Podstawy Programowania 2

Kolejki i ich zastosowania

Arkadiusz Chrobot

Katedra Systemów Informatycznych

17 marca 2020

Plan

1. Kolejki i ich klasyfikacja
2. Kolejka FIFO
   - Implementacja w postaci dynamicznej struktury danych
   - Implementacja za pomocą tablicy
3. Testowanie niektórych operacji dynamicznych struktur danych
4. Podsumowanie

Kolejki i ich klasyfikacja

Kolejka FIFO

Kolejki, podobnie jak stos są abstrakcyjną strukturą danych, składającą się z połączonych ze sobą elementów przechowujących dane. Inne są jednak reguły rządzące zarządzaniem tymi elementami. Wyraz *kolejka* jest najczęściej rozumiany jako skrócona nazwa dla *kolejka* FIFO i tak też będzie traktowany na tym wykładzie. Skrót FIFO oznacza w języku angielskim *First In First Out*, czyli w przekładzie polskim: *Pierwszy Nadszedł, Pierwszy Wychodzi*. Konsekwencją tej reguły jest to, że elementy są dodawane do kolejki na tylko jednym z jej końców, a usuwane wyłącznie na drugim. Koniec służący do usuwania elementów nazywany jest *czołem* lub *głową* kolejki (ang. *head*), natomiast przeciwległy koniec, na którym elementy są dodawane, nazywa się ogonem (ang. *tail*) kolejki.

Kolejki i ich klasyfikacja

Kolejki dwustronne

Oprócz kolejek FIFO istnieją także *kolejki dwustronne*, w których operacje dodawania i usuwania elementów mogą dotyczyć obu końców. Wśród nich wyróżniamy dwa dodatkowe podtypy:

- *kolejki dwustronne o ograniczonym wejściu* - elementy mogą być usuwane na obu końcach kolejki, ale dodawane tylko na jednym,
- *kolejki dwustronne o ograniczonym wyjściu* - elementy mogą być dodawane na obu końcach kolejki, ale usuwane tylko na jednym.

Kolejka FIFO

Dalsza część wykładu będzie dotyczyła wyłącznie kolejek FIFO. Takie kolejki mogą być zrealizowane w oparciu o dynamiczne struktury danych lub o tablicę. Oba sposoby implementacji zostaną w ramach tego wykładu przedstawione. W ostatniej części wykładu zostanie zaprezentowany prosty sposób testowania funkcji realizujących niektóre operacje na kolejkach i podobnych dynamicznych strukturach danych. Sposób tworzenia i obsługi kolejek będzie przedstawiony na przykładzie kolejki przechowującej liczby typu `int`. Rozpoczniemy od jej implementacji w postaci dynamicznej struktury danych.

Kolejka FIFO

Tak, jak w przypadku stosu i innych abstrakcyjnych struktur danych do implementacji kolejki będziemy potrzebować typu bazowego oraz funkcji realizujących podstawowe operacje na tej strukturze. W tym ostatnim przypadku niezbędnym minimum są dwie operacje: dodawanie elementu do kolejki i usuwanie elementu z kolejki. Noszą one angielskie nazwy: *enquene* i *dequeue*. Aby ułatwić ich wykonywanie do obsługi kolejek stosuje się dwa wskaźniki. Jeden wskazuje na element początkowy i nazywa się zwyczajowo HEAD, a drugi na element końcowy i nazywa się TAIL. Pierwszy używany jest podczas usuwania elementu z kolejki, a drugi podczas dodawania. W literaturze można spotkać również nazwy FRONT i REAR dla tych wskaźników.

Implementacja w postaci dynamicznej struktury danych

Program ilustrujący działanie kolejki FIFO korzysta z funkcji zarządzających pamięcią na stercie oraz obsługujących standardowe wyjście (monitor). W związku z tym, w jego kodzie źródłowym zawarte są instrukcje włączające pliki nagłówkowe `stdio.h` i `stdlib.h`.

Implementacja w postaci dynamicznej struktury danych

Pliki nagłówkowe

```
1  #include<stdio.h>
2  #include<stdlib.h>
```

Implementacja w postaci dynamicznej struktury danych

Typ bazowy kolejki FIFO

Zdefiniowany w programie typ bazowy kolejki FIFO bazuje na strukturze i jest taki sam, z dokładnością do nazwy, jak typ bazowy stosu. Może on również być w dowolny sposób modyfikowany, zawsze jednak musi zawierać pole wskaźnikowe pozwalające wiązać ze sobą elementy kolejki. Definicja tego typu podana jest na następnym slajdzie.

Implementacja w postaci dynamicznej struktury danych

Typ bazowy kolejki FIFO

```
1  struct fifo_node
2  {
3      int data;
4      struct fifo_node *next;
5  };
```

Implementacja w postaci dynamicznej struktury danych

Wskaźniki HEAD i TAIL

Jak wspomniano wcześniej, aby implementacja kolejki była efektywna niezbędne są dwa wskaźniki. Jeden z nich będzie wskazywał na element początkowy kolejki, a drugi na końcowy. Można te wskaźniki zadeklarować jako osobne zmienne lokalne lub globalne. W opisywanym programie posłużymy się jednak osobną strukturą, w której te wskaźniki będą zaimplementowane jako pola. Definicja typu tej struktury oraz deklaracja zmiennej globalnej tego typu zostały przedstawione na kolejnym slajdzie. Ponieważ zmienne globalne mają zerową wartość początkową, to kolejka jest wstępnie kolejką pustą.

