STU FEI

Programovacie techniky

2. Nedefinované správanie, typová bezpečnosť, menné priestory, organizácia pamäte, C++ zásobník, zásobník, fronta, zreťazený zoznam

Hello world!

Hello world v C

```
#include <stdio.h>

int main() {

    printf("%s","Hello world!");

    return 0;
}
```

Jediná funkcia printf()

Hello world v C++

```
#include <iostream>

int main() {

    std::cout << "Hello world!";

    return 0;
}
```

C++ knižnice bez .h

Menný priestor

```
#include <iostream>

int main() {

  std::cout << "Hello, World!";

  return 0;
}
```

Menný priestor

```
#include <cstdio> //stdio.h v std::


namespace NS {
  int printf(const char * format, …) {}; //moja funkcia
}
int main() {
  std::printf("%s", "Hello, World!"); //z std::


  return 0;
}
```

Menný priestor

```
#include <iostream>
using namespace std;

int main() {

  cout << "Hello, World!";

  return 0;
}
```

Menný priestor

```
#include <iostream>
using std::cout;

int main() {

  cout << "Hello, World!";

  return 0;
}
```

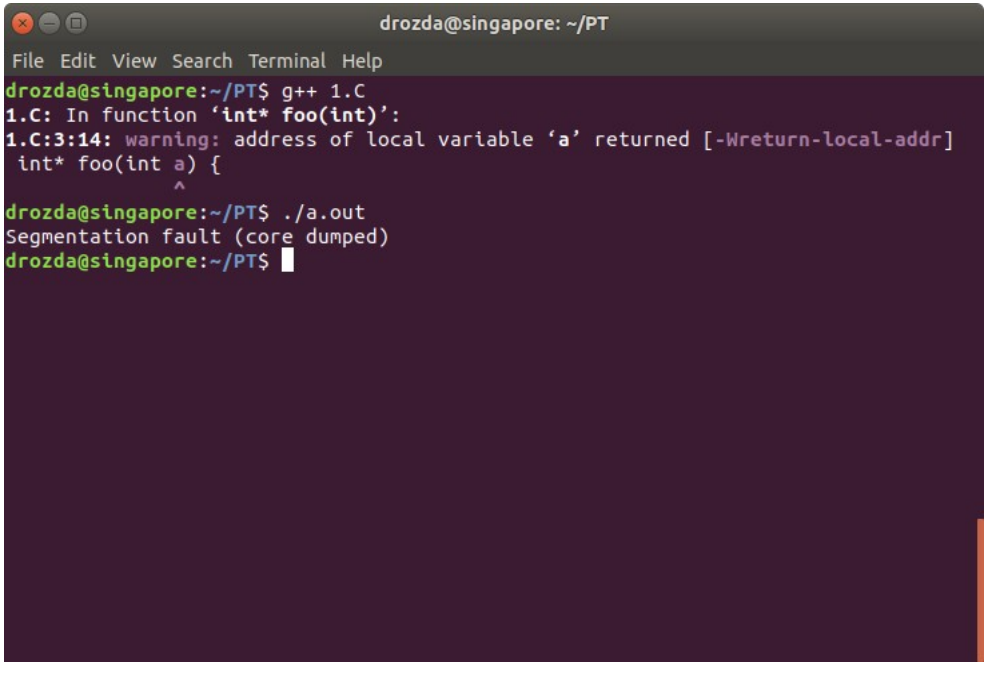
Menný priestor

```
namespace N1 { int x; }
namespace N2 { int x; }
int globalna_premenna;

int main() {
   int n = 4;

   N1::x = 1;
   N2::x = 2;
   ::globalna_premenna = 5;
}
```

Nedefinované správanie

```
#include <iostream>
int* foo(int a) {
  return &a;
}

int main() {
  std::cout << *foo(3);
  return 0;
}
```

