

Programovacie techniky

2. Nedefinované správanie, typová bezpečnosť, menné priestory, organizácia pamäte, C++ zásobník, zásobník, fronta, zret'azený 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;
```

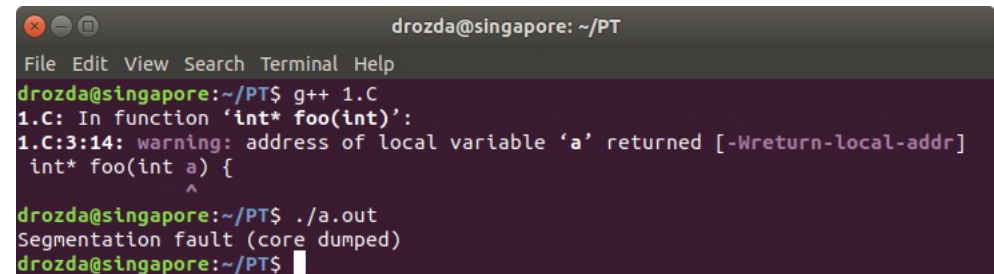
```
} ← Lokálna premenná zanikne
```

```
int main() {
```

```
    std::cout << *foo(3);
```

```
    return 0;
```

```
}
```



```
drozda@singapore: ~/PT
File Edit View Search Terminal Help
drozda@singapore:~/PT$ g++ 1.C
1.C: In function 'int* foo(int)':
1.C:3:14: warning: address of local variable 'a' returned [-Wreturn-local-addr]
    int* foo(int a) {
        ^
drozda@singapore:~/PT$ ./a.out
Segmentation fault (core dumped)
drozda@singapore:~/PT$
```


Nedefinované správanie

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

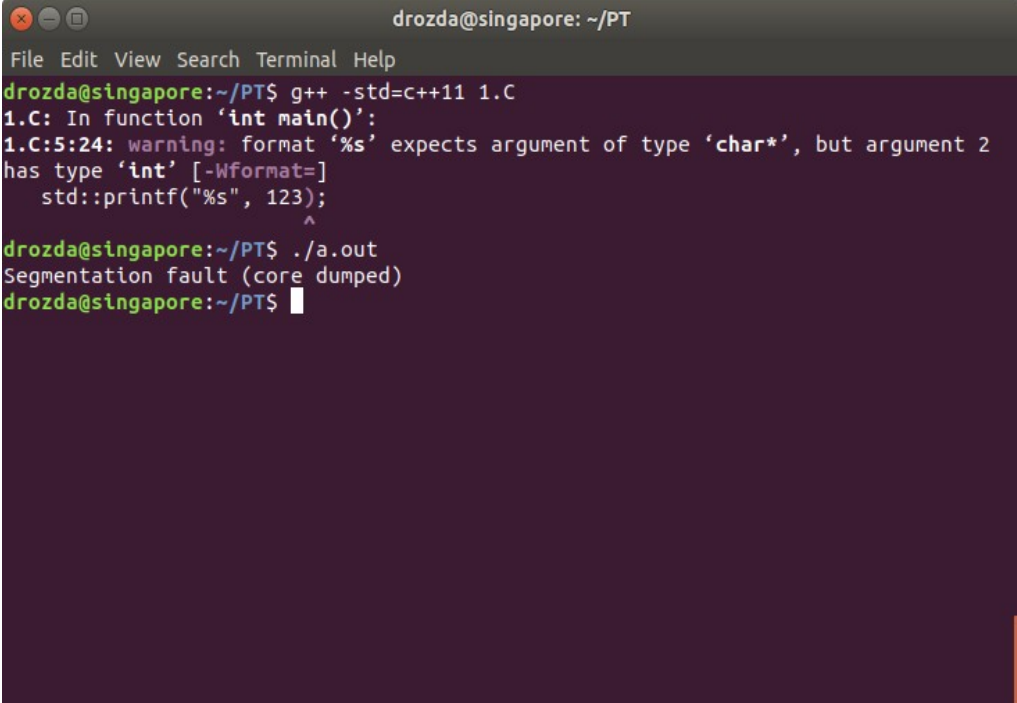
Príklady:

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

Typová bezpečnost

```
#include <cstdio>

int main() {
    std::printf("%s", 123);
    return 0;
}
```



```
drozda@singapore: ~/PT
File Edit View Search Terminal Help
drozda@singapore:~/PT$ g++ -std=c++11 1.C
1.C: In function 'int main()':
1.C:5:24: warning: format '%s' expects argument of type 'char*', but argument 2
has type 'int' [-Wformat=]
    std::printf("%s", 123);
                    ^
drozda@singapore:~/PT$ ./a.out
Segmentation fault (core dumped)
drozda@singapore:~/PT$
```

Typová bezpečnosť

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

