Structuri de Date și Algoritmi
Stive și cozi

Mihai Nan

Departamentul de Calculatoare
Facultatea de Automatică și Calculatoare
Universitatea POLITEHNICA din București

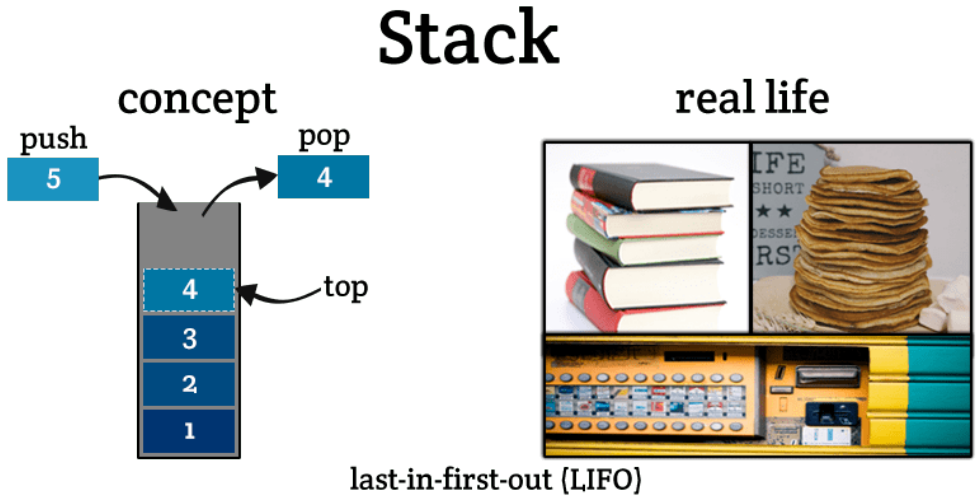
Anul Universitar 2022–2023

Conținutul cursului

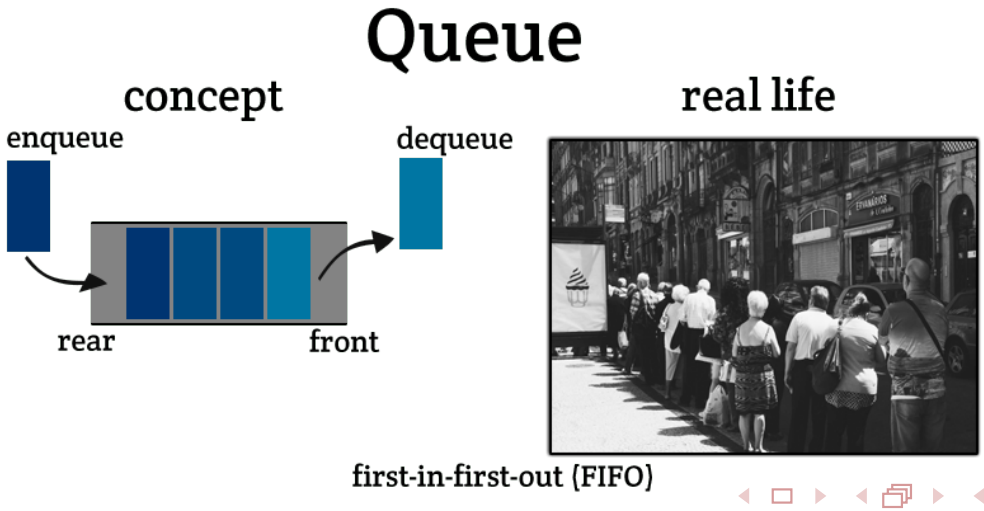
1. **Introducere**
2. **Stiva**
   - Motivație
   - TAD pentru stivă
   - Implementare folosind vectori
   - Implementare folosind liste
3. **Coada**
   - Motivație
   - TAD pentru coadă
   - Implementare folosind vectori
   - Implementare folosind liste
4. **Probleme tip interviu**

Introducere

1. **Stiva** – ultimul venit → primul servit

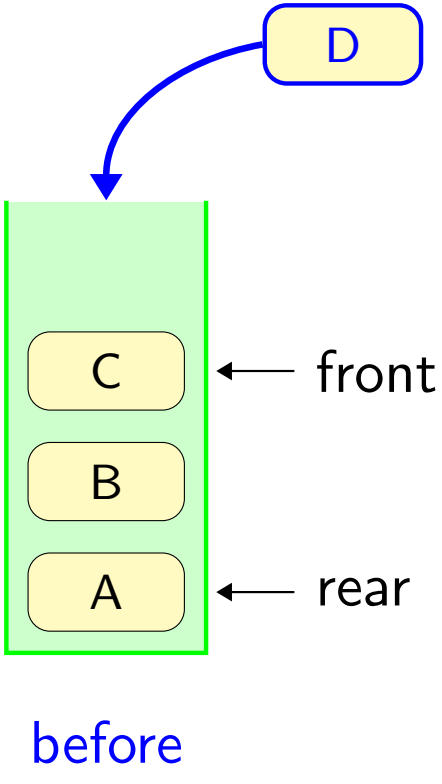


2. **Coada** – primul venit → primul servit



Stiva – Motivație

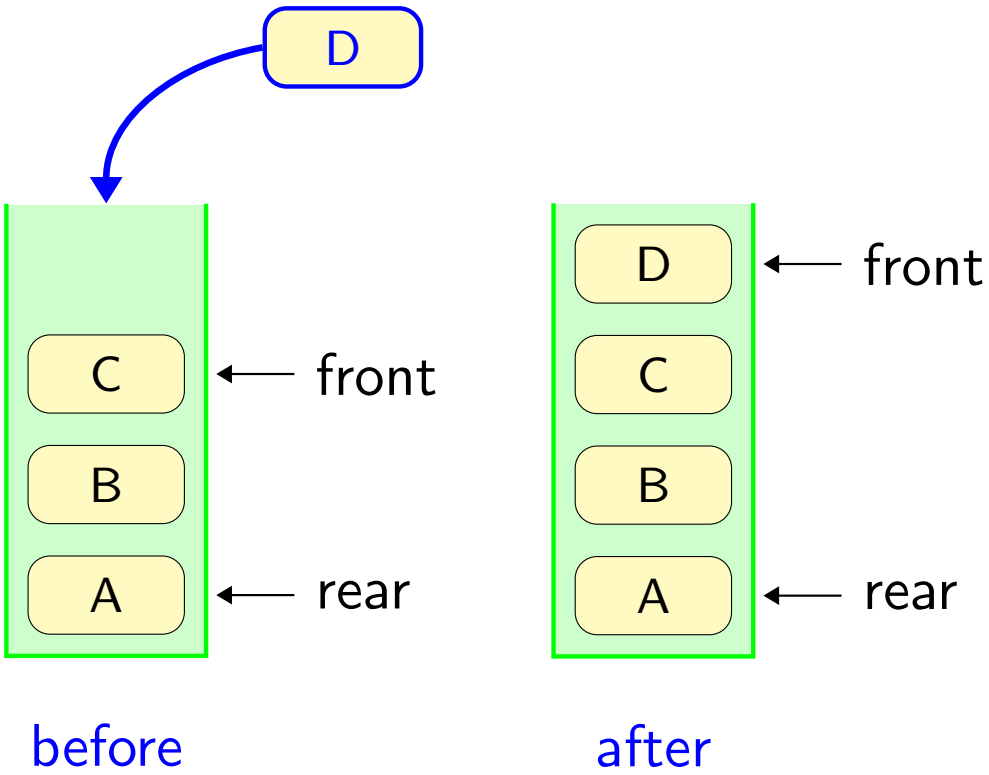
- O structură de date care funcționează după principiul **LIFO** (**L**ast **I**n **F**irst **O**ut)
- Este o structură de date pentru care putem realiza următoarele operații:
  1. Adăugarea unui element în vârful stivei



before

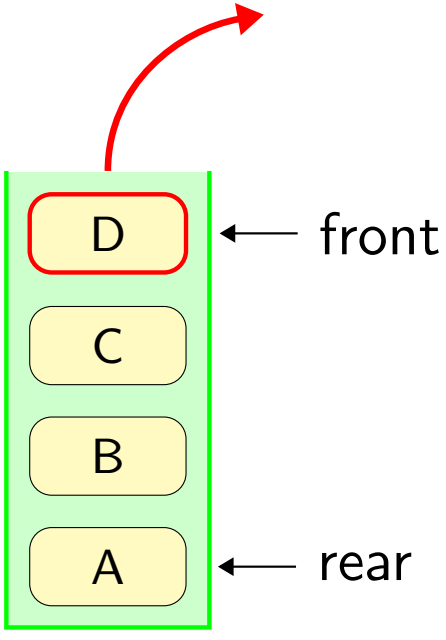
Stiva – Motivație

- O structură de date care funcționează după principiul **LIFO** (**L**ast **I**n **F**irst **O**ut)
- Este o structură de date pentru care putem realiza următoarele operații:
  1. Adăugarea unui element în vârful stivei