Lokálna premenná zanikne



Nedefinované správanie

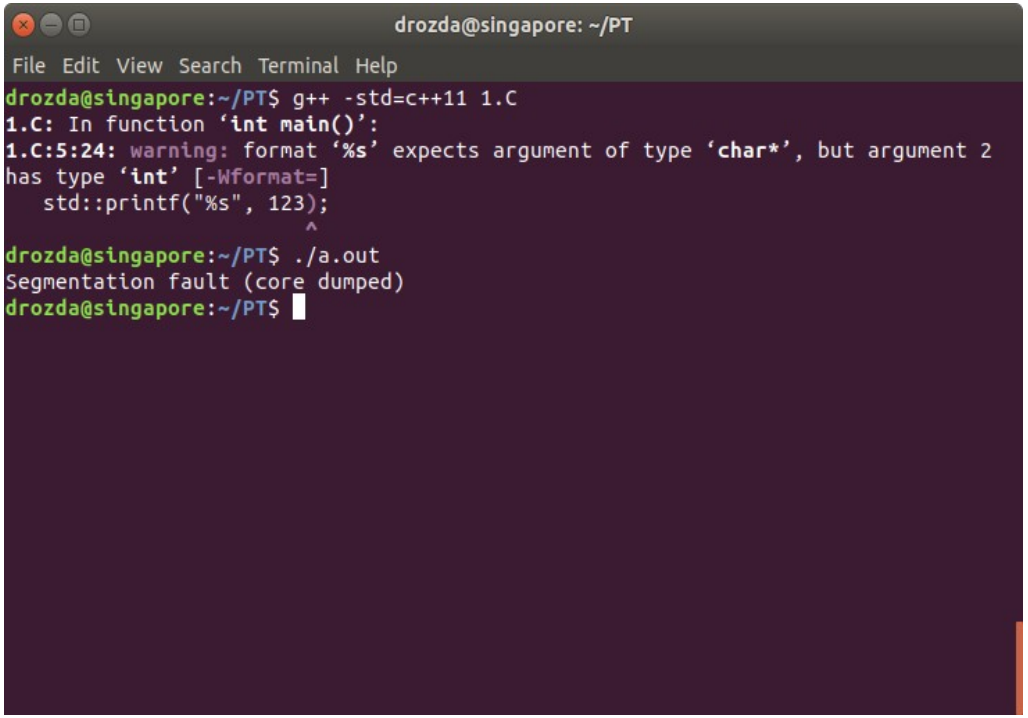
Nedefinované správanie sa udeje, keď program spraví niečo, čoho výsledok nie je definovaný štandardom (môže nastať čokoľvek)

Príklady:

- Dereferencovanie smerníka na objekt, ktorého životnosť skončila
- Pretečenie (signed) int

Typová bezpečnosť

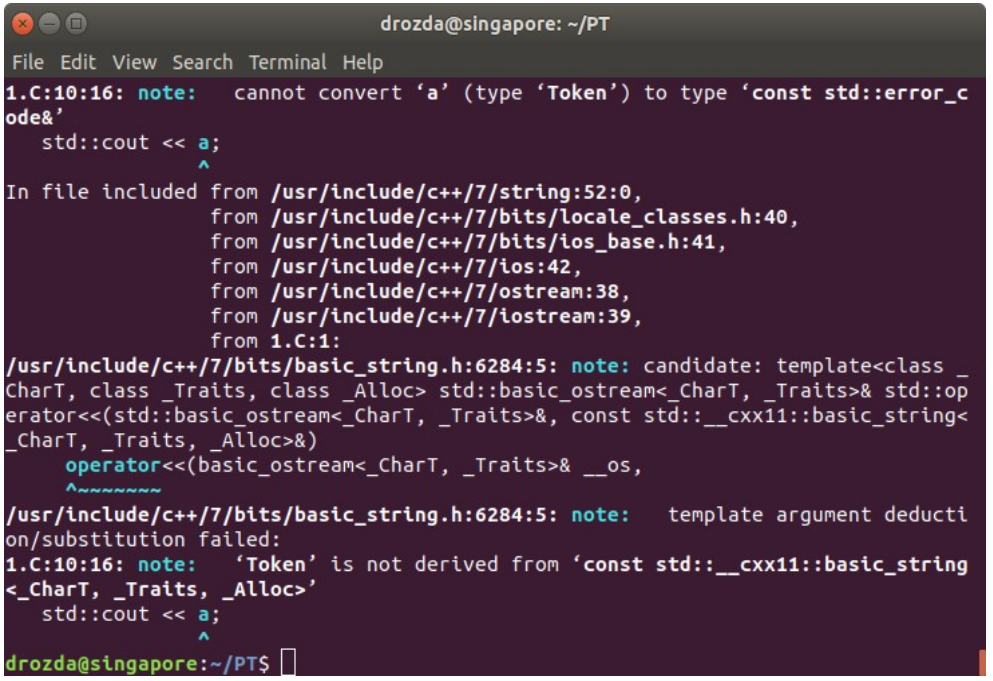
```
#include <cstdio>
int main() {
  std::printf("%s", 123);
  return 0;
}
```



Typová bezpečnosť

Typová bezpečnosť znamená, že kompilátor vie zistiť, či je použitý správny typ.

```
#include <iostream>

struct Token {
  int a = 0;
};

int main() {
  Token a;
  std::cout << a;
  return 0;
}
```



Životnosť premennej, objektu

```
int main() {
  int a = 3;
  return 0;
}
```

Dočasný objekt (rvalue) 3 zanikne

Lokálna premenná a zanikne

Životnosť premennej, objektu

```
int main() {
  int&& a = 3;


  return 0;
}
```

Lokálna premenná a zanikne,
dočasný objekt (rvalue) 3 zanikne

Životnosť premennej, objektu

```
#include <iostream>
using std::cout;

int main() {
  int&& a = 3;

  std::cout << &a; //0x7ffcad28f4ec
  return 0;
}
```

Rozdelenie len na l-value a r-value je od C++11 nedostatočné.

