połowę - w tym miejscu jednoznacznie jest określony (rozróżniony) gracz A i B,
     - wstawiania do kolejki kart zdobytych w każdym konflikcie (w tym, na wojnie):
       Po rozstrzygnięciu konfliktu, zwycięzca przenosi karty leżące na stole na koniec swojej kolejki kart w kolejności - najpierw swoje począwszy od pierwszej wyłożonej na stół, a później karty przeciwnika, w tej samej kolejności,

     są zadane i nie należy ich zmieniać - każda zmiana spowoduje niezgodność wyników otrzymanych i przewidywanych w automatycznym ocenianiu.

- **Wersja uproszczona gry**
  Różni się od opisanej wyżej wersji standardowej inną reakcją na spotkanie się dwóch kart o tej samej mocy. W wersji standardowej dochodzi do "wojny", natomiast w wersji uproszczonej każdy z graczy zabiera ze stołu swoją kartę i wstawia ją na koniec swojej kolejki kart. Liczba konfliktów jest powiększana także w przypadku spotkania się dwóch równoważnych kart.

- **Wejście**
  Wartość startowa generatora liczb pseudolosowych seed (liczba naturalna typu int).
  Kod wybieranej wersji: 0 - standardowa, 1 - uproszczona.

Maksymalna liczba konfliktów. Jeżeli gra nie zakończy się zwycięstwem jednego z graczy po tej liczbie konfliktów, to gra kończy się wynikiem 0.

- **Wyjście**
  W przypadku:
  - Niedokończenia gry (nie jest wyłoniony zwycięzca po rozstrzygnięciu maksymalnej liczby konfliktów):
    * liczba 0
    * liczba kart gracza A
    * liczba kart gracza B.
  - Nierozstrzygnięcia konfliktu lub wojny (do rozstrzygnięcia ostatniego konfliktu lub wojny zabrakło kart jednemu lub obu graczom):
    * liczba 1
    * liczba kart gracza A
    * liczba kart gracza B.
  - Wygranej gracza A:
    * liczba 2
    * liczba konfliktów, do jakich doszło.
  - Wygranej gracza B:
    * liczba 3
    * ciąg kodów kart jakie gracz B miał po zakończeniu gry – ciąg w kolejności od kodu pierwszej karty przeznaczonej do wyłożenia na stół.

  W sytuacji, gdy gra nie jest dokończona albo jest nierozstrzygnięta, do kart należących do gracza dolicza się jego karty, które położył (i leżą) na stole.

- **Przykład**
  Wejście:
  10444 0 100
  Wyjście:
  3
  43 21 13 10 20 8 48 16 33 23 46 25 18 0 41 14 34 2 49 1 37 27 47 39 5 9 28 19 44 36 38 45 30 24 29 22 6 3 50 17 40 12 15 11 51 26 7 42 35 4 32 31