```
#include <iostream>
```

```
struct Token {
```

```
    int a = 0;
```

```
};
```

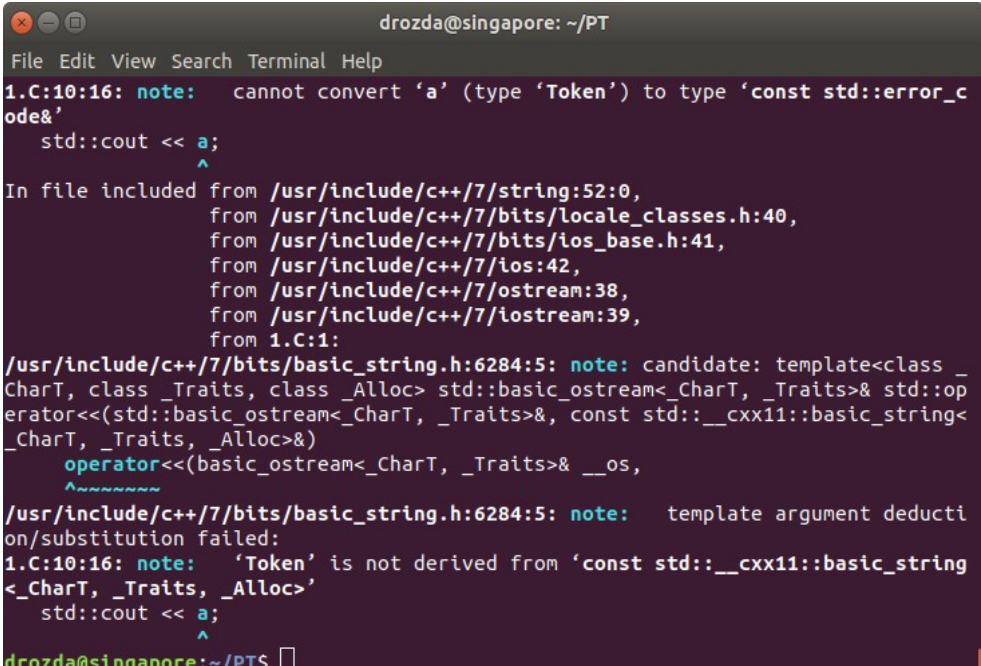
```
int main() {
```

```
    Token a;
```

```
    std::cout << a;
```

```
    return 0;
```

```
}
```



```
drozda@singapore: ~/PT
File Edit View Search Terminal Help
1.C:10:16: note:   cannot convert 'a' (type 'Token') to type 'const std::error_c
ode&'
    std::cout << a;
                    ^
In file included from /usr/include/c++/7/string:52:0,
                 from /usr/include/c++/7/bits/locale_classes.h:40,
                 from /usr/include/c++/7/bits/ios_base.h:41,
                 from /usr/include/c++/7/ios:42,
                 from /usr/include/c++/7/ostream:38,
                 from /usr/include/c++/7/iostream:39,
                 from 1.C:1:
/usr/include/c++/7/bits/basic_string.h:6284:5: note: candidate: template<class _
CharT, class _Traits, class _Alloc> std::basic_ostream<_CharT, _Traits>& std::op
erator<<(std::basic_ostream<_CharT, _Traits>&, const std::__cxx11::basic_string<
_CharT, _Traits, _Alloc>&)
    operator<<(basic_ostream<_CharT, _Traits>& __os,
              ^~~~~~
/usr/include/c++/7/bits/basic_string.h:6284:5: note:   template argument deducti
on/substitution failed:
1.C:10:16: note:   'Token' is not derived from 'const std::__cxx11::basic_string
<_CharT, _Traits, _Alloc>'
    std::cout << a;
                    ^
drozda@singapore:~/PT$
```

Životnosť premennej, objektu

```
int main() {
```

```
    int a = 3; ← Dočasný objekt (rvalue) 3 zanikne
```

```
    return 0;
```

```
} ← 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++ pamät'

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

Globálne a statické premenné

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

C++ program (skompilovaný kód)