Stiva – Motivație

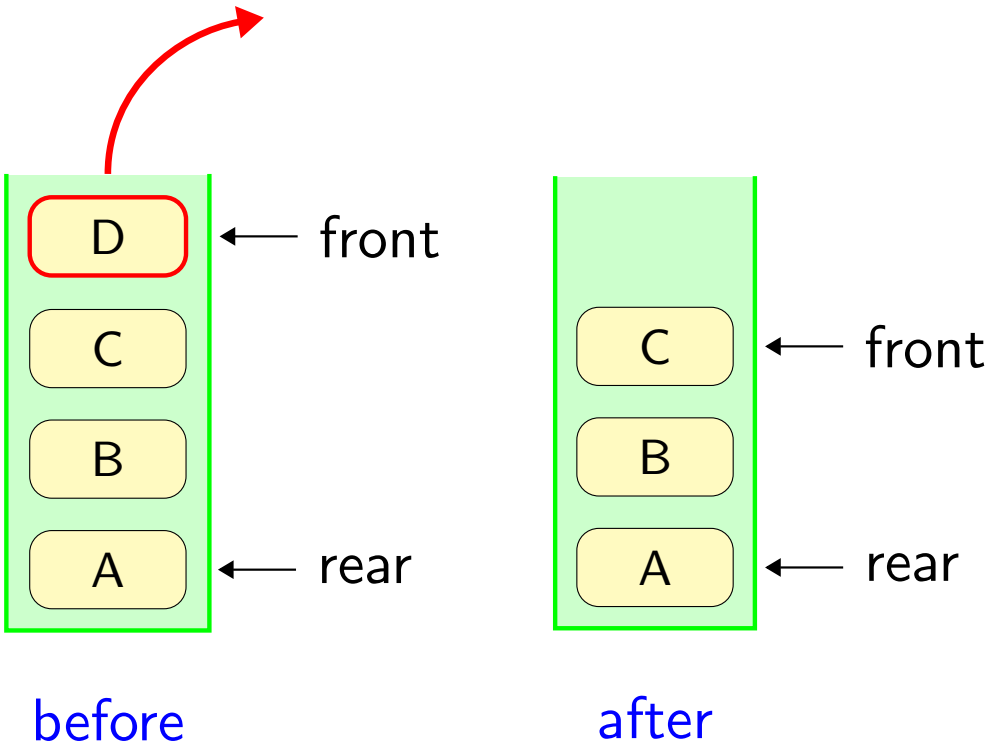
- O structură de date care funcționează după principiul **LIFO** (**L**ast **I**n **F**irst **O**ut)
- Este o structură de date pentru care putem realiza următoarele operații:
  1. Adăugarea unui element în vârful stivei
  2. Eliminarea elementului din vârful stivei



before

Stiva – Motivație

- O structură de date care funcționează după principiul **LIFO** (**L**ast **I**n **F**irst **O**ut)
- Este o structură de date pentru care putem realiza următoarele operații:
  1. Adăugarea unui element în vârful stivei
  2. Eliminarea elementului din vârful stivei



Stiva – Motivație

- O structură de date care funcționează după principiul **LIFO** (**L**ast **I**n **F**irst **O**ut)
- Este o structură de date pentru care putem realiza operații precum:
  1. Adăugarea unui element în vârful stivei
  2. Eliminarea elementului din vârful stivei

Avem acces **doar** la elementul din vârful stivei!

- Există foarte multe aplicabilități pentru această structură de date.
  - Gestionarea memoriei unui program
  - Reținerea apelurilor funcțiilor
  - Executarea funcțiilor recursive
  - Inversarea unui șir
  - Verificarea parantezării
  - Evaluarea expresiilor aritmetice
  - Algoritmul de parcurgere în adâncime

Utilizarea stivei pentru apelurile funcțiilor

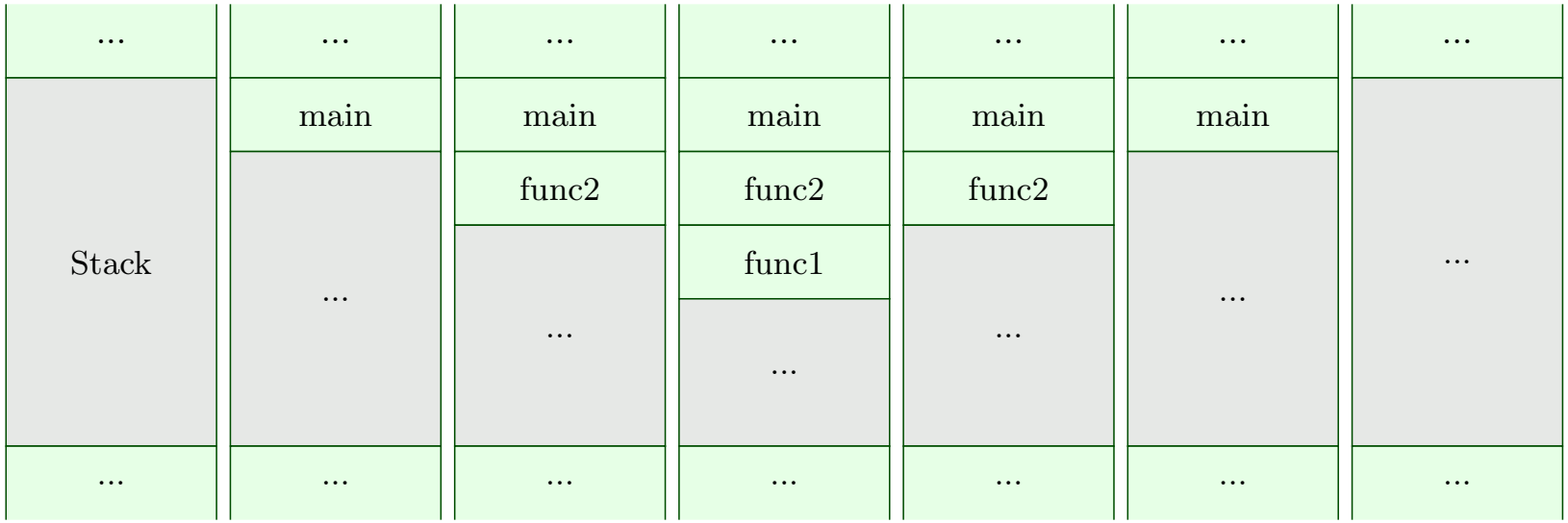
- Stiva este o regiune dinamică în cadrul unui proces, fiind gestionată automat de compilator.
- Stiva este folosită pentru a stoca *stack frame-uri*. Pentru fiecare apel de funcție se va crea un nou *stack frame*.
- Un *stack frame* conține:
  - variabile locale
  - argumentele funcției
  - adresa de retur

Important

Pe majoritatea arhitecturilor moderne stiva crește în jos (de la adrese mari la adrese mici) și heap-ul crește în sus.
Stiva crește la fiecare apel de funcție și scade la fiecare revenire din funcție.

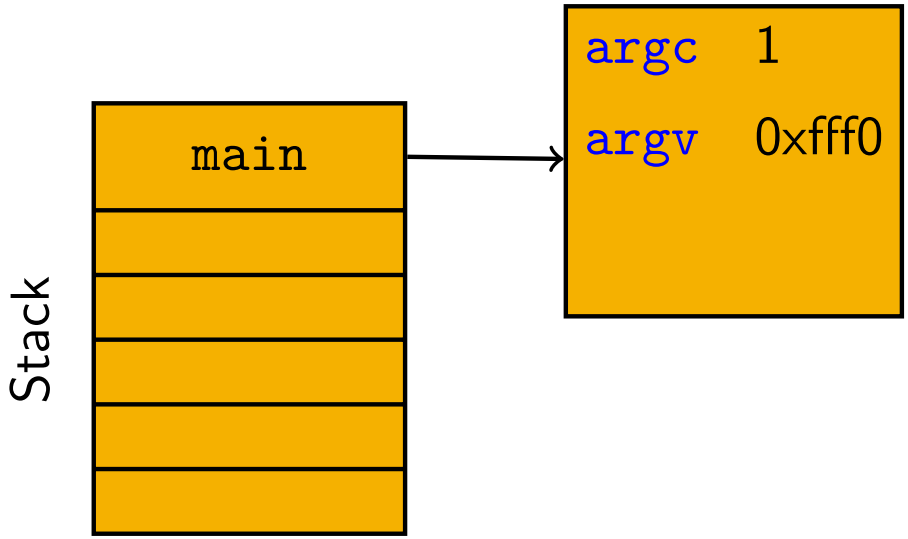
Stiva de apeluri – Exemplu

```
void func1() { }
void func2() {
    func1();
}
int main() {
    func2();
    return 0;
}
```



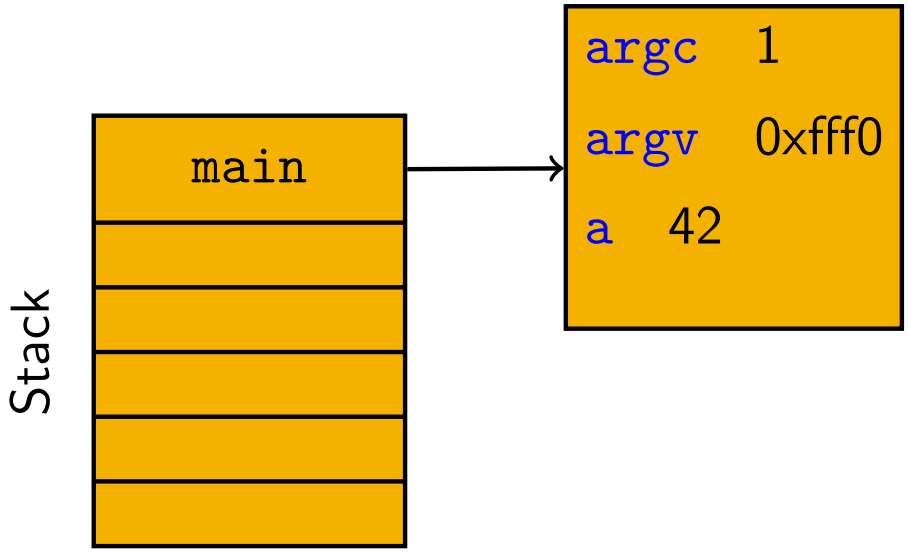
Stiva de apeluri – Exemplu detaliat

```
void func2() {
    int d = 0;
}
void func1() {
    int c = 99;
    func2();
}
int main(int argc, char *argv[])
↪   {
    int a = 42;
    int b = 17;
    func1();
    printf("Done!\n");
    return 0;
}
```