A čo ak vieme, pretypovať l-value na r-value??

C++ pamäť

Premenné s automatickou životnosťou, lokálne premenné (C++ zásobník)

Globálne a statické premenné

Dynamicky vytvorené objekty v heape (malloc, new, …)

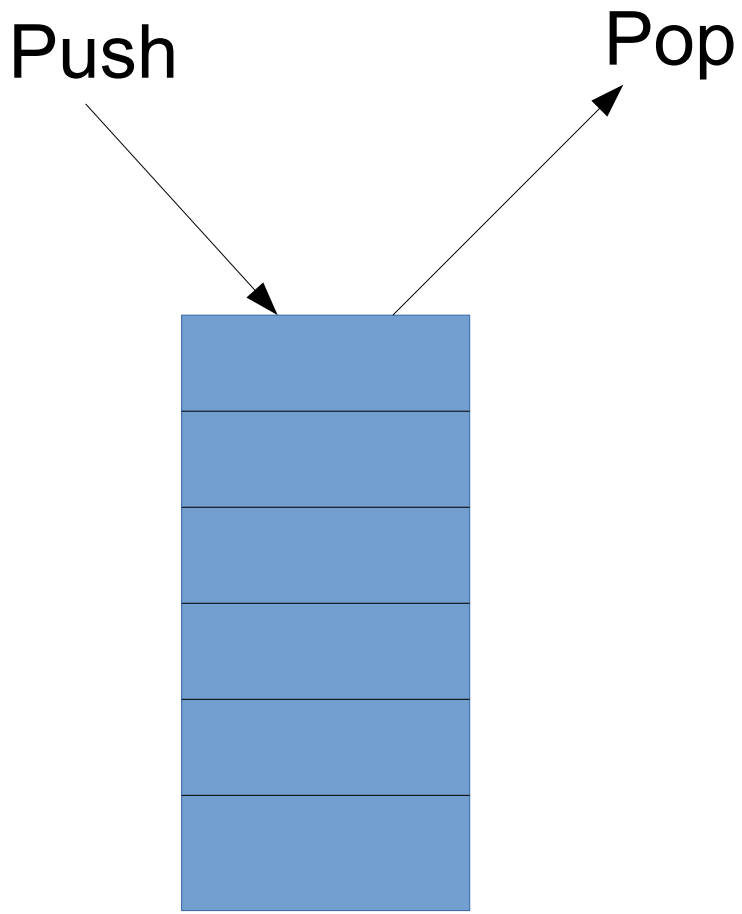
C++ program (skompilovaný kód)

C++ pamäť

```
#include <iostream>
using std::cout;
using std::endl;
int main() {
  int a = 3;
  int b = 3;
  int* c  = (int*) malloc(sizeof(int));
  std::cout << &a << endl; //0x7ffeb10576a8, zásobník
  std::cout << &b << endl; //0x7ffeb10576ac, zásobník
  std::cout << c << endl; //0x55fa820c5e70, heap
  return 0; }
```