C++ pamät'

```
#include <iostream>

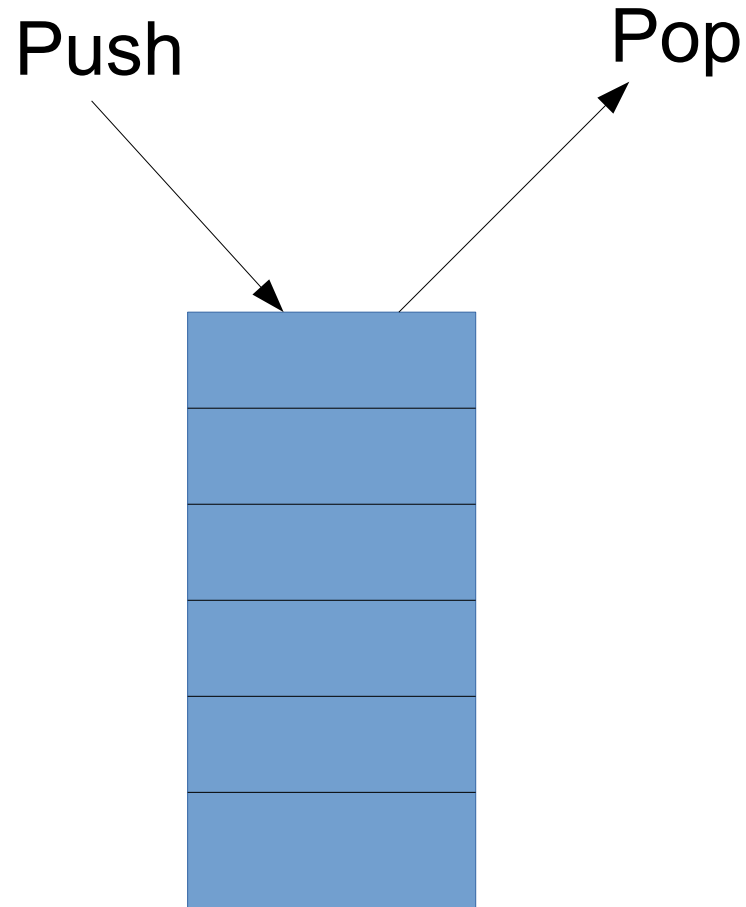
using std::cout;
using std::endl;

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

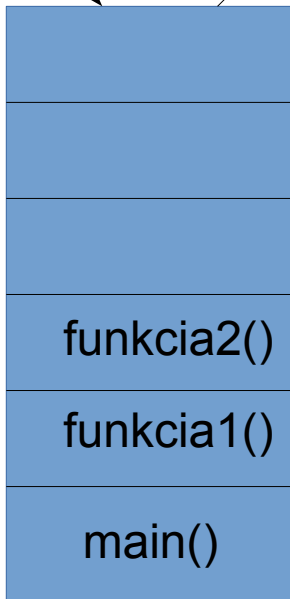


Dve operácie:

Vlož (push)

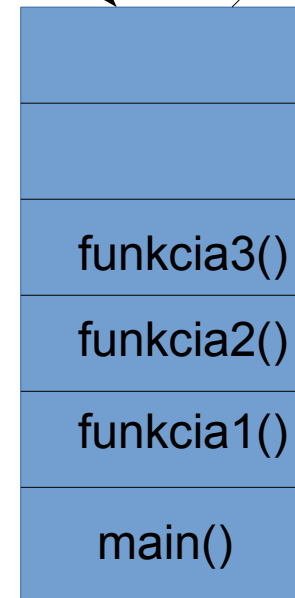
Vyber (pop)

Push



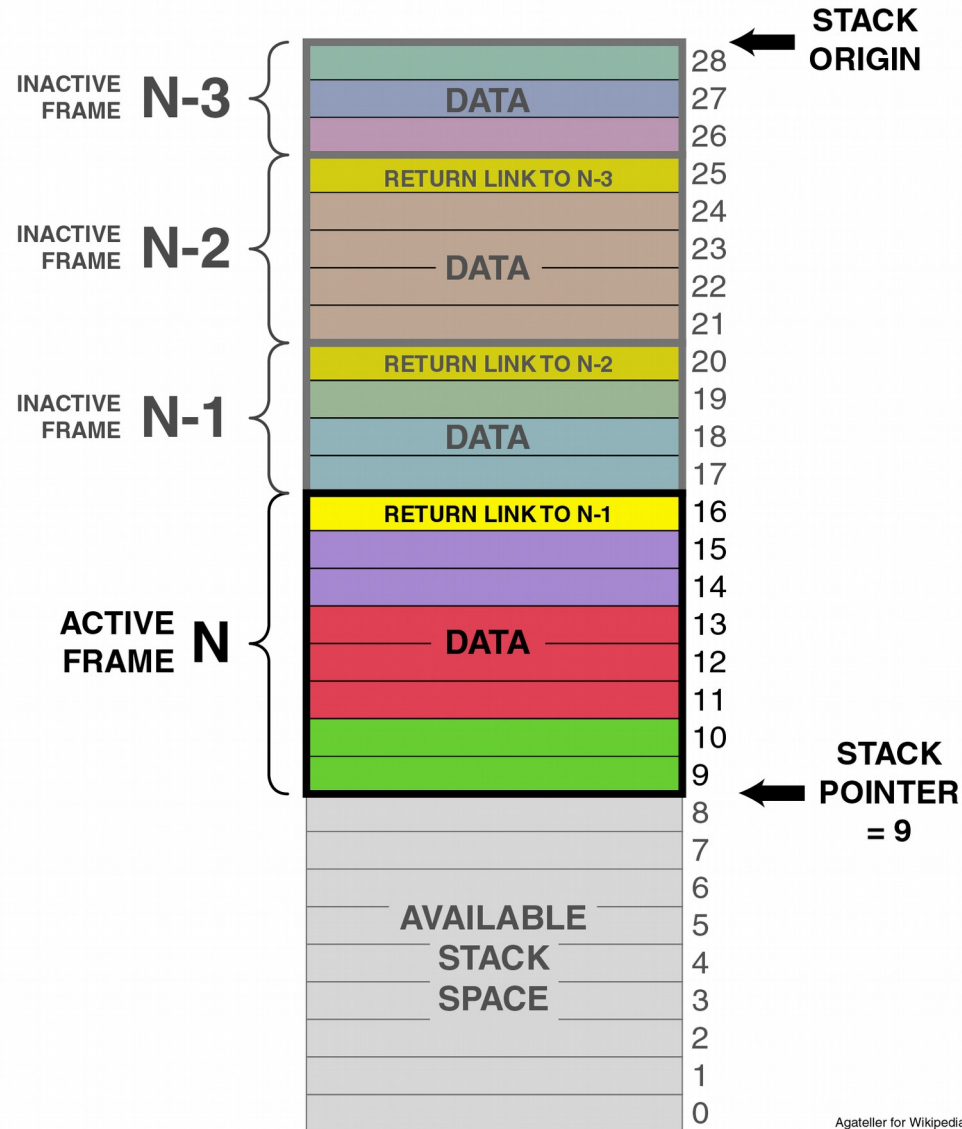