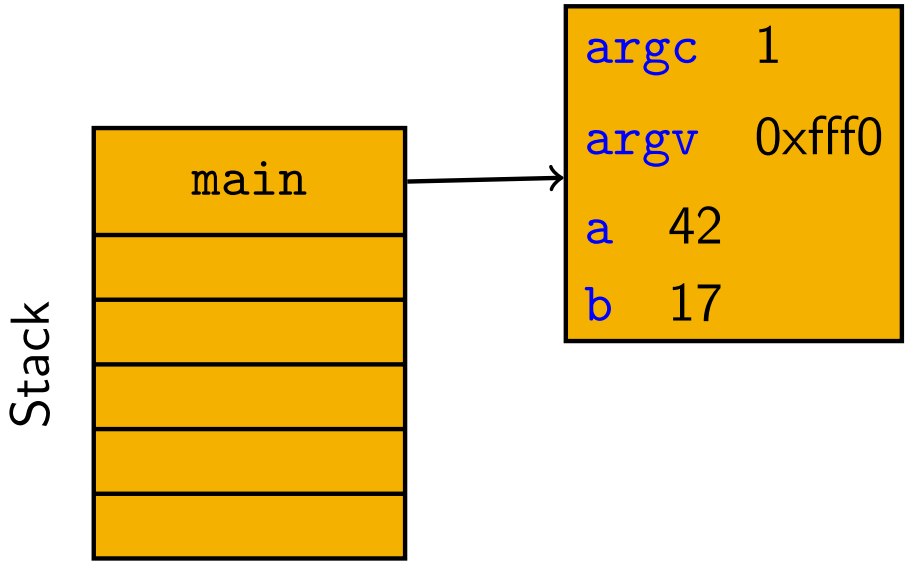
Stiva de apeluri – Exemplu detaliat

```c
void func2() {
    int d = 0;
}
void func1() {
    int c = 99;
    func2();
}
int main(int argc, char *argv[])
↪   {
    int a = 42;
    int b = 17;
    func1();
    printf("Done!\n");
    return 0;
}
```



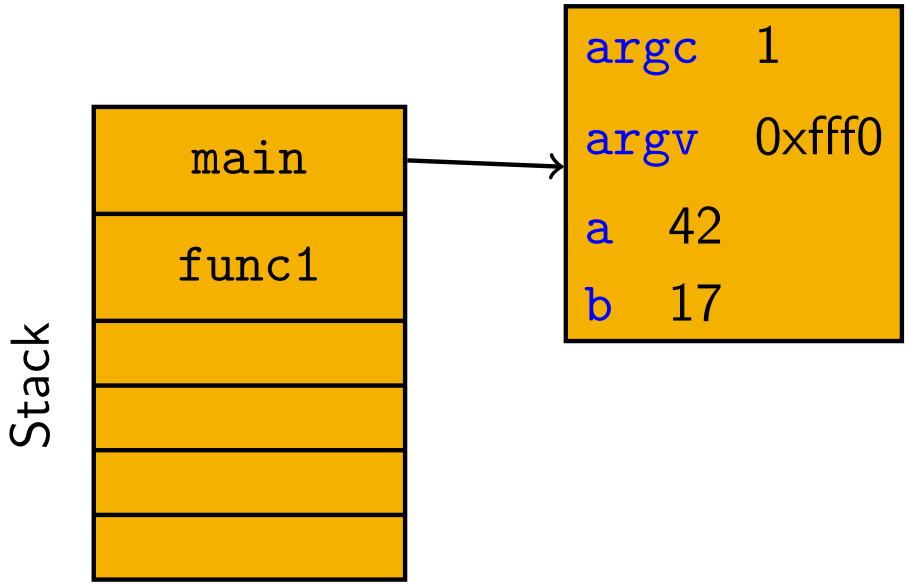
Stiva de apeluri – Exemplu detaliat

```c
void func2() {
    int d = 0;
}
void func1() {
    int c = 99;
    func2();
}
int main(int argc, char *argv[])
↪   {
    int a = 42;
    int b = 17;
    func1();
    printf("Done!\n");
    return 0;
}
```



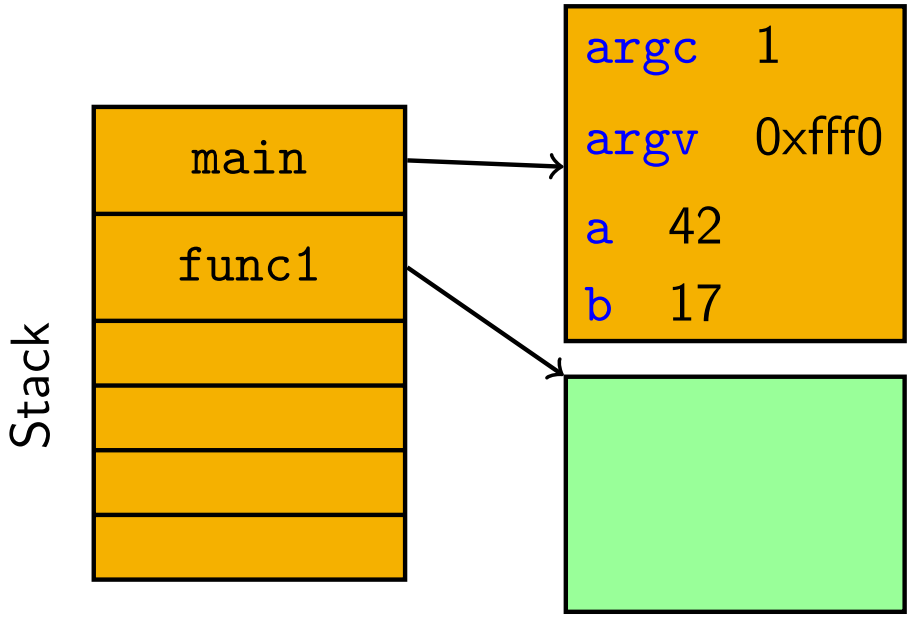
Stiva de apeluri – Exemplu detaliat

```c
void func2() {
    int d = 0;
}
void func1() {
    int c = 99;
    func2();
}
int main(int argc, char *argv[])
↪   {
    int a = 42;
    int b = 17;
    func1();
    printf("Done!\n");
    return 0;
}
```



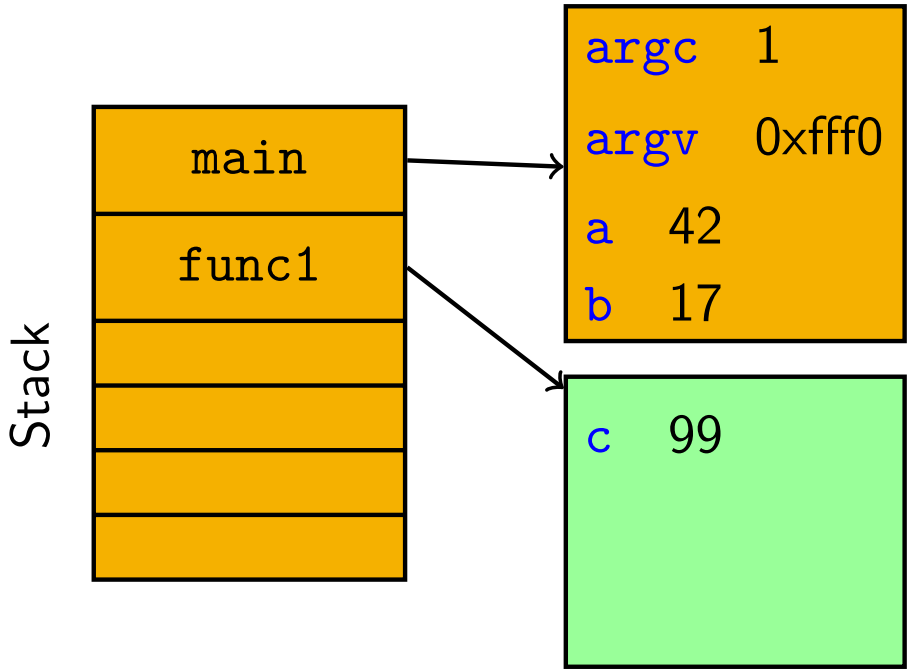
Stiva de apeluri – Exemplu detaliat

```c
void func2() {
    int d = 0;
}
void func1() {
    int c = 99;
    func2();
}
int main(int argc, char *argv[])
↪   {
    int a = 42;
    int b = 17;
    func1();
    printf("Done!\n");
    return 0;
}
```



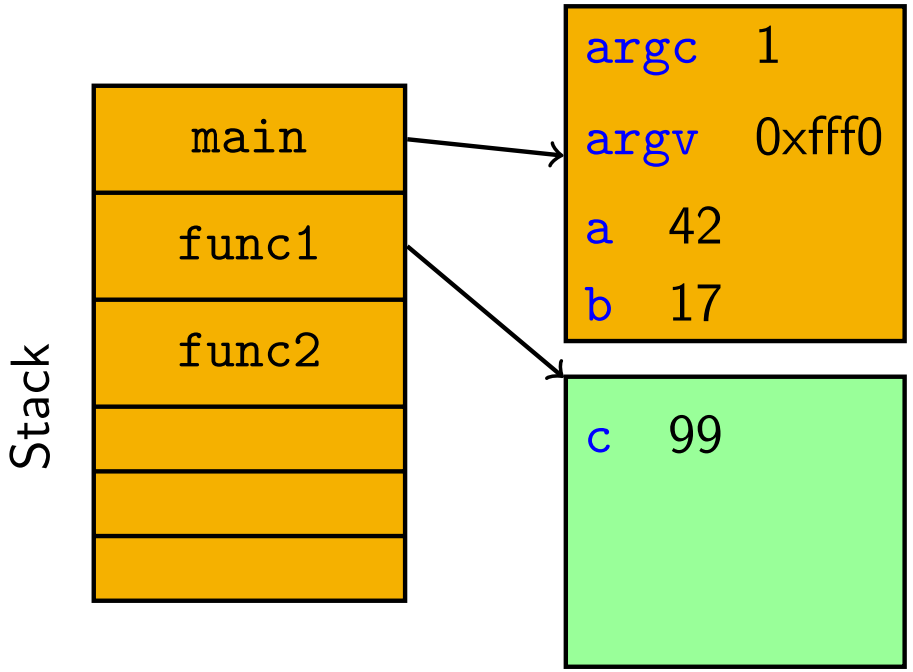
Stiva de apeluri – Exemplu detaliat

```
void func2() {
    int d = 0;
}
void func1() {
    int c = 99;
    func2();
}
int main(int argc, char *argv[])
↪   {
    int a = 42;
    int b = 17;
    func1();
    printf("Done!\n");
    return 0;
}
```