Implementacja w postaci dynamicznej struktury danych

Struktura wskaźników

```
1  struct fifo_pointers
2  {
3      struct fifo_node *head, *tail;
4  } fifo;
```

Implementacja w postaci dynamicznej struktury danych

Operacja *enqueue*

Mając zdefiniowaną strukturę wskaźników oraz typ bazowy kolejki FIFO możemy przystąpić do zdefiniowania funkcji realizujących operacje na tej kolejce. Zaczniemy od operacji dodania nowego elementu do kolejki. Zakładamy, że będzie ona spełniała następujące warunki:

- Jeśli kolejka istnieje, to operacja dodaje nowy element na jej końcu, a jeśli kolejka nie istnieje (jest pusta), to operacja tworzy ją przydzielając pamięć na jej pierwszy element, inicjując go i dodając.
- Jeśli nie uda się stworzyć nowego elementu, to stan kolejki pozostaje bez zmian.
- W wyniku pomyślnego zakończenia operacji kolejka powiększa się o nowy element, a jeśli nie istniała to zostaje utworzona.

Na kolejnym slajdzie znajduje się kod źródłowy funkcji implementującej tę operację.

Implementacja w postaci dynamicznej struktury danych

Funkcja enqueue()

```
1  void enqueue(struct fifo_pointers *fifo, int data)
2  {
3      struct fifo_node *new_node =
4              (struct fifo_node *)malloc(sizeof(struct fifo_node));
5      if(new_node) {
6          new_node->data = data;
7          new_node->next = NULL;
8          if(fifo->head==NULL)
9              fifo->head = fifo->tail = new_node;
10         else {
11             fifo->tail->next=new_node;
12             fifo->tail=new_node;
13         }
14     } else
15         fprintf(stderr,"Nowy element nie został utworzony!\n");
16 }
```

Implementacja w postaci dynamicznej struktury danych

Funkcja `enqueue()`

Opisywana operacja została w programie zaimplementowana w postaci funkcji o takiej samej nazwie. Funkcja ta nie zwraca żadnej wartości. W przypadku niepowodzenia utworzenia i dodania nowego elementu do kolejki wypisuje ona jedynie komunikat na ekranie. Stan kolejki pozostaje w takim przypadku bez zmian. Jeśli pozostanie ona pusta, to nie wpłynie to negatywnie na działanie pozostałych funkcji ją obsługujących, bo wszystkie one, zanim wykonają jakąkolwiek operację na przekazanej im kolejce, najpierw sprawdzają, czy nie jest ona pusta. Do `enqueue()` przekazywana jest przez wskaźnik struktura zawierająca wskaźniki `head` i `tail`. Ich wartości mogą być modyfikowane w funkcji i wtedy muszą być zachowane po jej zakończeniu, stąd taki sposób przekazania. Drugi parametr funkcji służy do przekazania danej, która ma być zapisana w kolejce.

Implementacja w postaci dynamicznej struktury danych

Funkcja `enqueue()`

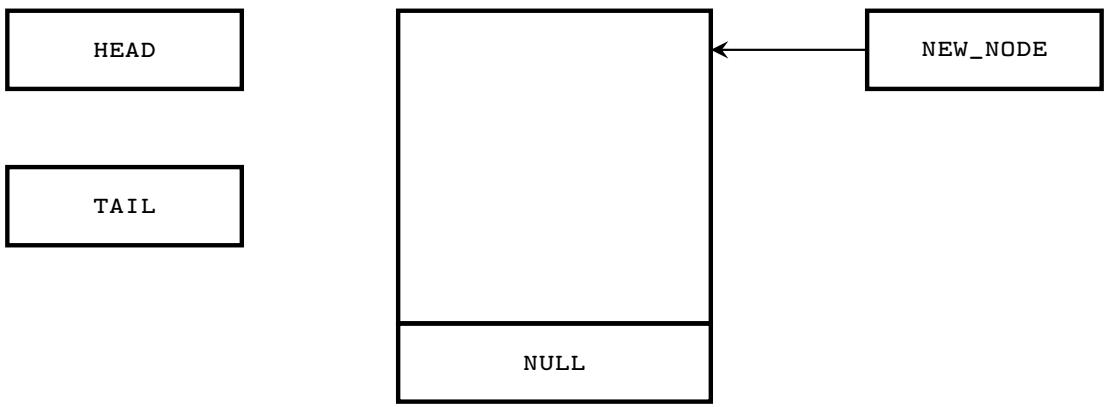
W wierszach nr 3 i 4 funkcji przydzielana jest pamięć na nowy element kolejki. Po sprawdzeniu w wierszu nr 5 czy ten przydział się powiódł wykonywana jest inicjacja pól nowo utworzonego elementu. Pole `next` zyskuje wartość `NULL`, celem zaznaczenia, że będzie to ostatni element w kolejce. Tworząc fragment funkcji `enqueue()` odpowiedzialny za dodanie nowego elementu na koniec kolejki należy rozpatrzeć dwa przypadki:

1. element dodawany jest na koniec istniejącej kolejki
2. element dodawany jest do pustej (nieistniejącej) kolejki.

Rozróżnienie między nimi zachodzi w 8 wierszu funkcji. Jeśli oba wskaźniki na kolejkę mają wartość `NULL`, to znaczy, że występuje drugi przypadek i dlatego do wskaźników zapisywany jest adres nowego elementu, który staje się pierwszym i jednocześnie ostatnim elementem kolejki. Wykonanie tego wariantu ilustrują rysunki na kolejnych slajdach.