The stack

Zásobník

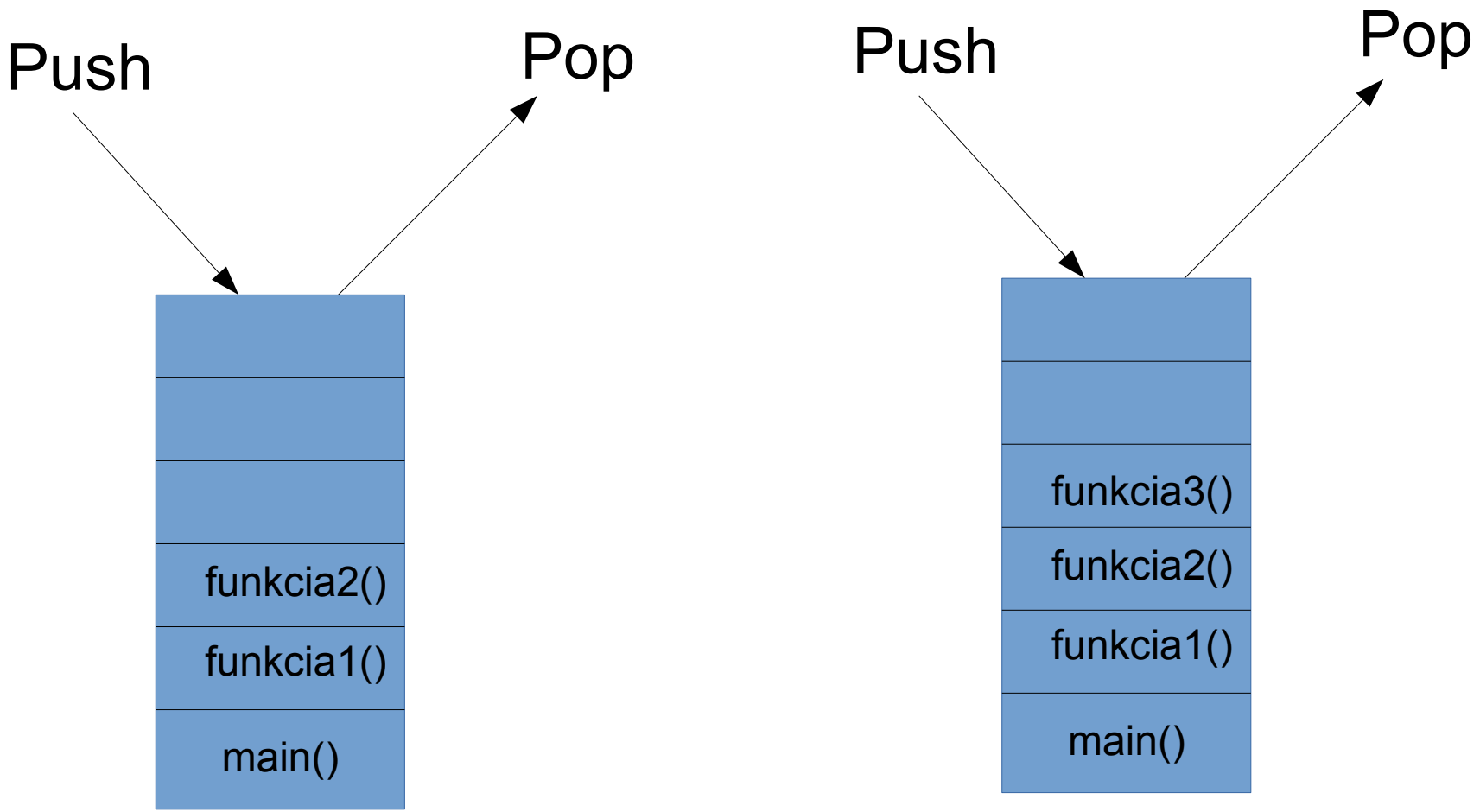


Dve operácie:

Vlož (push)