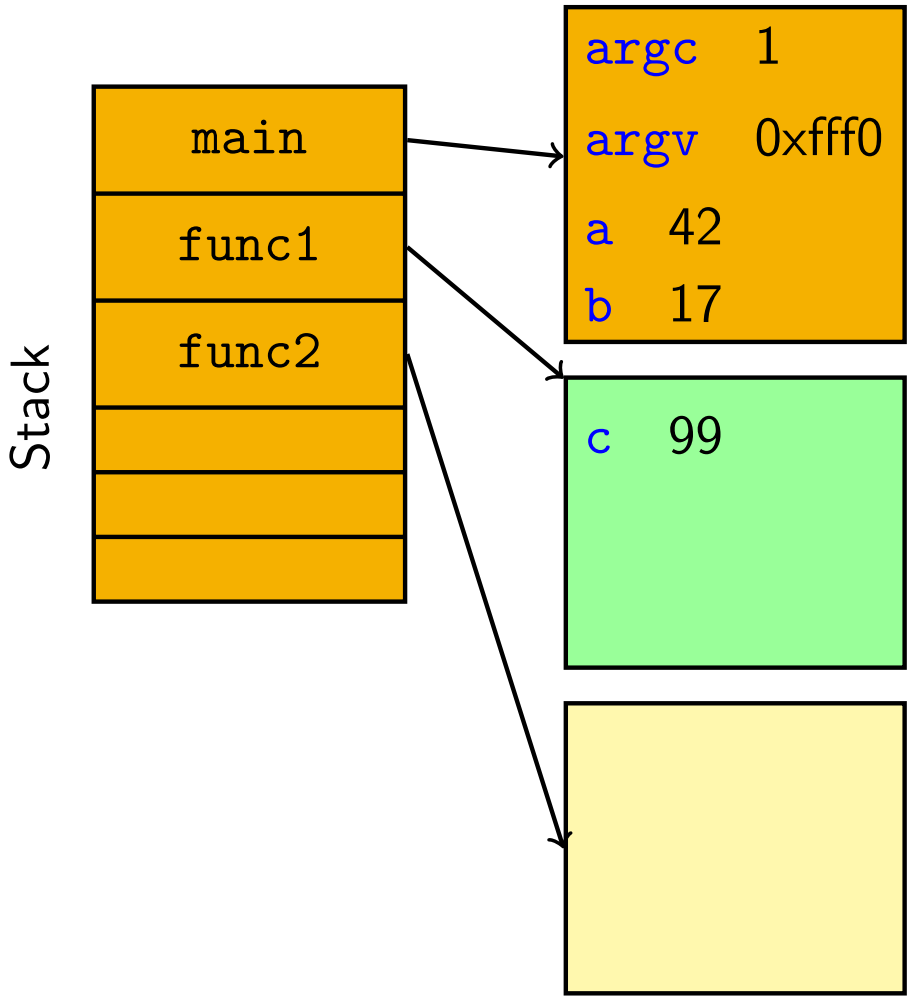
Stiva de apeluri – Exemplu detaliat

```
void func2() {
    int d = 0;
}
void func1() {
    int c = 99;
    func2();
}
int main(int argc, char *argv[])
↪   {
    int a = 42;
    int b = 17;
    func1();
    printf("Done!\n");
    return 0;
}
```



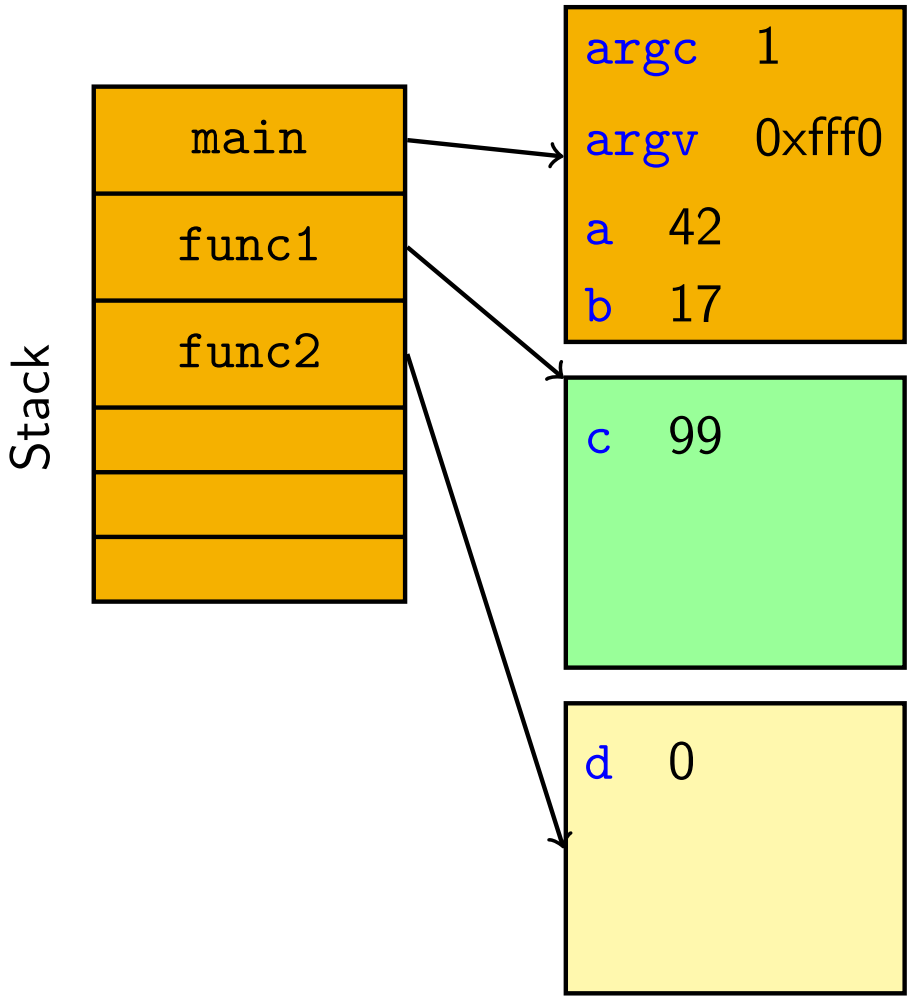
Stiva de apeluri – Exemplu detaliat

```
void func2() {
    int d = 0;
}
void func1() {
    int c = 99;
    func2();
}
int main(int argc, char *argv[])
 ↪  {
    int a = 42;
    int b = 17;
    func1();
    printf("Done!\n");
    return 0;
}
```



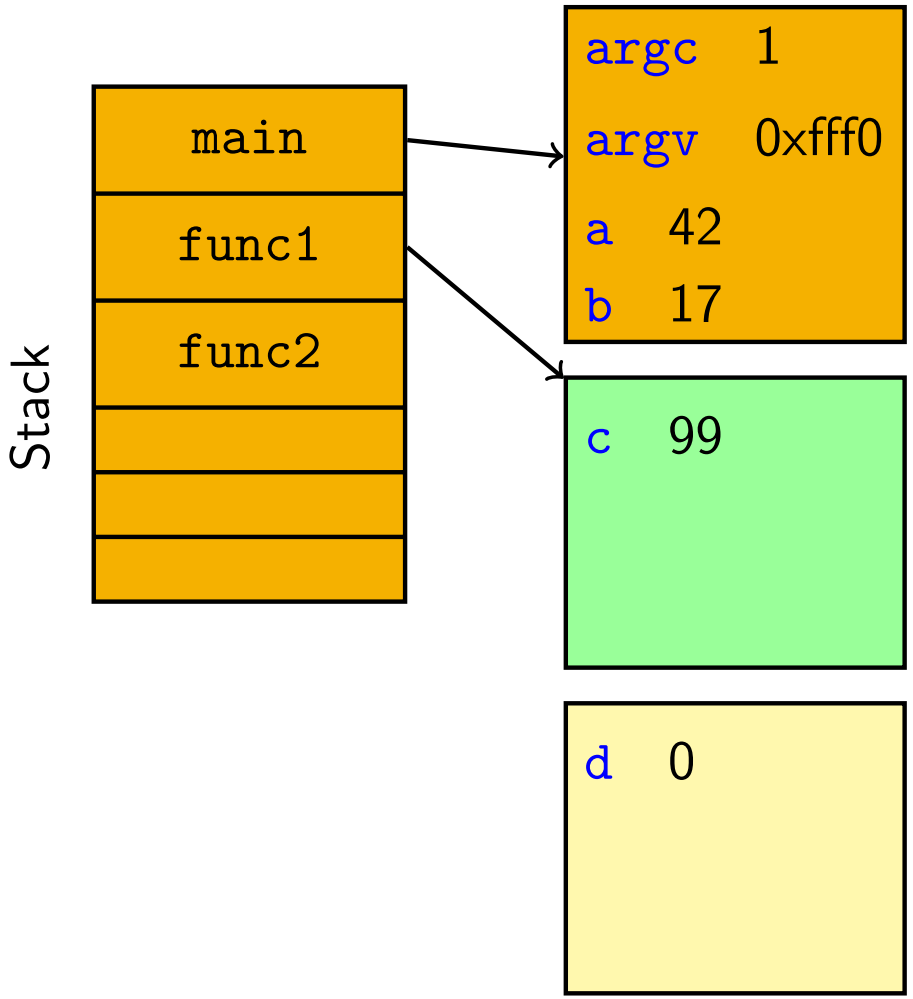
Stiva de apeluri – Exemplu detaliat

```c
void func2() {
    int d = 0;
}
void func1() {
    int c = 99;
    func2();
}
int main(int argc, char *argv[])
↪  {
    int a = 42;
    int b = 17;
    func1();
    printf("Done!\n");
    return 0;
}
```