Implementacja w postaci dynamicznej struktury danych

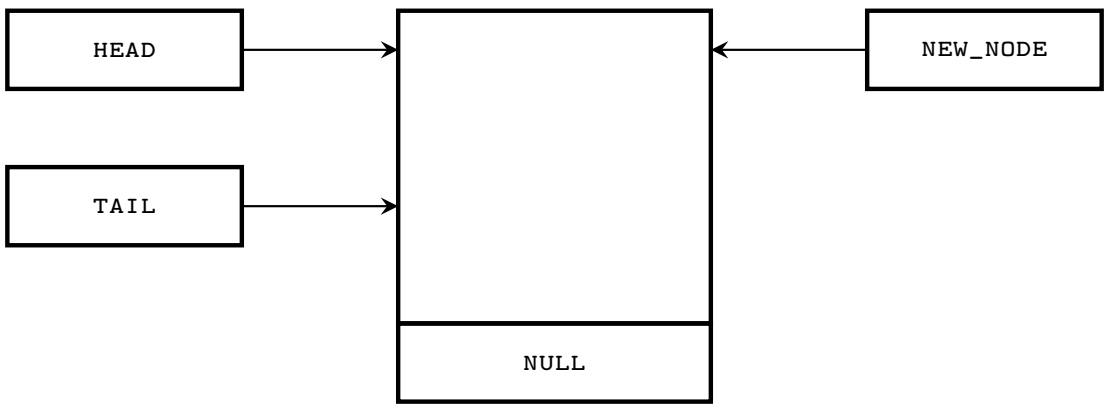
Funkcja `enqueue()` - tworzenie nowej kolejki



Kolejka przed wykonaniem 9 wiersza funkcji `enqueue()`

Implementacja w postaci dynamicznej struktury danych

Funkcja `enqueue()` - tworzenie nowej kolejki



Kolejka po wykonaniu 9 wiersza funkcji `enqueue()`

Implementacja w postaci dynamicznej struktury danych

Funkcja `enqueue()`

Inaczej przebiega dodanie elementu do już istniejącej kolejki. W wierszu nr 11 za pomocą wskaźnika `tail` funkcja `enqueue()` sięga do obecnego ostatniego elementu w kolejce i zapisuje w jego polu `next` adres nowego elementu. W ten sposób staje się on ostatnim elementem kolejki. Aby jednak pozostawić kolejkę w poprawnym stanie funkcja musi przed zakończeniem swojego działania zapewnić, że wskaźnik `tail` nadal będzie wskazywał na ostatni element kolejki. Dlatego w wierszu nr 12 zapisywany jest w nim adres nowego elementu. Proszę zwrócić uwagę, że wiersze 11 i 12 są wzajemnie ze sobą powiązane, co znaczy, że nie można zamienić ich kolejności. Wiersz 12 można zastąpić wierszem: `fifo->tail=fifo->tail->next;` jednak ta instrukcja jest trochę mniej czytelna. Będziemy jednak używać podobnych zapisów w następnych funkcjach i na następnych wykładach. Komunikat w wierszu 15 wyświetlany jest tylko wtedy, gdy utworzenie nowego elementu się nie powiedzie. Wówczas stan kolejki pozostaje bez zmian.

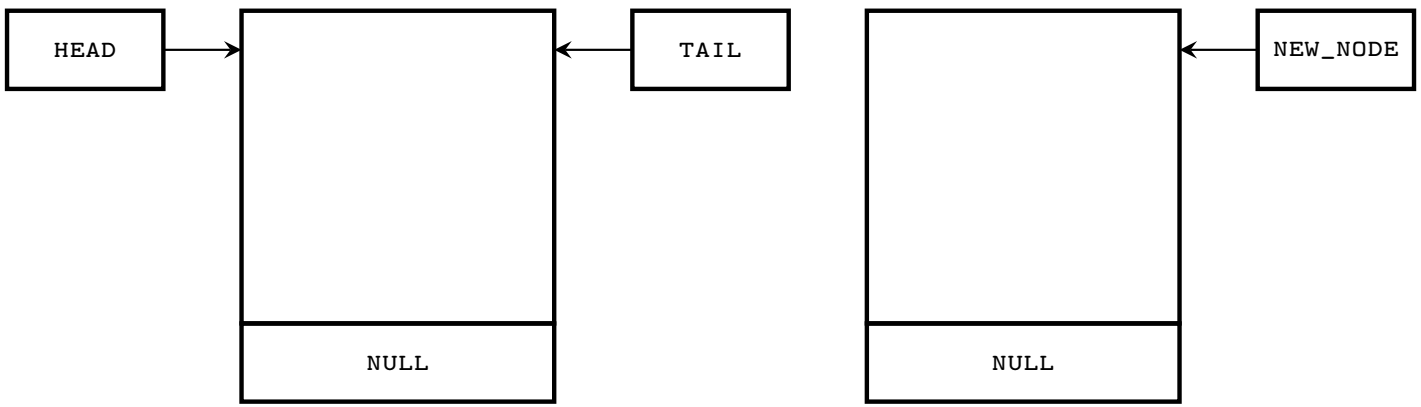
Implementacja w postaci dynamicznej struktury danych

Funkcja `enqueue()`

Kolejne slajdy zawierają ilustrację dodania nowego elementu do jednoelementowej kolejki. Działanie to jest takie samo dla kolejek zawierających więcej niż jeden element.

Implementacja w postaci dynamicznej struktury danych

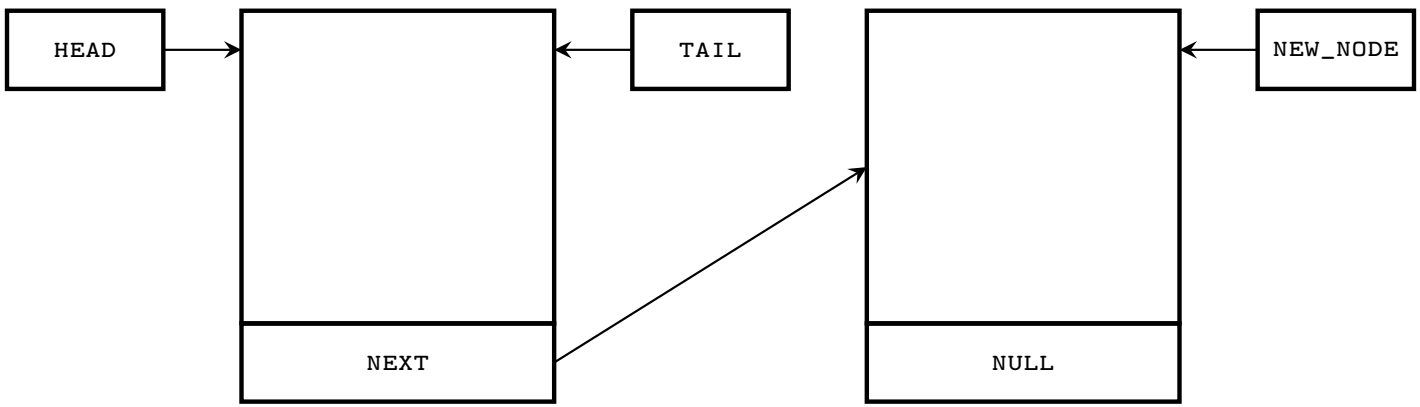
Funkcja enqueue() - dodawanie nowego elementu