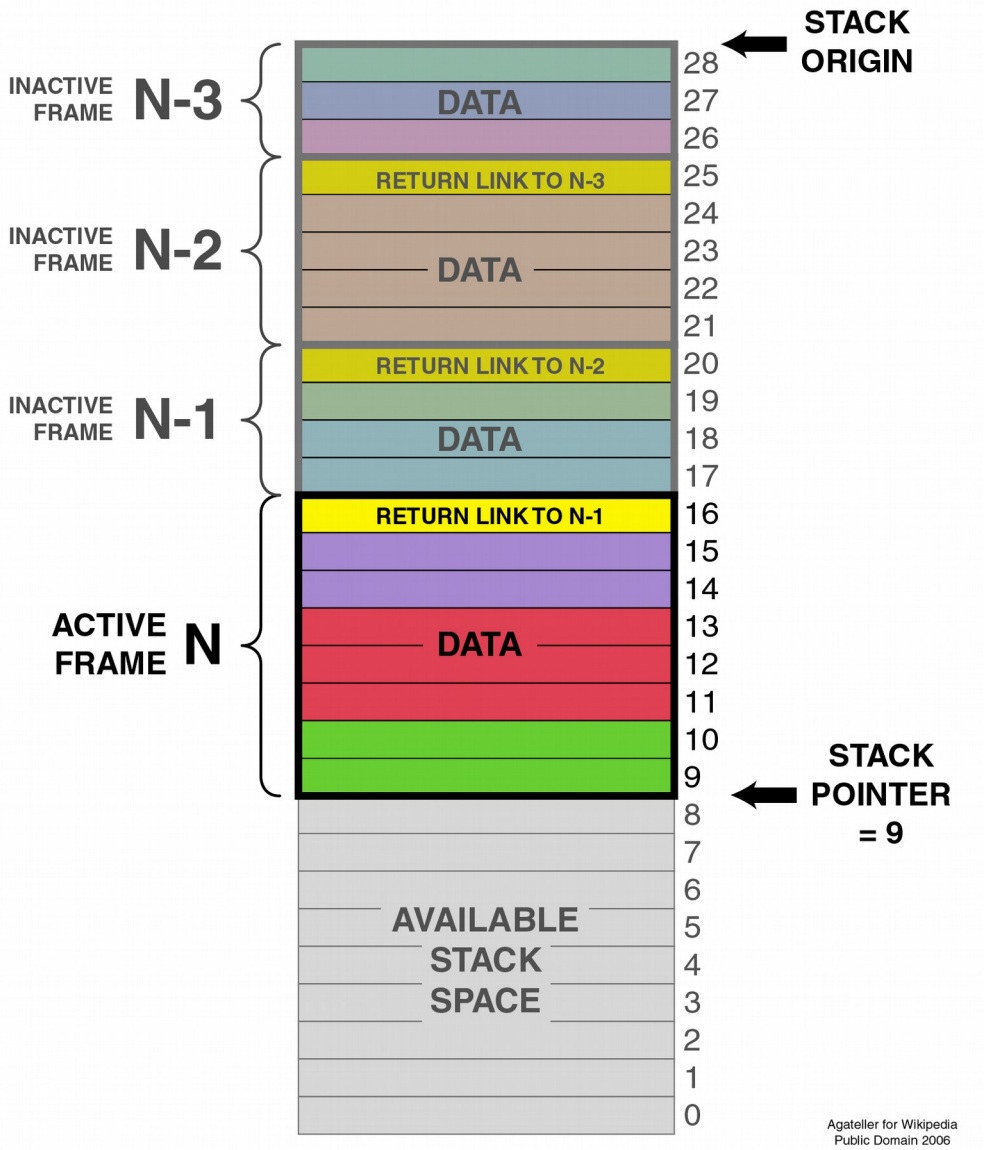
Vyber (pop)

Zásobník



Main() obsahuje
funkcia1() a tá obsahuje
funkcia2()

Zásobník



DATA = lokálne premenné, dočasné data

Zásobník

Veľkosť zásobníka:

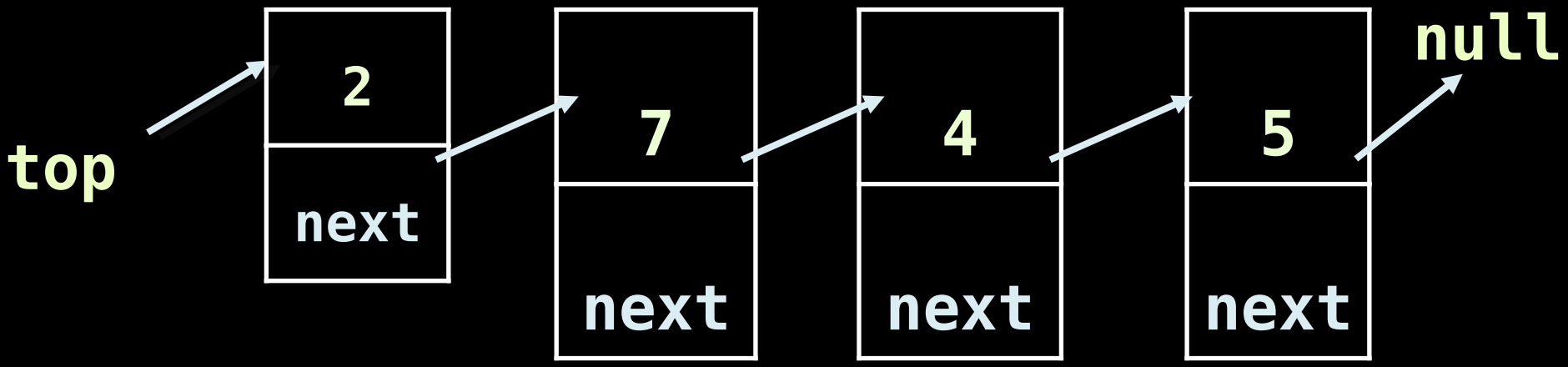
1-32 MB podľa architektúry

Keď sa zásobník naplní, dôjde k pretečeniu.

