Wykład 10

- Struktury liniowe o zmiennym podłożu
  - Stos, kolejka
  - Implementacje struktur
- Zastosowania stosu
  - Derekursywacja funkcji
  - Obliczanie wartości wyrażenia

Struktury liniowe o zmiennym podłożu

- Są to struktury **nie posiadające adresacji**.
- Dostęp do poszczególnych elementów struktury jest organizowany poprzez **wyróżnione elementy**.
- Do tych struktur należą:
  - stos,
  - kolejka,
  - talia (dostęp do elementów z obu stron).

Stos

- Dostęp poprzez wierzchołek stosu.
- Operacje proste (4 operacje):
  - inicjalizacja stosu – `init(s)`,
  - testowanie czy stos jest pusty – `empty(s)`,
  - dołączanie elementu na wierzchołek – `push(s,e)`,
  - pobieranie elementu z wierzchołka – `pop(s)`.
- Uwaga:
  - Stos określany jest jako struktura LIFO (Last In First Out).
- Zastosowanie:
  - przeglądanie grafu,
  - obliczania wartości wyrażenia,
  - usuwanie rekurencji z programu.

Kolejka

- Dostęp poprzez początek kolejki i koniec kolejki.
- Operacje proste (4 operacje):
  - inicjalizacja kolejki – `init(k)`,
  - testowanie czy kolejka jest pusta – `empty(k)`,
  - dołączanie elementu na koniec – `put(k,e)`,
  - pobieranie elementu z początku – `get(k)`.
- Uwaga:
  - Kolejka określana jest jako struktura FIFO (First In First Out).
- Zastosowanie:
  - przeglądanie grafu,
  - kolejka zadań o jednakowym priorytecie.

Implementacja struktur

- Tablicowa
  - Zalety: szybkość, prosta implementacja
  - Wady: ograniczenia pamięciowe
- Wskaźnikowa
- Mieszana

Stos implementacja tablicowa

```
struct stos {
  int t[MAX];
  int size;
};

void init(stos &st) { st.size=0; }
void push(stos &st, int el) { st.t[st.size++]=el; } // brak kontr
int pop(stos &st) { return st.t[--st.size]; }    // brak kontroli
bool empty(stos &st) { return (st.size==0); }


int main() {
  stos s;

  init(s);
  for (int i=0; i<10; i++) push(s,i*i);
  while (!empty(s)) cout<<pop(s)<<endl;
}
```

Stos implementacja tablicowa

```
struct stos {
  int t[MAX];
  int size;

  void init();
  void push(int el);
  int pop();
  bool empty();
};

void stos::init() { size=0; }
void stos::push(int el) { t[size++]=el; }   // bez kontroli
int stos::pop() { return t[--size]; }       // bez kontroli
bool stos::empty() { return (size==0); }

int main() {
  stos s;
  s.init();
  for (int i=0; i<10; i++) s.push(i*i);
  while (!s.empty()) cout<<s.pop()<<endl;
}
```

7

Stos implementacja tablicowa

```
struct stos {
  int t[MAX];
  int size;

  stos();
  void push(int el);
  int pop();
  bool empty();
};

stos::stos() { size=0; }
void stos::push(int el) { t[size++]=el; }   // bez kontroli
int stos::pop() { return t[--size]; }       // bez kontroli
bool stos::empty() { return (size==0); }

int main() {
  stos s; // konstruktor wywołany automatycznie
  s.init();
  for (int i=0; i<10; i++) s.push(i*i);
  while (!s.empty()) cout<<s.pop()<<endl;
}
```

8

Kolejka implementacja tablicowa

```
struct kolejka {
  int t[MAX];
  int p,k;
};

void init(kolejka &kol) { kol.p=-1; }
void put(kolejka &kol, int el) {
  if (kol.p==-1) kol.t[ kol.p=kol.k=0 ]=el; // wstaw do pustej
  else {
    kol.k=(kol.k+1)%MAX;
    if (kol.p==kol.k) error();
    kol.t[kol.k]=el;
  }
}
int get(kolejka &kol) {
  if (kol.p==-1) error();
  int tmp=kol.t[kol.p];
  if (kol.p==kol.k) kol.p=-1; else kol.p=(kol.p+1)%MAX;
  return tmp; }
bool empty(kolejka &kol) { return (kol.p==-1); }
```