Przed wykonaniem 11 wiersza funkcji enquene()

Implementacja w postaci dynamicznej struktury danych

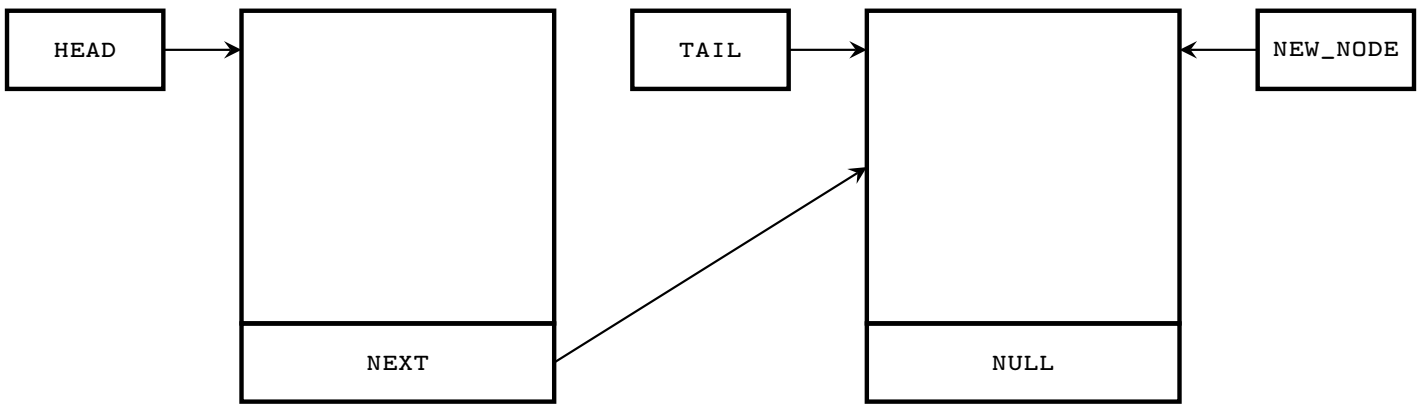
Funkcja enqueue() - dodawanie nowego elementu



Po wykonaniu 11 wiersza funkcji enquene()

Implementacja w postaci dynamicznej struktury danych

Funkcja enqueue() - dodawanie nowego elementu



Po wykonaniu 12 wiersza funkcji enquene()

Implementacja w postaci dynamicznej struktury danych

Operacja *dequeue*

Operacja *dequeue* usuwa element z czoła (początku) kolejki FIFO. Operacja ta powinna spełniać następujące warunki:

- Jeśli kolejka nie istnieje, to stan jej wskaźników po wykonaniu operacji usuwania elementu nie może ulec zmianie - wartości obu powinny wynosić `NULL`.
- Jeśli usuwany jest element z kolejki jednoelementowej, to po wykonaniu tej operacji oba wskaźniki kolejki powinny mieć wartość `NULL`.
- Jeśli usuwany jest element z kolejki składającej się z więcej niż jednego elementu, to kolejka zostaje zmniejszona o jeden element, a po wykonaniu tej operacji wskaźniki poprawnie wskazują początek i koniec kolejki.

Implementację tej operacji w postaci funkcji przedstawiono na następnym slajdzie.

Implementacja w postaci dynamicznej struktury danych

Funkcja dequeue()

```
1  int dequeue(struct fifo_pointers *fifo)
2  {
3      if(fifo->head) {
4          struct fifo_node *tmp = fifo->head->next;
5          int data = fifo->head->data;
6          free(fifo->head);
7          fifo->head=tmp;
8          if(tmp==NULL)
9              fifo->tail = NULL;
10         return data;
11     }
12     return -1;
13 }
```

Implementacja w postaci dynamicznej struktury danych

Funkcja `dequeue()`

Proszę zwrócić uwagę, że funkcja `dequeue()` jest bardzo podobna do funkcji `pop()` implementowanej w przypadku stosu. Podobnie jako ona zwraca liczbę `-1` jeśli zostanie wywołania dla pustej kolejki. Operacja usunięcia elementu z czoła kolejki jest bardzo podobna do operacji usunięcia elementu ze stosu. Różnica występuje tylko w dwóch miejscach. Po pierwsze wskaźniki `head` i `tail` są polami struktury, po drugie, zgodnie z podanymi na poprzednim slajdzie warunkami, konieczne jest nadanie wskaźnikowi `tail` wartości `NULL` po usunięciu elementu z kolejki jednoelementowej. Instrukcje w wierszach 8 i 9 gwarantują spełnienie tego wymogu.

Implementacja w postaci dynamicznej struktury danych

Funkcje `enquene()` i `dequeue()` - podsumowanie

Operacje *enquene* i *dequeue* są podstawowymi operacjami jakie należy zaimplementować dla kolejki FIFO. Stanowią one niezbędne minimum, aby móc posłużyć się taką strukturą danych w programie. Na poprzednich slajdach zaprezentowano przykładowe ich implementacje. Można je zrealizować także w inny sposób, np. funkcja `dequeue()` mogłaby nic nie zwracać lub zwracać jedynie wartość sygnalizującą stan wykonania operacji usuwania elementu. Do doczytania wartości pierwszego elementu kolejki mogłaby być zdefiniowana osobna funkcja. Sposób implementacji tych funkcji zależy od potrzeb programisty i rozwiązywanego przez niego problemu.