http://www.cs.nyu.edu/exact/core/doc/stackOverflow.txt

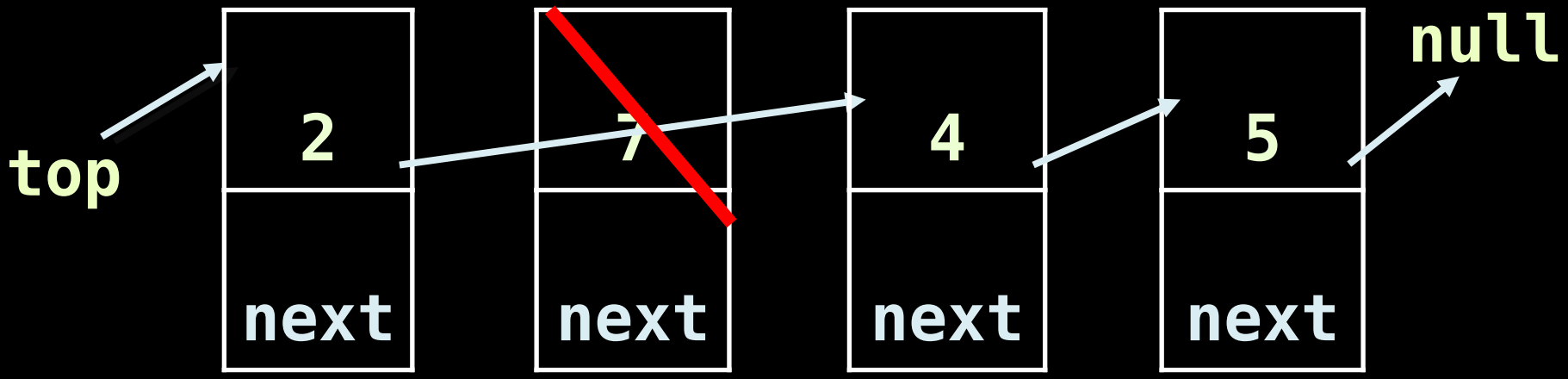
Úloha: vieš napísať kód, ktorý zapríčiní pretečenie zásobníka?

Zreťazený zoznam



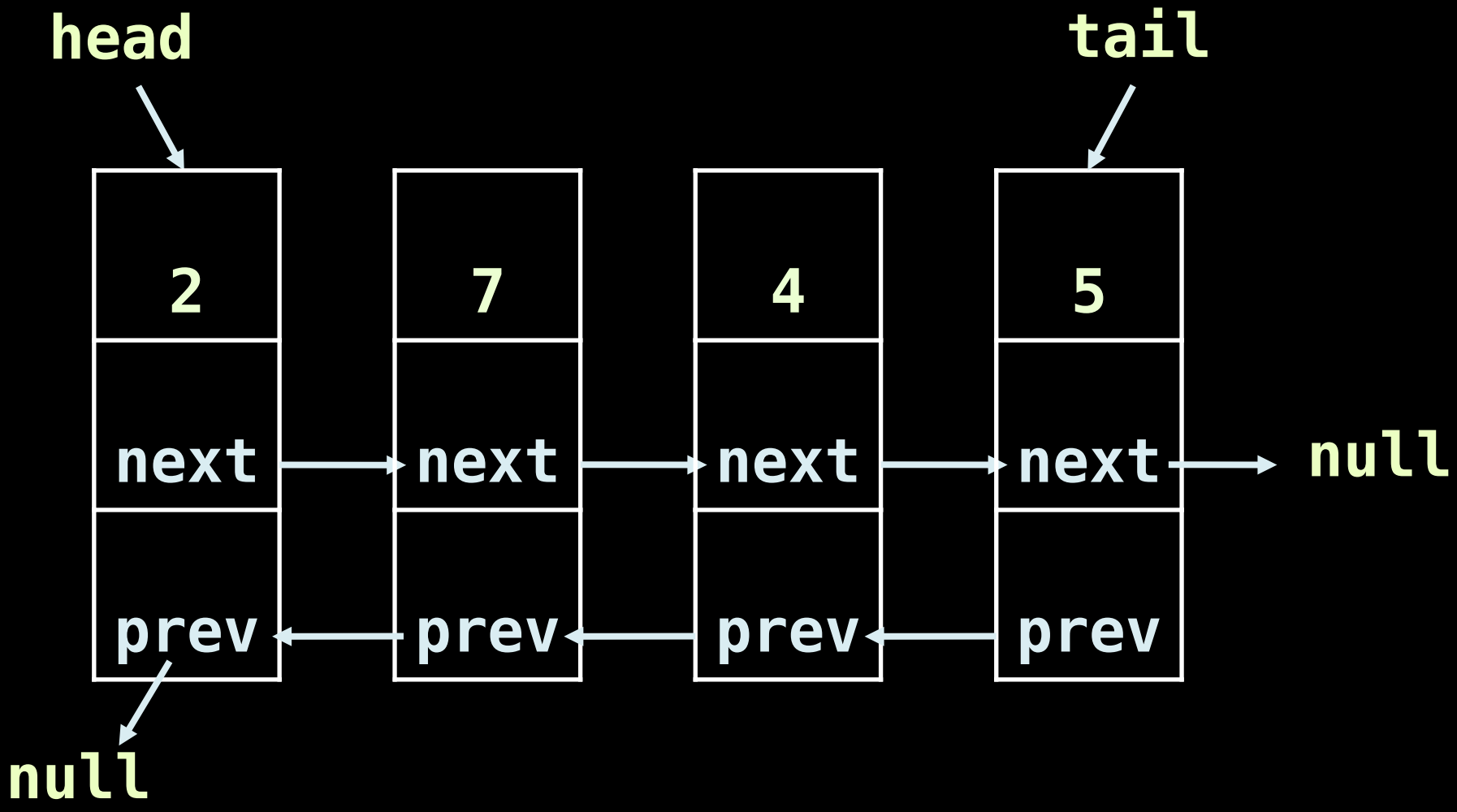
Každá položka má hodnotu a smerník na nasledujúcu položku