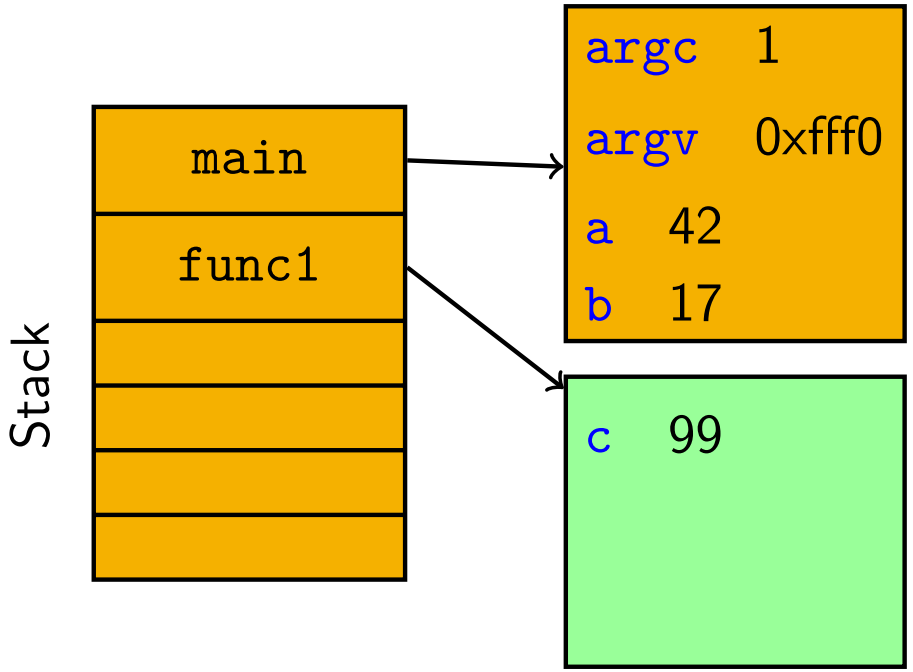
Stiva de apeluri – Exemplu detaliat

```c
void func2() {
    int d = 0;
}
void func1() {
    int c = 99;
    func2();
}
int main(int argc, char *argv[])
↪   {
    int a = 42;
    int b = 17;
    func1();
    printf("Done!\n");
    return 0;
}
```



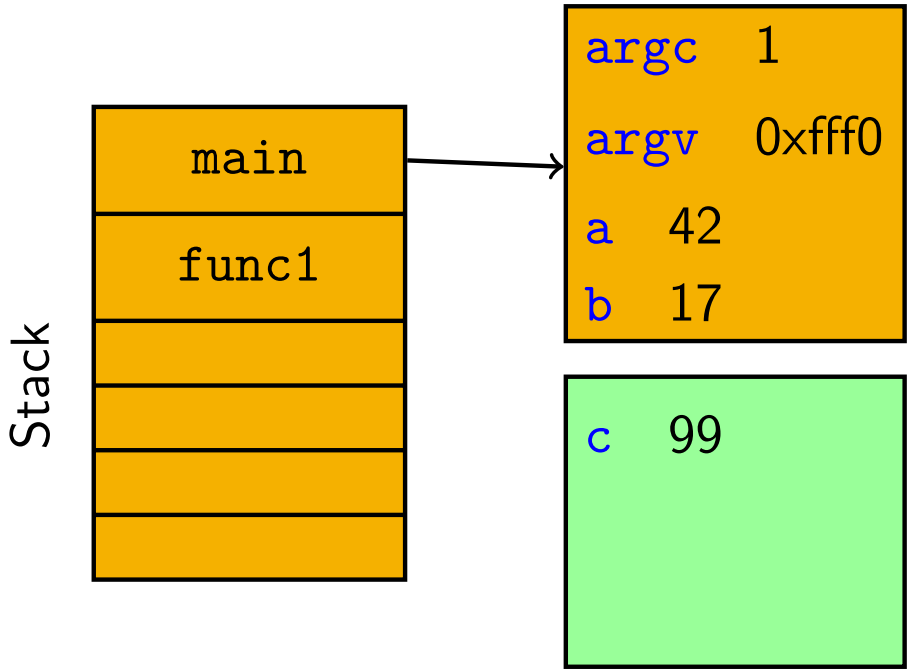
Stiva de apeluri – Exemplu detaliat

```
void func2() {
    int d = 0;
}
void func1() {
    int c = 99;
    func2();
}
int main(int argc, char *argv[])
↪   {
    int a = 42;
    int b = 17;
    func1();
    printf("Done!\n");
    return 0;
}
```



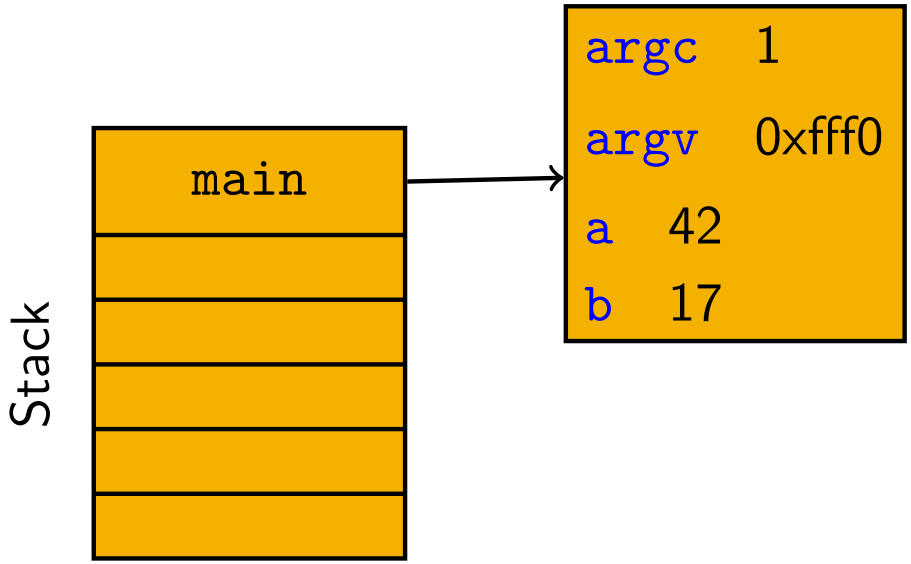
Stiva de apeluri – Exemplu detaliat

```c
void func2() {
    int d = 0;
}
void func1() {
    int c = 99;
    func2();
}
int main(int argc, char *argv[])
↪   {
    int a = 42;
    int b = 17;
    func1();
    printf("Done!\n");
    return 0;
}
```



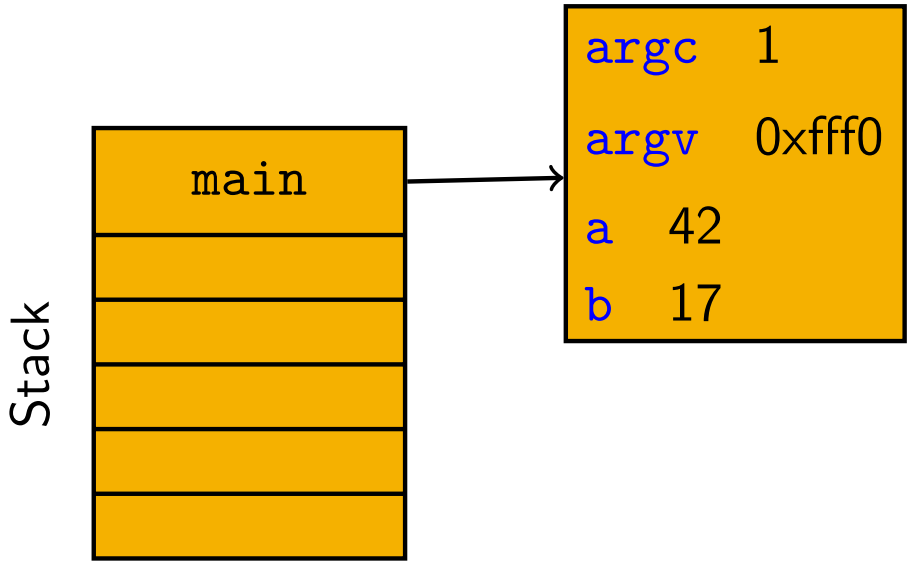
Stiva de apeluri – Exemplu detaliat

```c
void func2() {
    int d = 0;
}
void func1() {
    int c = 99;
    func2();
}
int main(int argc, char *argv[])
↪   {
    int a = 42;
    int b = 17;
    func1();
    printf("Done!\n");
    return 0;
}
```



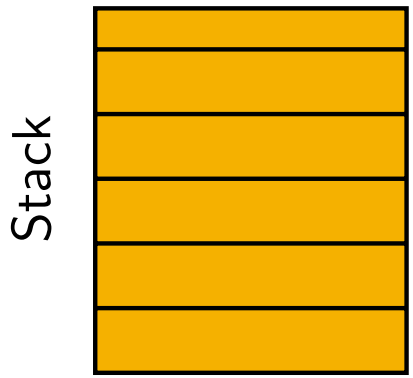
Stiva de apeluri – Exemplu detaliat

```c
void func2() {
    int d = 0;
}
void func1() {
    int c = 99;
    func2();
}
int main(int argc, char *argv[])
↪   {
    int a = 42;
    int b = 17;
    func1();
    printf("Done!\n");
    return 0;
}
```



Stiva de apeluri – Exemplu detaliat

```
void func2() {
    int d = 0;
}
void func1() {
    int c = 99;
    func2();
}
int main(int argc, char *argv[])
↪   {
    int a = 42;
    int b = 17;
    func1();
    printf("Done!\n");
    return 0;
}
```



Definirea TAD-ului pentru o stivă

Constructori

- Aceștia au ca rezultat o stivă nouă cu elemente de tip T
- Considerăm ca nume pentru TAD `TStack`
  1. Inițializarea stivei: `init:` $\rightarrow$ `TStack`
  2. Adăugarea unui element: `push:` $T \times$ `TStack` $\rightarrow$ `TStack`
  3. Eliminarea elementului din vârful stivei: `pop:TStack` $\rightarrow$ `TStack`

Funcții

- Operați care furnizează informații despre o stivă.
  1. Verificare stivă vidă: `isEmpty:TStack` $\rightarrow \{0, 1\}$
  2. Determinarea valorii pentru elementul din vârful stivei: `top:TStack` $\rightarrow$ `T`

Ce putem utiliza pentru a reține elementele unei stive?