Implementacja w postaci dynamicznej struktury danych

Wypisanie wartości elementów na ekran

Implementacja operacji wypisania wartości elementów na ekran komputera nie jest obowiązkowa w implementacji kolejki FIFO, ale jest dosyć wygodnym rozwiązaniem. Na następnych slajdach zostaną przedstawione dwie funkcje, które realizują tę operację.

Implementacja w postaci dynamicznej struktury danych

Funkcja print_queue()

```
1  void print_queue(struct fifo_pointers fifo)
2  {
3      while(fifo.head) {
4          printf("%d ",fifo.head->data);
5          fifo.head = fifo.head->next;
6      }
7      puts("");
8  }
```

Implementacja w postaci dynamicznej struktury danych

Funkcja `print_queue()`

Do tej funkcji struktura wskaźników kolejki jest przekazywana przez wartość. To dlatego, że wygodniej będzie użyć w niej wskaźnika `head` do iterowania, czyli „poruszania się" po elementach kolejki, ale wiąże się to z modyfikacją jego wartości. Te modyfikacje nie mogą jednak „wyjść" poza funkcję `print_queue()`, co zapewnia nam taki sposób przekazania. Gdyby wskaźnik `head` po zakończeniu działania funkcji miał inną wartość niż przed jej rozpoczęciem, to zostałby zgubiony adres elementu początkowego kolejki. W funkcji wykonywana jest pętla `while` tak długo, jak długo wskaźnik `head` będzie miał wartość różną od `NULL`, co oznacza, że w kolejce będą jeszcze elementy, których wartość nie została jeszcze wypisana na ekran. Samo wypisanie wykonywane jest w wierszu nr 4. W piątym wierszu wskaźnik `head` jest „przestawiany" na następny element kolejki poprzez zapisanie w nim adresu z pola `next` wskazywanego przez niego elementu.

Implementacja w postaci dynamicznej struktury danych

Funkcja print_queue() - wersja z pętlą for

```
1  void print_queue_with_for(struct fifo_pointers fifo)
2  {
3      for(;fifo.head;fifo.head=fifo.head->next)
4          printf("%d ",fifo.head->data);
5      puts("");
6  }
```

Implementacja w postaci dynamicznej struktury danych

Funkcja `print_queue()` - wersja z pętlą `for`

Ta sama operacja wypisania elementów kolejki FIFO może być zrealizowana w języku C za pomocą pętli `for`, tak jak w funkcji zamieszczonej na poprzednim slajdzie. Zmienną sterującą dla tej pętli jest wskaźnik `head`. Proszę zwrócić uwagę, że pomijamy wyrażenie inicjujące. Warunkiem kontynuacji jest to, że wskaźnik `head` będzie miał wartość różną od `NULL`, a wyrażenie zmieniające wartość tego wskaźnika przypisuje mu adres kolejnego elementu kolejki. Zapis ten jest bardziej zwięzły niż w przypadku pętli `while`, ale trochę mniej czytelny.

Implementacja w postaci dynamicznej struktury danych

Przykład użycia - funkcja `main()`

```
1  int main(void)
2  {
3      int i;
4      for(i=0;i<20;i++)
5          enqueue(&fifo,i);
6      print_queue_with_for(fifo);
7      while(fifo.head)
8          printf("%d ",dequeue(&fifo));
9      puts("");
10     return 0;
11 }
```

Implementacja w postaci dynamicznej struktury danych

Przykład użycia - funkcja `main()`

W funkcji `main()` programu wywoływane są wszystkie wcześniej zdefiniowane funkcje do obsługi kolejki FIFO, za wyjątkiem `print_queue()`. Można jej wywołaniem zastąpić wywołanie `print_queue_with_for()` lub po prostu dodać je do programu. W wierszach 4 i 5 dodawane są do kolejki elementy zawierające liczby naturalne od 0 do 19, następnie wypisywana jest zawartość kolejki na ekran (wiersz 6), usuwane są z niej wszystkie elementy i ponownie wypisywane są na ekran ich wartości (wiesze 7 i 8).