Zmazanie položky



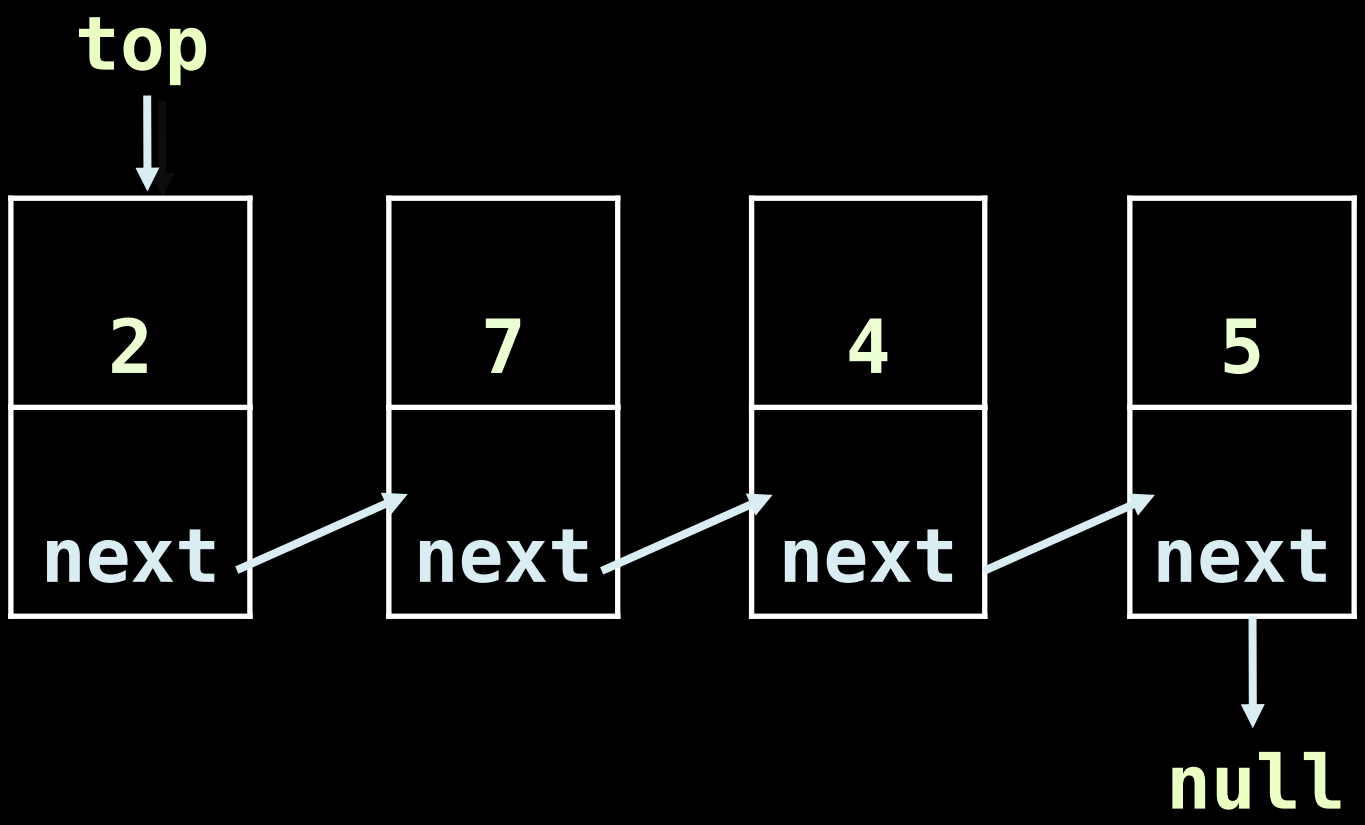
Ako pridať novú položku?