Definirea TAD-ului pentru o stivă

Constructori

- Aceștia au ca rezultat o stivă nouă cu elemente de tip T
- Considerăm ca nume pentru TAD `TStack`
  1. Inițializarea stivei: $\texttt{init}: \rightarrow \texttt{TStack}$
  2. Adăugarea unui element: $\texttt{push}: T \times \texttt{TStack} \rightarrow \texttt{TStack}$
  3. Eliminarea elementului din vârful stivei: $\texttt{pop}: \texttt{TStack} \rightarrow \texttt{TStack}$

Funcții

- Operați care furnizează informații despre o stivă.
  1. Verificare stivă vidă: $\texttt{isEmpty}: \texttt{TStack} \rightarrow \{0, 1\}$
  2. Determinarea valorii pentru elementul din vârful stivei: $\texttt{top}: \texttt{TStack} \rightarrow \texttt{T}$

Ce putem utiliza pentru a reține elementele unei stive?

Vectori sau Liste

Implementarea stivei cu vector alocat dinamic

```
1  typedef int T;
2  typedef struct stack {
3      T *elements;
4      int top;
5      int maxSize;
6  } TStack;
```

Implementarea stivei cu vector alocat dinamic

```
1  typedef int T;
2  typedef struct stack {
3      T *elements;
4      int top;
5      int maxSize;
6  } TStack;
7  TStack init() {
8      TStack stack;
9      stack.elements = malloc(100 * sizeof(T));
10     stack.maxSize = 100;
11     stack.top = -1;
12     return stack;
13 }
```

Implementarea stivei cu vector alocat dinamic

```
1  typedef int T;
2  typedef struct stack {
3      T *elements;
4      int top;
5      int maxSize;
6  } TStack;
7  TStack init() {
8      TStack stack;
9      stack.elements = malloc(100 * sizeof(T));
10     stack.maxSize = 100;
11     stack.top = -1;
12     return stack;
13 }
```

Putem reprezenta stiva vidă din moment ce nu am considerat pointer la structură?

Implementarea stivei cu vector alocat dinamic

Ne folosim de câmpul `top` pentru a verifica dacă stiva e vidă.

Implementarea stivei cu vector alocat dinamic

Ne folosim de câmpul `top` pentru a verifica dacă stiva e vidă.

```
14  int isEmpty(TStack stack) {
15      return stack.top < 0;
16  }
```

Implementarea stivei cu vector alocat dinamic

Ne folosim de câmpul `top` pentru a verifica dacă stiva e vidă.

```
14  int isEmpty(TStack stack) {
15      return stack.top < 0;
16  }
```

Ce trebuie să facem pentru a adăuga un element în vârf?

Implementarea stivei cu vector alocat dinamic

Ne folosim de câmpul `top` pentru a verifica dacă stiva e vidă.

```
14  int isEmpty(TStack stack) {
15      return stack.top < 0;
16  }
```

Ce trebuie să facem pentru a adăuga un element în vârf?

Ne asigurăm că avem suficientă memorie pentru a reține noul element.

Implementarea stivei cu vector alocat dinamic

Dacă nu avem suficientă memorie, realocăm dinamic vectorul de elemente.

```
17  TStack push(TStack stack, T elem) {
18      if (stack.top == stack.maxSize - 1) {
19          stack.maxSize *= 2;
20          stack.elements = realloc(stack.elements,
    ↪   stack.maxSize * sizeof(T));
21      }
22      stack.elements[++stack.top] = elem;
23      return stack;
24  }
```

Implementarea stivei cu vector alocat dinamic

```
24  TStack pop(TStack stack) {
25      if (isEmpty(stack))
26          exit(1);
27      stack.top--;
28      return stack;
29  }
```

Implementarea stivei cu vector alocat dinamic

```
24  TStack pop(TStack stack) {
25      if (isEmpty(stack))
26          exit(1);
27      stack.top--;
28      return stack;
29  }
30  T top(TStack stack) {
31      if (isEmpty(stack))
32          exit(1);
33      return stack.elements[stack.top];
34  }
```

Implementarea stivei cu vector alocat dinamic

```
24  TStack pop(TStack stack) {
25      if (isEmpty(stack))
26          exit(1);
27      stack.top--;
28      return stack;
29  }
30  T top(TStack stack) {
31      if (isEmpty(stack))
32          exit(1);
33      return stack.elements[stack.top];
34  }
```

Cum putem dealoca memoria pentru această structură de date?

Implementarea stivei cu vector alocat dinamic

Am alocat memorie doar pentru vectorul de elemente.

Implementarea stivei cu vector alocat dinamic

Am alocat memorie doar pentru vectorul de elemente.

```
30  TStack freeStack(TStack stack) {
31      free(stack.elements);
32      stack.top = -1;
33      stack.maxSize = 0;
34      return stack;
35  }
```

Implementarea stivei cu vector alocat dinamic

Exemplu de utilizare

```
1  int main() {
2      TStack stack;
3      stack = init();
4      int i;
5      for (i = 0; i < 10; i++) {
6          stack = push(stack, i);
7      }
8      while (!isEmpty(stack)) {
9          printf("%d ", top(stack));
10         stack = pop(stack);
11     }
12     printf("\n");
13     stack = freeStack(stack);
14     return 0;
15 }
```

Implementarea stivei folosind listă

Modalitatea de reprezentare

```
1  typedef int T;
2  typedef struct stack {
3      T data;
4      struct stack *next;
5  }*TStack;
```

Există ceva diferențe, din perspectiva definiției structurii, între o stivă și o listă simplu înlănțuită?

```
6  TStack initStack(T data) {
7      TStack s = (TStack) malloc(sizeof(struct stack));
8      s->data = data;
9      s->next = NULL;
10     return s;
11 }
```

Implementarea stivei folosind listă

```
12  int isEmpty(TStack s) {
13      return s == NULL;
14  }
15  TStack push(TStack s, T data) {
16      TStack top;
17      if (isEmpty(s))
18          return initStack(data);
19      top = initStack(data);
20      top->next = s;
21      return top;
22  }
```

Cu ce operație de la liste seamănă operația push și ce complexitate are?

Implementarea stivei folosind listă

```
23  TStack pop(TStack s) {
24      TStack tmp;
25      if (isEmpty(s))
26          return s;
27      tmp = s;
28      s = s->next;
29      free(tmp);
30      return s;
31  }
32  T top(TStack s) {
33      if (isEmpty(s))
34          exit(1);
35      return s->data;
36  }
```

Implementarea stivei folosind listă

Cum putem dealoca memoria pentru o stivă reprezentată folosind liste?

Implementarea stivei folosind listă

Cum putem dealoca memoria pentru o stivă reprezentată folosind liste?

```
37  TStack freeStack(TStack s) {
38      while (!isEmpty(s))
39          s = pop(s);
40      return NULL;
41  }
```

Implementarea stivei folosind listă

Cum putem dealoca memoria pentru o stivă reprezentată folosind liste?

```
37  TStack freeStack(TStack s) {
38      while (!isEmpty(s))
39          s = pop(s);
40      return NULL;
41  }
```

Cum putem implementa constructorul `init` care nu primește niciun argument și întoarce stiva vidă?

Implementarea stivei folosind listă

Cum putem dealoca memoria pentru o stivă reprezentată folosind liste?

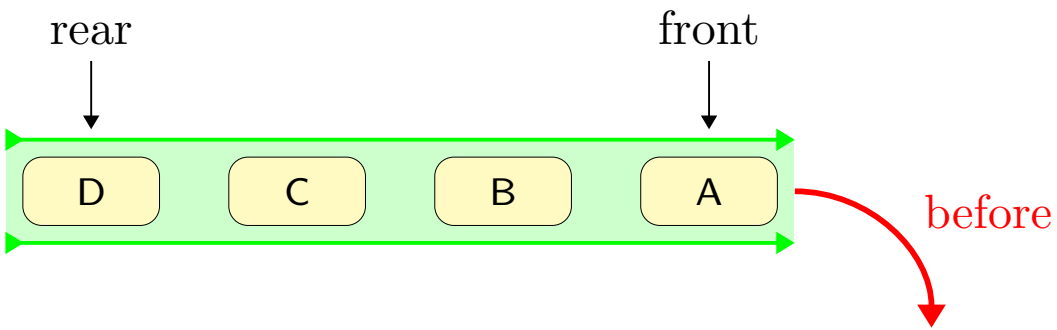
```
37  TStack freeStack(TStack s) {
38      while (!isEmpty(s))
39          s = pop(s);
40      return NULL;
41  }
```

Cum putem implementa constructorul `init` care nu primește niciun argument și întoarce stiva vidă?

```
42  TStack init() {
43      return NULL;
44  }
```

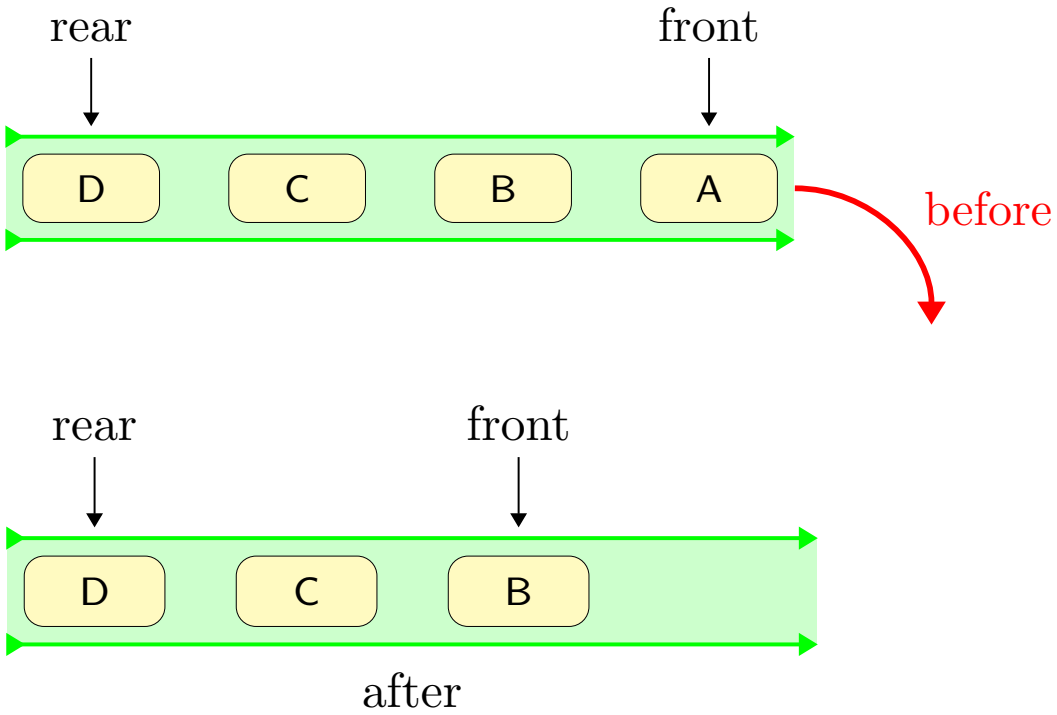
Coada – Motivație

- O structură de date care funcționează după principiul **FIFO** (**F**irst **I**n **F**irst **O**ut)
- Este o structură de date pentru care putem realiza următoarele operații:
  1. Eliminarea unui element de la începutul cozii