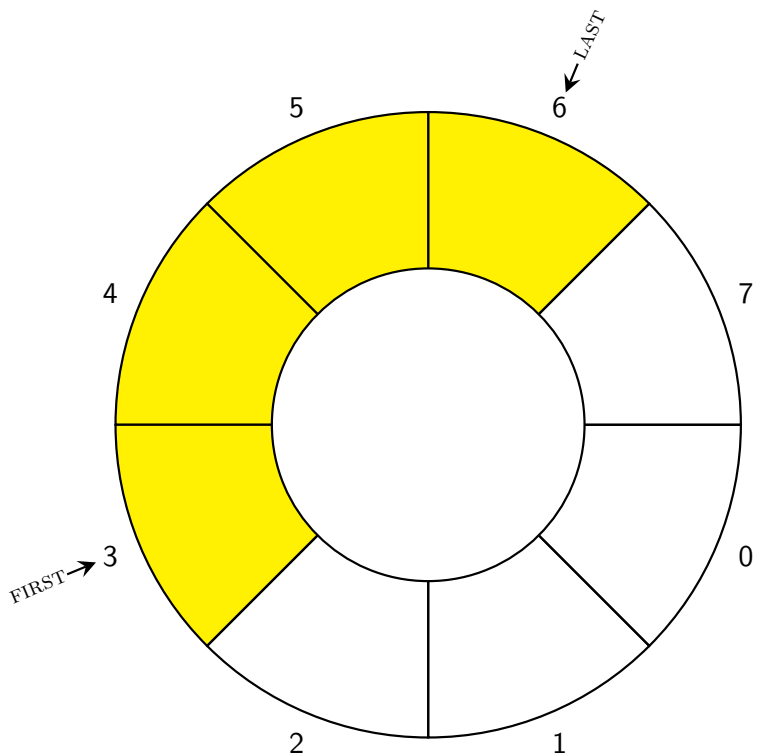
Kolejka FIFO

Implementacja za pomocą tablicy

Kolejka FIFO może zostać zrealizowana za pomocą tablicy. Będzie ona miała pojemność ograniczoną liczbą elementów tej tablicy, ale poza tym będzie zachowywała się jak jej odpowiedniczka zrealizowana za pomocą dynamicznej struktury danych. Kolejkę całkowicie zapełnioną będziemy nazywać *kolejką pełną*. Przykładowa implementacja takiej kolejki zostanie opisana na przykładzie programu, który tak jak ten wcześniej zaprezentowany, przechowuje w kolejce FIFO liczby całkowite. Wskaźniki `head` i `tail` zostaną w tej kolejce zastąpione zmiennymi indeksującymi o nazwach `first` i `last`. Aby ułatwić realizację takiej kolejki można potraktować tablicę jako tablicę cykliczną, czyli taką, która nie ma początku ani końca. Kolejkę uzyskaną za pomocą takiej tablicy przedstawia kolejny slajd.

Kolejka FIFO

Implementacja za pomocą tablicy



Częściowo wypełniona kolejka FIFO

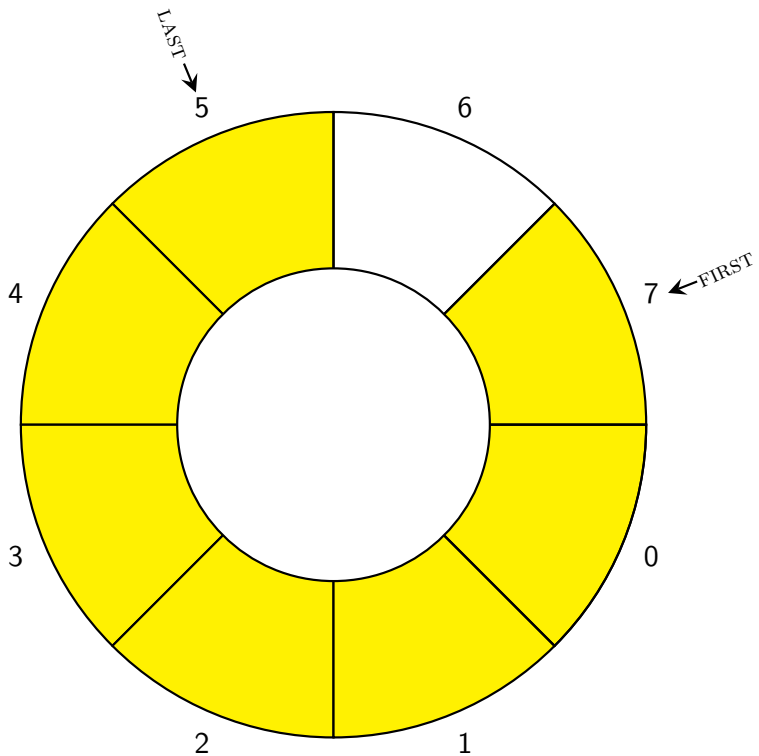
Kolejka FIFO

Implementacja za pomocą tablicy

Takie podejście ma dwie konsekwencje. Z jednej strony wartości obu indeksów podczas operacji dodawania i usuwania elementów są jedynie zwiększane o jeden. Z drugiej strony należy ustalić w jaki sposób będziemy wykrywać, że kolejka jest pusta lub pełna. Można zliczać, za pomocą osobnej zmiennej, ile obecnie jest elementów w kolejce, ale inne rozwiązanie tego problemu podali Alfred V. Aho, John E. Hopcroft i Jeffrey D. Ullman w książce „Algorytmy i struktury danych". Zaprezentowany program bazuje na ich rozwiązaniu. Pełne i puste kolejki według ich propozycji są zilustrowane na kolejnych slajdach.

Kolejka FIFO

Implementacja za pomocą tablicy



Pełna kolejka FIFO

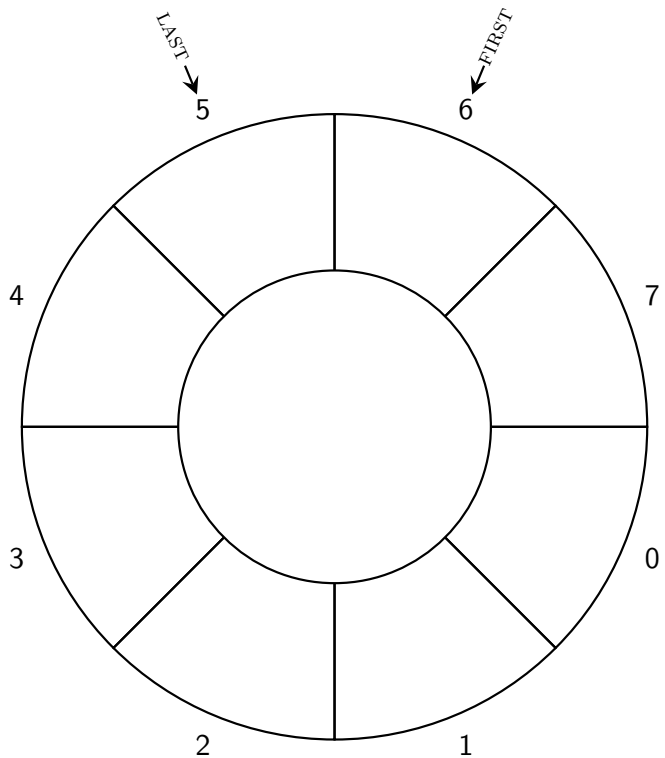
Kolejka FIFO

Implementacja za pomocą tablicy

Pełna kolejka to taka, w której indeksy `last` i `first` różnią się wartościami o 2 modulo liczba elementów tablicy. Proszę zwrócić uwagę, że przy takiej definicji warunku wypełnienia kolejki jeden element tablicy zawsze będzie pusty, tak jak pokazano to na rysunku.