Dvojito zreťazený zoznam



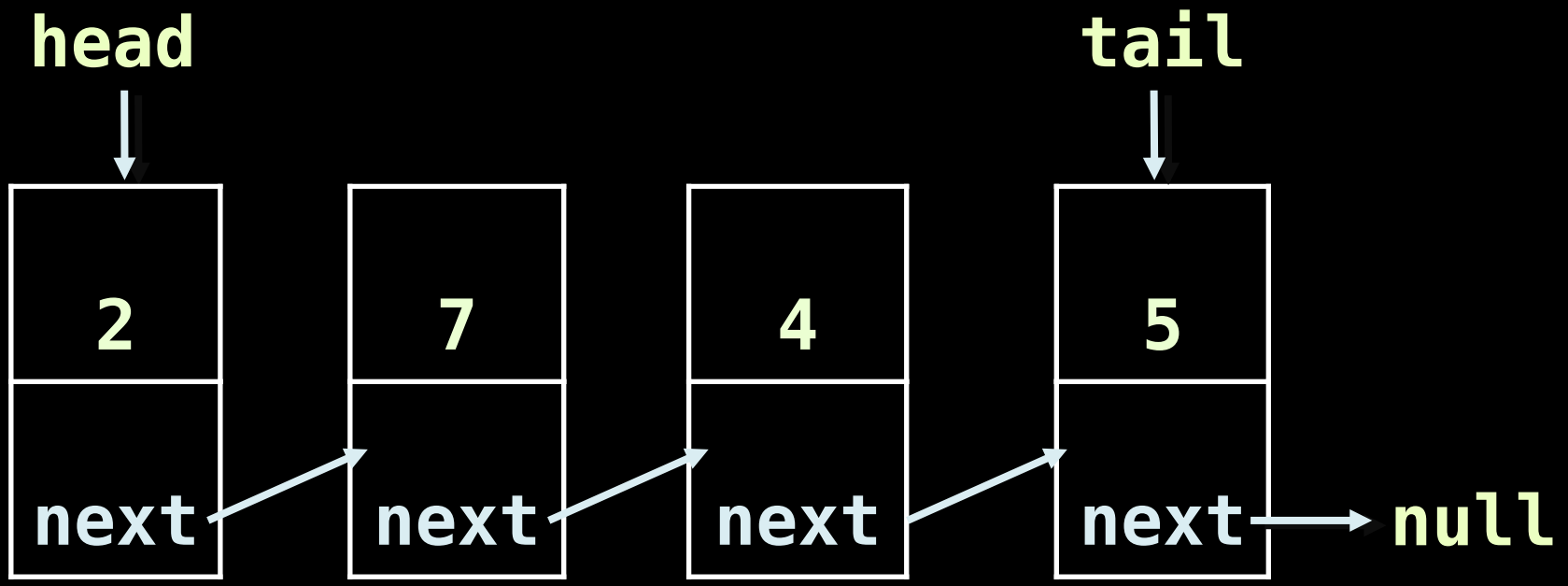
Každá položka má hodnotu a smerník na nasledujúcu a predchádzajúcu položku

Zásobník (stack)



top je smerník na položku, ktorá bola vložená ako posledná

Fronta (queue)



head je smerník na položku, ktorá bola vložená ako posledná, **tail** smerník na položku, ktorá bola vložená ako prvá

insert, delete, merge

Pole: ak je potrebný prístup cez index

Zreťazený zoznam: ak je potrebný push a pop

Ak chcem nájsť konkrétnu položku, je lepší zreťazený zoznam alebo pole?

Čo sa zmení, keď usporiadam prvky v poli (od najmenšieho po najväčšie)?

Merge: zreťazený zoznam alebo pole?

položka

```
struct item {
int value;
struct item *next;
struct item *prev;
};
```

Úloha

Naprogramuj zreťazený zoznam!

(turoriál)

Chvostová rekurzia

```
#include <iostream>
void rekurzia() {

  static int i = 1;
  std::cout << i << std::endl;
  int pole[1000];
  ++i;

  rekurzia();  //chvostová rekurzia, tail recursion
}
```