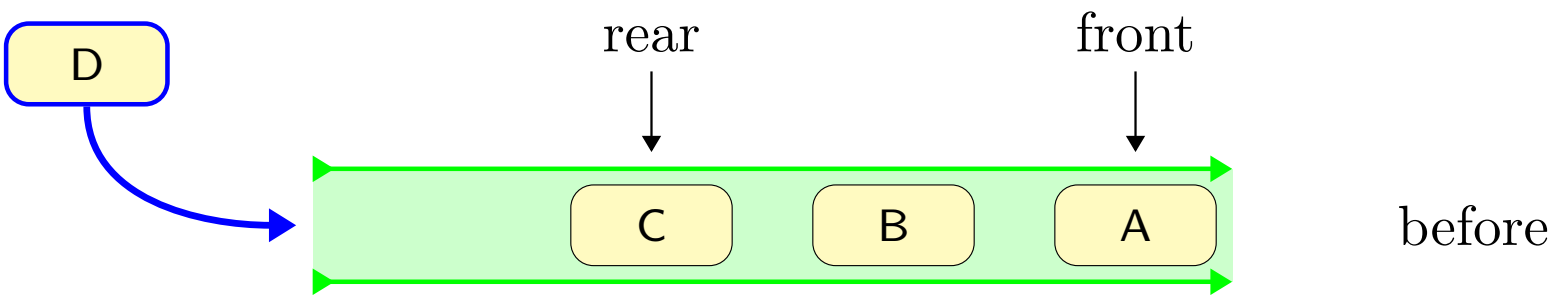
Coada – Motivație

- O structură de date care funcționează după principiul **FIFO** (**F**irst **I**n **F**irst **O**ut)
- Este o structură de date pentru care putem realiza următoarele operații:
  1. Eliminarea unui element de la începutul cozii



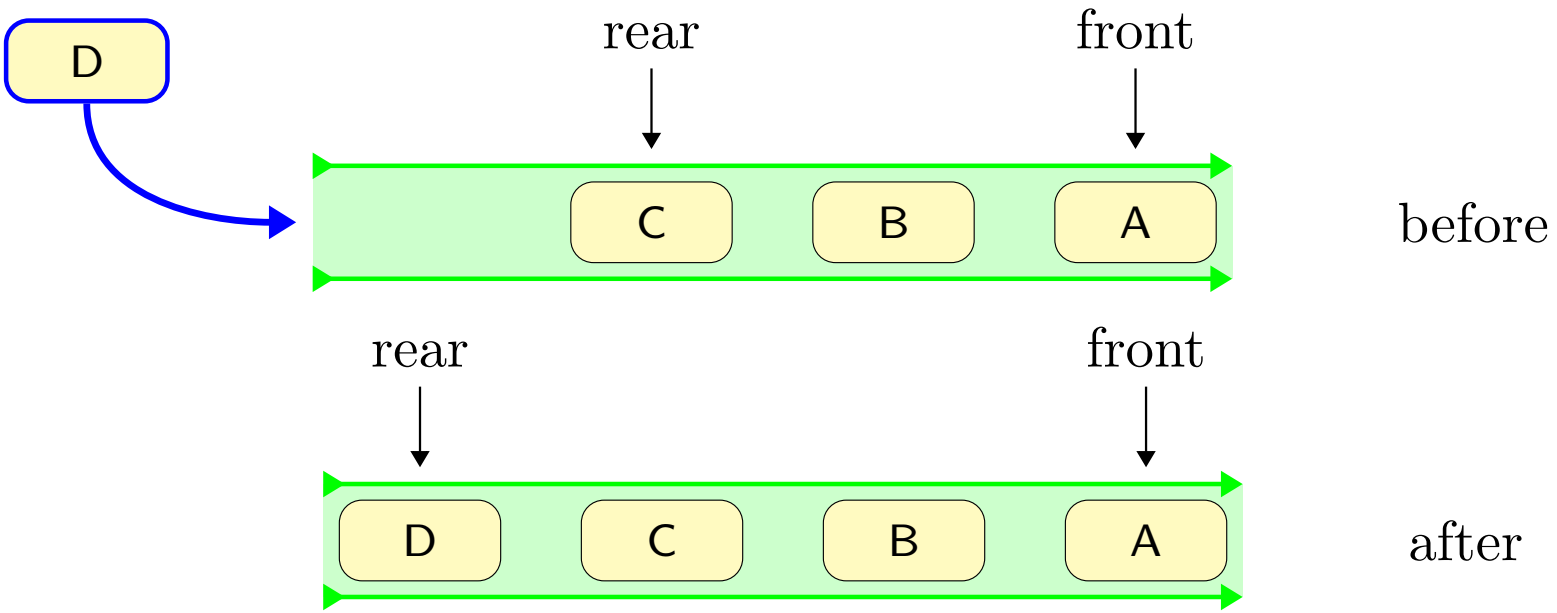
Coada – Motivație

- O structură de date care funcționează după principiul **FIFO** (**F**irst **I**n **F**irst **O**ut)
- Este o structură de date pentru care putem realiza următoarele operații:
  1. Eliminarea unui element de la începutul cozii
  2. Adăugarea unui element la finalul cozii



Coada – Motivație

- O structură de date care funcționează după principiul **FIFO** (**F**irst **I**n **F**irst **O**ut)
- Este o structură de date pentru care putem realiza următoarele operații:
  1. Eliminarea unui element de la începutul cozii
  2. Adăugarea unui element la finalul cozii



Coada – Motivație

- O structură de date care funcționează după principiul **FIFO** (**F**irst **I**n **F**irst **O**ut)
- Este o structură de date pentru care putem realiza următoarele operații:
  1. Eliminarea unui element de la începutul cozii
  2. Adăugarea unui element la finalul cozii

Avem acces **doar** la primul element introdus în coadă!

- Există foarte multe aplicabilități pentru această structură de date.
  - Programarea sarcinilor de lucru (planificare)
  - Alocarea resurselor
  - Batch Processing
  - Message Buffering
  - Algoritmul de parcurgere în lățime

Definirea TAD-ului pentru o coadă

Constructori

- Aceștia au ca rezultat o coadă nouă cu elemente de tip T
- Considerăm ca nume pentru TAD `TQueue`
  1. Inițializarea cozii: `init:` $\rightarrow$ `TQueue`
  2. Adăugarea unui element la finalul cozii: `enqueue:TQueue`$\rightarrow$`TQueue`
  3. Eliminarea elementului de la începutul cozii: `dequeue:TQueue`$\rightarrow$`TQueue`

Funcții

- Operați care furnizează informații despre o coadă.
  1. Verificare coadă vidă: `isEmpty:TQueue`$\rightarrow \{0, 1\}$
  2. Determinarea valorii pentru element din coadă: `front:TQueue`$\rightarrow$`T`

Ce putem utiliza pentru a reține elementele unei cozi?

Definirea TAD-ului pentru o coadă

Constructori

- Aceștia au ca rezultat o coadă nouă cu elemente de tip T
- Considerăm ca nume pentru TAD `TQueue`
  1. Inițializarea cozii: `init:` $\rightarrow$ `TQueue`
  2. Adăugarea unui element la finalul cozii: `enqueue:TQueue`$\rightarrow$`TQueue`
  3. Eliminarea elementului de la începutul cozii: `dequeue:TQueue`$\rightarrow$`TQueue`

Funcții

- Operați care furnizează informații despre o coadă.
  1. Verificare coadă vidă: `isEmpty:TQueue`$\rightarrow \{0, 1\}$
  2. Determinarea valorii pentru element din coadă: `front:TQueue`$\rightarrow$`T`

Ce putem utiliza pentru a reține elementele unei cozi?

Vectori sau Liste

Implementarea cozii cu vector alocat dinamic

```
1  typedef int T;
2  typedef struct queue {
3      T *elements;
4      int front; // capul cozii
5      int rear; // finalul cozii
6      int count; // nr de elemente din coadă
7      int maxSize;
8  } TQueue;
9  TQueue init() {
10     TQueue queue;
11     queue.elements = malloc(100 * sizeof(T));
12     queue.maxSize = 100;
13     queue.front = 0;
14     queue.rear = 0;
15     queue.count = 0;
16     return queue;
17 }
```

Implementarea cozii cu vector alocat dinamic

```
18  int isEmpty(TQueue queue) {
19      return queue.count == 0;
20  }
21  TQueue enqueue(TQueue queue, T elem) {
22      if (queue.maxSize == queue.count) {
23          queue.maxSize *= 2;
24          queue.elements = realloc(queue.elements,
    ↪   queue.maxSize * sizeof(T));
25      }
26      queue.elements[queue.rear++] = elem;
27      queue.count++;
28      return queue;
29  }
```

Implementarea cozii cu vector alocat dinamic

```
18  int isEmpty(TQueue queue) {
19      return queue.count == 0;
20  }
21  TQueue enqueue(TQueue queue, T elem) {
22      if (queue.maxSize == queue.count) {
23          queue.maxSize *= 2;
24          queue.elements = realloc(queue.elements,
    ↪   queue.maxSize * sizeof(T));
25      }
26      queue.elements[queue.rear++] = elem;
27      queue.count++;
28      return queue;
29  }
```

Ce complexitate are operația enqueue?