Kolejka FIFO

Implementacja za pomocą tablicy



Pusta kolejka FIFO

Kolejka FIFO

Implementacja za pomocą tablicy

Pusta kolejka to taka, w której indeksy `last` i `first` mają wartości różniące się o `1` modulo liczba elementów tablicy.

Kolejka FIFO

Implementacja za pomocą tablicy - struktura kolejki

```
1  #include<stdio.h>
2  #include<stdbool.h>
3
4  #define FIFO_SIZE 20
5
6  struct queue
7  {
8      int elements[FIFO_SIZE], first, last;
9  } fifo;
```

Kolejka FIFO

Implementacja za pomocą tablicy - struktura kolejki

Poprzedni slajd zawiera początek programu implementującego kolejkę w oparciu o tablicę. Plik nagłówkowy `stdlib.h` został zastąpiony plikiem `stdbool.h`, ponieważ jedna z funkcji będzie zwracać wartość typ `bool`, a w programie nie będą wykorzystywane funkcje zarządzające pamięcią na stercie. Stała `FIFO_SIZE` określa liczę elementów tablicy na której będzie oparta kolejka. Pojemność tej kolejki będzie o jeden element mniejsza. Tablica i indeksy kolejki zostały w programie zaimplementowane jako pola struktury. Zmienna tego typu strukturalnego o nazwie `fifo` jest po prostu tą kolejką.

Kolejka FIFO

Implementacja za pomocą tablicy - funkcja add_one()

```
1  int add_one(int index)
2  {
3      return (index+1)%FIFO_SIZE;
4  }
```

Kolejka FIFO

Implementacja za pomocą tablicy - funkcja `add_one()`

Funkcja `add_one()` służy do zwiększania wartości indeksów kolejki o jeden. Dzięki zastosowaniu operatora reszty z dzielenia w tej operacji, wartość żadnego z tych indeksów nie przekroczy dopuszczalnego zakresu. Funkcja przez parametr przyjmuje bieżącą wartość indeksu, a zwraca następną.

Kolejka FIFO

Implementacja za pomocą tablicy - funkcja `make_empty()`

```
1  void make_empty(struct queue *fifo)
2  {
3      fifo->first = 0;
4      fifo->last = FIFO_SIZE-1;
5  }
```

Kolejka FIFO

Implementacja za pomocą tablicy - funkcja `make_empty()`

Funkcja `make_empty()` dokonuje inicjacji kolejki, czy nadania jej indeksom wartości „neutralnych”. Indeks `first` będzie po jej wykonaniu określał pierwszy element tablicy, a indeks `last` ostatni.

Kolejka FIFO

Implementacja za pomocą tablicy - funkcja `is_empty()`

```
1  bool is_empty(struct queue fifo)
2  {
3      return add_one(fifo.last)==fifo.first;
4  }
```

Kolejka FIFO

Implementacja za pomocą tablicy - funkcja `is_empty()`

Funkcja `is_empty()` zwraca wartość `true` jeśli nie ma elementów w kolejce lub `false` jeśli jest choć jeden. Jej działanie polega na sprawdzeniu, czy zwiększona za pomocą wywołania funkcji `add_one()` wartość indeksu `last` odpowiada wartości indeksu `first`[1]. Jeśli tak jest, to kolejka jest pusta.

[1] Proszę porównać z odpowiednią ilustracją na poprzednich slajdach.

Kolejka FIFO

Implementacja za pomocą tablicy - funkcja `first_one()`

```
1  int first_one(struct queue fifo)
2  {
3      if(is_empty(fifo)==true)
4          return -1;
5      else
6          return fifo.elements[fifo.first];
7  }
```

Kolejka FIFO

Implementacja za pomocą tablicy - funkcja `first_one()`

W opisywanym programie operacja *dequeue* będzie polegała jedynie na usunięciu pierwszego elementu kolejki. Funkcja `first_one()` zwraca wartość pierwszego elementu kolejki. Jest to ten, który określany jest przez indeks `first`. Jeśli kolejka byłaby pusta, to funkcja zwraca wartość `-1`.

Kolejka FIFO

Implementacja za pomocą tablicy - funkcja enqueue()

```
1  void enqueue(struct queue *fifo, int data)
2  {
3
4      if(add_one(add_one(fifo->last))!=fifo->first)
5      {
6          fifo->last = add_one(fifo->last);
7          fifo->elements[fifo->last] = data;
8      } else
9          fprintf(stderr, "Kolejka jest pełna!\n");
10 }
```

Kolejka FIFO

Implementacja za pomocą tablicy - funkcja `enqueue()`

Funkcja `enqueue()` dodaje do kolejki nowy element, zapisując w nim wartość przekazaną jej przez parametr `data`. Zanim to jednak nastąpi funkcja ta najpierw upewnia się, że kolejka nie jest pełna. Robi to dwukrotnie zwiększając za pomocą funkcji `add_one()` wartość indeksu `last` i porównując wynik z indeksem `first`. Jeśli te wartości są sobie równie, to kolejka jest pełna i nie można dodać do niej nowego elementu[2]. W takim wypadku na ekran wypisywany jest odpowiedni komunikat. Jeśli jednak kolejka nie jest pełna, to funkcja najpierw zwiększa wartość indeksu `last` przy pomocy funkcji `add_one()`, a następnie zapisuje w elemencie tablicy określonym tą nową wartością indeksu wartość parametru `data` (wiersze 6 i 7).

[2] Proszę porównać z odpowiednia ilustracją znajdującą się na poprzednich slajdach.