9

Kolejka i stos w Pythonie

```
>>> from collections import deque

>>> stack = deque()
>>> stack
deque([])

>>> stack.append(23)
>>> stack.append(29)

>>> stack
deque([23,29])

>>> stack.pop()
29

>>> stack
deque([23])
```

```
>>> queue = deque()

>>> queue.append(23)
>>> queue.append(29)
>>> queue
deque([23,29])

>>> queue.popleft()
23

>>> queue.appendleft(6)

>>> queue
deque([6,29])
```

10

Sortowania przez podział

Sortowanie przez podział – Quicksort

- wybieramy element dzielący, względem którego dzielimy tablicę na elementy mniejsze i większe, wymieniając elementy położone daleko od siebie, operację powtarzamy dla obu części tablicy, aż do podziału na części o długości 1.
- wersja rekurencyjna i nierekurencyjna,

11

Sortowania przez podział

```
T[N]    # sortowane elementy l..p

def qs(T, l, p):   # qs(T,0,N-1)

  i = l
  j = p
  x = T[(l+p)//2]
  while i<=j:
    while T[i]<x: i += 1
    while T[j]>x: j -= 1
    if i<=j:
      T[j],T[i] = T[i],T[j]
      i += 1
      j -= 1

  if l<j: qs(T, l, j)
  if i<p: qs(T, i, p)
# end
```

12

Sortowania przez podział

```
def qs(T):
  stos = []
  stos.append((0,N-1))
  while len(stos)>0:
    l,p = stos.pop()

    i = l
    j = p
    x = T[(l+p)//2]
    while i<=j:
      while T[i]<x: i += 1
      while T[j]>x: j -= 1
      if i<=j:
        T[j],T[i] = T[i],T[j]
        i += 1
        j -= 1

    if l<j: stos.append((l,j))
    if i<p: stos.append((i,p))
# end
```

13

Postać wyrażenia

Elementy wyrażenia:
- Zmienne: x, y, z, ...
- Stałe: 12, -5, ...
- Operatory dwuargumentowe: ^ * / + - (priorytet i łączność)
- Minus unarny: -
- Nawiasy: ( )

Postać wyrostkowa, infiksowa:
(x+y) - (z*3)

Postać przedrostkowa, prefiksowa, notacja Łukasiewicza:
- (+(x,y), *(z,3))

Postać przyrostkowa, postfiksowa (Reverse Polish Notation, RPN):
x y + z 3 * -

Obliczanie wyrażeń

```
procedure onp(w) # zamienia wyrażenie infiksowe na postfiksowe
   while w ?:= 2(="(",tab(bal(')')),pos(-1))

   w ? every p:=bal('+-')
   if \p then return onp(w[1:p])||onp(w[p+1:0])||w[p]

   w ? every p:=bal('*/')
   if \p then return onp(w[1:p])||onp(w[p+1:0])||w[p]

   w ? p:=bal('^')
   if \p then return onp(w[1:p])||onp(w[p+1:0])||w[p]

   return(w)
end
```

Obliczanie wyrażeń

```
procedure value(w)
  st:=[] # inicjalizacja stosu
  every elem:=!w do
    if any('+-*/^',elem) then
      push(st,a:=pop(st) & elem(pop(st),a))
    else
      push(st,variable(el))

  return pop(st)
end

global x,y,z
procedure main()
  x:=2
  y:=3
  z:=5
  while write(value(onp(read())))
end
```

Obliczanie wyrażeń

```
mag@wierzba:/home/mag$ ./wyr
x+y
5
x*(y+z)
16
x^y^x
512
x^(y+z)/x
128
```

Pytania i zadania

1. Dana jest funkcja określona rekurencyjnie:
   f(0,b) = b+1
   f(a,0) = f(a-1, 1) a>0
   f(a,b) = f(a-1, f(a,b-1)) a>0, b>0
   Proszę napisać funkcję w wersji rekurencyjnej oraz iteracyjnej.

2. Dana jest tablica T[N] zawierająca nieuporządkowane liczby naturalne. Proszę zmodyfikować funkcję qs, tak aby szybko wyznaczyć sumę k najmniejszych wartości z tablicy. Przykładowe dane to N=1000000 i k=50.

3. Proszę napisać w języku Python program zamieniający wyrażenie z postaci infiksowej na postać postfiksową, a następnie wyliczający wartość tego wyrażenia. Proszę nie używać funkcji eval.