Implementarea cozii cu vector alocat dinamic

```
30  TQueue dequeue(TQueue queue) {
31      if (isEmpty(queue))
32          exit(1);
33      queue.front++;
34      queue.count--;
35      return queue;
36  }
37  T front(TQueue queue) {
38      if (isEmpty(queue))
39          exit(1);
40      return queue.elements[queue.front];
41  }
```

Implementarea cozii cu vector alocat dinamic

```
30  TQueue dequeue(TQueue queue) {
31      if (isEmpty(queue))
32          exit(1);
33      queue.front++;
34      queue.count--;
35      return queue;
36  }
37  T front(TQueue queue) {
38      if (isEmpty(queue))
39          exit(1);
40      return queue.elements[queue.front];
41  }
```

Ce complexitate are operația dequeue?

Implementarea cozii cu vector alocat dinamic

Cum putem dealoca memoria pentru această structură de date?

Implementarea cozii cu vector alocat dinamic

Cum putem dealoca memoria pentru această structură de date?

Am alocat memorie doar pentru vectorul de elemente.

Implementarea cozii cu vector alocat dinamic

Cum putem dealoca memoria pentru această structură de date?

Am alocat memorie doar pentru vectorul de elemente.

```
42  TQueue freeQueue(TQueue queue) {
43      free(queue.elements);
44      queue.front = 0;
45      queue.count = 0;
46      queue.maxSize = 0;
47      return queue;
48  }
```

Implementarea cozii cu vector alocat dinamic

Exemplu de utilizare

```
1  int main() {
2      TQueue queue = init();
3      int i;
4      for (i = 0; i < 10; i++)
5          queue = enqueue(queue, i);
6      for (i = 0; i < 5; i++) {
7          printf("%d ", front(queue));
8          queue = dequeue(queue);
9      }
10     printf("\n");
11     for (i = 20; i < 25; i++)
12         queue = enqueue(queue, i);
13     while (!isEmpty(queue)) {
14         printf("%d ", front(queue));
15         queue = dequeue(queue);
16     }
```

Implementarea cozii cu vector alocat dinamic

```
17      printf("\n");
18      queue = freeQueue(queue);
19      return 0;
20  }
```

Implementarea cozii cu vector alocat dinamic

```
17      printf("\n");
18      queue = freeQueue(queue);
19      return 0;
20  }
```

Ce o să afișeze acest exemplu?

Implementarea cozii cu vector alocat dinamic

```
17      printf("\n");
18      queue = freeQueue(queue);
19      return 0;
20  }
```

Ce o să afișeze acest exemplu?

```
0 1 2 3 4
5 6 7 8 9 20 21 22 23 24
```

Există ceva probleme cu această implementare?!

Implementarea cozii folosind listă

Modalitatea de reprezentare

```
1  typedef int T;
2  typedef struct node {
3      T data;
4      struct node *next;
5  } Node;
6  typedef struct queue {
7      Node *head, *tail;
8      int size;
9  } *TQueue;
```

De ce avem nevoie să reținem și un pointer către ultimul nod (`tail`)?

Implementarea cozii folosind listă

```
10  Node *initNode(T data) {
11      Node *node = malloc(sizeof(struct node));
12      node->data = data;
13      node->next = NULL;
14      return node;
15  }
16  Node *freeNode(Node *node) {
17      if (node)
18          free(node);
19      return NULL;
20  }
21  TQueue initQueue(T data) {
22      TQueue queue = malloc(sizeof(struct queue));
23      queue->head = queue->tail = initNode(data);
24      queue->size = 1;
25      return queue;
26  }
```

Implementarea cozii folosind listă

```
27  TQueue init() {
28      return NULL;
29  }
30  int isEmpty(TQueue queue) {
31      return queue == NULL || queue->head == NULL ||
    ↪   queue->size == 0;
32  }
```

Implementarea cozii folosind listă

```
27  TQueue init() {
28      return NULL;
29  }
30  int isEmpty(TQueue queue) {
31      return queue == NULL || queue->head == NULL ||
    ↪   queue->size == 0;
32  }
```

De ce nu am avut nevoie să folosim listă dublu înlănțuită? Putem actualiza pointerul `tail` dacă nu avem listă dublu înlănțuită?

Implementarea cozii folosind listă

```
27  TQueue init() {
28      return NULL;
29  }
30  int isEmpty(TQueue queue) {
31      return queue == NULL || queue->head == NULL ||
    ↪   queue->size == 0;
32  }
```

De ce nu am avut nevoie să folosim listă dublu înlănțuită? Putem actualiza pointerul `tail` dacă nu avem listă dublu înlănțuită?

Este în regulă, deoarece realizăm doar inserare la final.

Implementarea cozii folosind listă

```
47  TQueue enqueue(TQueue queue, T data) {
48      Node *node;
49      if (isEmpty(queue)) {
50          if (queue == NULL)
51              return initQueue(data);
52          free(queue);
53          return initQueue(data);
54      }
55      node = initNode(data);
56      queue->tail->next = node;
57      queue->tail = node;
58      queue->size++;
59      return queue;
60  }
```

Implementarea cozii folosind listă

```
33  TQueue dequeue(TQueue queue) {
34      Node *tmp;
35      if (!isEmpty(queue)) {
36          tmp = queue->head;
37          queue->head = queue->head->next;
38          tmp = freeNode(tmp);
39          queue->size--;
40      }
41      return queue;
42  }
```

Implementarea cozii folosind listă

```
33  TQueue dequeue(TQueue queue) {
34      Node *tmp;
35      if (!isEmpty(queue)) {
36          tmp = queue->head;
37          queue->head = queue->head->next;
38          tmp = freeNode(tmp);
39          queue->size--;
40      }
41      return queue;
42  }
```

Ce complexitate au operațiile enqueue și dequeue?

Implementarea cozii folosind listă

```
43  T front(TQueue queue) {
44      if (!isEmpty(queue))
45          return queue->head->data;
46      else
47          exit(1);
48  }
49  TQueue freeQueue(TQueue queue) {
50      while (!isEmpty(queue))
51          queue = dequeue(queue);
52      if (queue)
53          free(queue);
54      return NULL;
55  }
```

Probleme tip interviu

Enunț: Dându-se o expresie, care poate conține paranteze deschise și închise, implementați o funcție care verifică dacă expresia conține sau nu paranteze inutile. Alegeți o structură de date care să faciliteze rezolvarea problemei și motivați alegerea făcută.

Exemple:

`(x + y) * z` – Nu conține

`(x + y) * (z)` – Conține

`((x + y)) * (2 + 3)` – Conține

`(x + y) / (1 + 2)` – Nu conține

`((x + y) / (1 + 2))` – Nu conține (*caz particular*)

Probleme tip interviu

Enunț: Dându-se o expresie, care poate conține paranteze deschise și închise, implementați o funcție care verifică dacă expresia conține sau nu paranteze inutile. Alegeți o structură de date care să faciliteze rezolvarea problemei și motivați alegerea făcută.

Exemple:

`(x + y) * z` – Nu conține

`(x + y) * (z)` – Conține

`((x + y)) * (2 + 3)` – Conține

`(x + y) / (1 + 2)` – Nu conține

`((x + y) / (1 + 2))` – Nu conține (*caz particular*)

Ce structură de date putem folosi pentru rezolvarea acestei probleme?

Probleme tip interviu

Ne folosim de o stivă!

Probleme tip interviu

Ne folosim de o stivă!

Ce vom adăuga în stivă și când eliminăm din stivă?

Adăugăm în stivă toate simbolurile diferite de).

Probleme tip interviu

Ne folosim de o stivă!

Ce vom adăuga în stivă și când eliminăm din stivă?

Adăugăm în stivă toate simbolurile diferite de).
Când întâlnim) începem să scoatem din stivă până găsim (în vârful stivei și verificăm dacă întâlnim cel puțin o operație.

Probleme tip interviu

```
1  int checkRedundancy(char *str) {
2      Stack st = NULL;
3      char ch;
4      for (i = 0; i < strlen(str); i++) {
5          ch = str[i];
6          if (ch == ')') {
7              char t_ch = top(st);
8              st = pop(st);
9              int flag = 1;
10             while (t_ch != '(') {
11                 if (t_ch == '+' || t_ch == '-' ||
12                     t_ch == '*' || t_ch == '/')
13                     flag = 0;
14                 t_ch = top(st);
15                 st = pop(st);
16             }
```

Probleme tip interviu

```
17              if (flag == 1)
18                  return 1;
19          }
20          else
21              st = push(st, ch);
22      }
23      return 0;
24  }
```

Probleme de tip interviu

Enunț: Faimoasa echipă de fotbal *Liverpool* a decis să organizeze un eveniment, numit **PASS** and **BACK**, prin care să promoveze sportul. La acest eveniment vor participa N persoane, primind fiecare un id cuprins între 1 și 1.000.000. Inițial, mingea este în posesia jucătorului cu id-ul K. La fiecare pas, jucătorul care deține mingea poate alege să o paseze mai departe către un alt jucător sau să i-o paseze înapoi celui de la care a primit-o. Scrieți o funcție care să returneze id-ul jucătorului care deține mingea după M pase. Alegeți o structură de date care să faciliteze rezolvarea problemei și motivați alegerea făcută.

```
typedef struct move {
        char type;
        int next;
} Move;
int getPlayer(Move *moves, int M, int *ids, int N, int init);
```

Exemplu: **initialPlayer** $= 23$ / **moves** $=$ **P** 86; **P** 63; **P** 60; **B**; **P** 47; **B**
Output: 63

Probleme de tip interviu

```
17  int getPlayer(Move *moves, int M, int *ids, int N, int ini
18      Stack s = NULL;
19      int i = 0;
20      s = push(s, initialPlayer);
21      for (i = 0; i < M; i++) {
22          if (moves[i].type == 'P') {
23              s = push(s, moves[i].next);
24          } else {
25              s = pop(s);
26          }
27      }
28      return top(s);
29  }
```

Vă mulțumesc pentru atenție!