Kolejka FIFO

Implementacja za pomocą tablicy - funkcja dequeue()

```
1  void dequeue(struct queue *fifo)
2  {
3      if(is_empty(*fifo))
4          fprintf(stderr, "Kolejka jest pusta!\n");
5      else
6          fifo->first = add_one(fifo->first);
7  }
```

Kolejka FIFO

Implementacja za pomocą tablicy - funkcja `dequeue()`

Funkcja `dequeue()` w tej implementacji kolejki nie zwraca żadnej wartości, po prostu likwiduje początkowy element. Najpierw jednak sprawdza, czy kolejka na której ma być przeprowadzona operacja nie jest pusta. Jeśli okazałoby to się być nieprawdą, to funkcja wyświetla na ekranie odpowiedni komunikat i kończy swe działanie. W przeciwnym przypadku usunięcie elementu sprowadza się do zwiększenia przy pomocy wywołania funkcji `add_one()` wartości indeksu `first` (wiersz nr 6).

Kolejka FIFO

Implementacja za pomocą tablicy - funkcja main()

```
1  int main(void)
2  {
3      int i;
4      make_empty(&fifo);
5      for(i=0;i<FIFO_SIZE-1;i++)
6          enqueue(&fifo,i);
7      while(!is_empty(fifo)) {
8          printf("%d ",first_one(fifo));
9          dequeue(&fifo);
10     }
11     return 0;
12 }
```

Kolejka FIFO

Implementacja za pomocą tablicy - funkcja `main()`

W funkcji `main()` programu kolejka jest najpierw inicjowana przy pomocy wywołania `make_empty()`, a następnie w pętli `for` dodawane są do niej elementy przy pomocy wywoływania funkcji `enqueue()`. Kolejka może ich pomieścić tylko 19. Po zakończeniu pętli `for` wykonywana jest pętla `while`, gdzie wartości elementów kolejki są odczytywane za pomocą wywołania funkcji `first_one()` oraz usuwane z kolejki za pomocą wywołania funkcji `dequeue()`. Tego typu implementacje kolejek FIFO były realizowane w językach programowania, które nie zapewniały dynamicznego przydziału pamięci oraz w systemach, które nie posiadają dostatecznie dużo dostępnej pamięci operacyjnej (np. w mikrokontrolerach). W ten sposób jest zorganizowany tzw. bufor klawiatury, czyli miejsce, gdzie sterownik umieszcza informacje o naciśniętych przez użytkownika klawiszach, które następnie odczytuje procesor komputera. Jest to przykład kolejki o ograniczonej pojemności, dodatkowo obsługiwanej sprzętowo.

Testowanie operacji na strukturach danych

Korzystanie ze zmiennych i struktur dynamicznych jest dosyć złożonym zadaniem. Łatwo popełnić błędy implementując operacje na stosie, kolejce lub innej strukturze danych, a lokalizacja i usunięcia takich defektów może być trudnym przedsięwzięciem. Istnieje jednak prosty sposób na przetestowanie funkcji implementujących operacje nie wymagające przydziału lub zwalniania pamięci, takie jak np. `print_queue()`. Wystarczy stworzyć kolejkę lub inną strukturę danych z elementów będących zwykłymi zmiennymi globalnymi lub lokalnymi i sprawdzić na niej działanie wymienionej funkcji. W ograniczonym zakresie ten sposób można zastosować również do funkcji realizujących pozostałe operacje. Funkcja znajdująca się na następnym slajdzie realizuje tego typu rozwiązanie.

Testowanie operacji na strukturach danych

```
1  void print_queue_test(struct fifo_pointers *fifo)
2  {
3      struct fifo_node front, middle, rear;
4
5      front.data = 1;
6      front.next = &middle;
7      middle.data = 2;
8      middle.next = &rear;
9      rear.data = 3;
10     rear.next = NULL;
11
12     fifo->head = &front;
13     fifo->tail = &rear;
14     print_queue(*fifo);
15     fifo->head = fifo->tail = NULL;
16 }
```

Testowanie operacji na strukturach danych

W funkcji `print_queue_test()` zadeklarowane są trzy zmienne typu `struct fifo_node` o nazwach `front` (przód), `middle` (środek) i `rear` (tył). Zostaną one umieszczone na pozycjach odpowiadających ich nazwom w trójelementowej kolejce. Ta kolejka tworzona jest w wierszach 5 - 9, tzn. inicjowane są pola `data` tych elementów oraz `next`. Proszę zwrócić uwagę, ze wspomniane pola wskaźnikowe dwóch pierwszych elementów są inicjowane adresem kolejnego elementu w kolejce, a w trzecim, ostatnim elemencie umieszczana jest wartość `NULL`. W wierszach 12 i 13 inicjowane są wskaźniki kolejki adresami pierwszego i ostatniego elementu. W tej sposób powstaje kolejka, na której można testować działanie funkcji `print_queue()`, bez obawy, że ewentualne defekty w tej funkcji mogą uszkodzić strukturę kolejki i doprowadzić do wycieków pamięci. Wiersz nr 15 zeruje wskaźniki kolejki, która jest tworzona ze zmiennych lokalnych funkcji, a w związku z tym po zakończeniu jej działania przestaje istnieć.

Podsumowanie

Kolejki implementowane w postaci dynamicznych struktur danych lub w oparciu o tablice mają liczne zastosowania. Systemy operacyjne wykorzystują je do szeregowania procesów, realizacji specjalnych zmiennych nazywanych semaforami, organizacji zdań i wielu innych celów. Podobnie zastosowania mają w oprogramowaniu współbieżnym, czy kompilatorach i innych programach. Są również implementowane sprzętowo, jak wspomniany wcześniej bufor klawiatury.

Dosyć często, aby uprościć działanie i tworzenie kodu obsługującego kolejki programiści decydują się na stworzenie pojedynczego elementu kolejki, którego istnienie jest zawsze zagwarantowane. Ten element jest zazwyczaj atrapą, tzn. nie przechowuje istotnych danych i nazywany jest *wartownikiem*. Kolejka zrealizowana z jego pomocą nazywa się *kolejką z wartownikiem*. To rozwiązanie można zastosować również w przypadku stosu.

Pytania

?

KONIEC

Dziękuję Państwu za uwagę!