Pop

Push



Pop

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



Agateller for Wikipedia
Public Domain 2006

DATA = lokálne premenné, dočasné data

Veľkosť zásobníka:

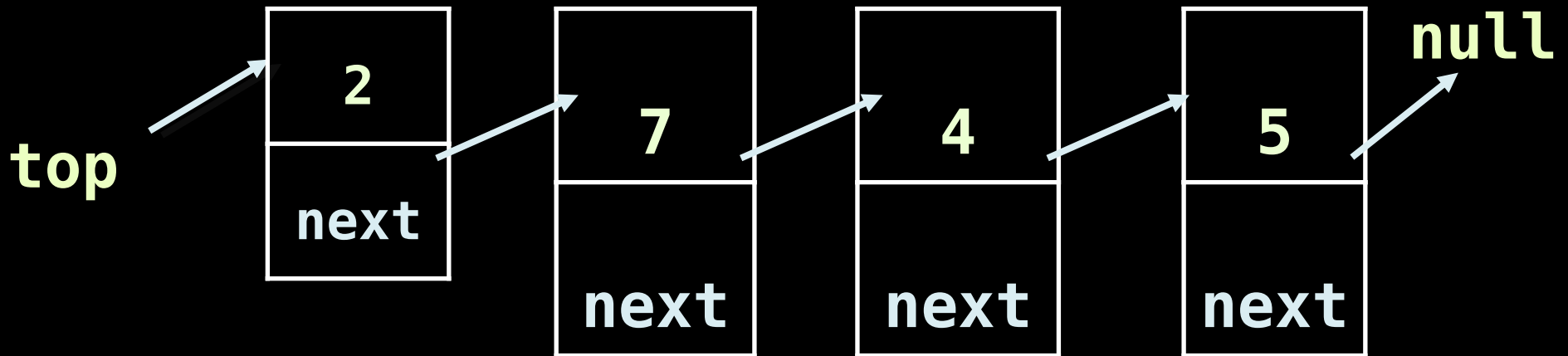
1-32 MB podľa architektúry

Keď sa zásobník naplní, dôjde k pretečeniu.

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

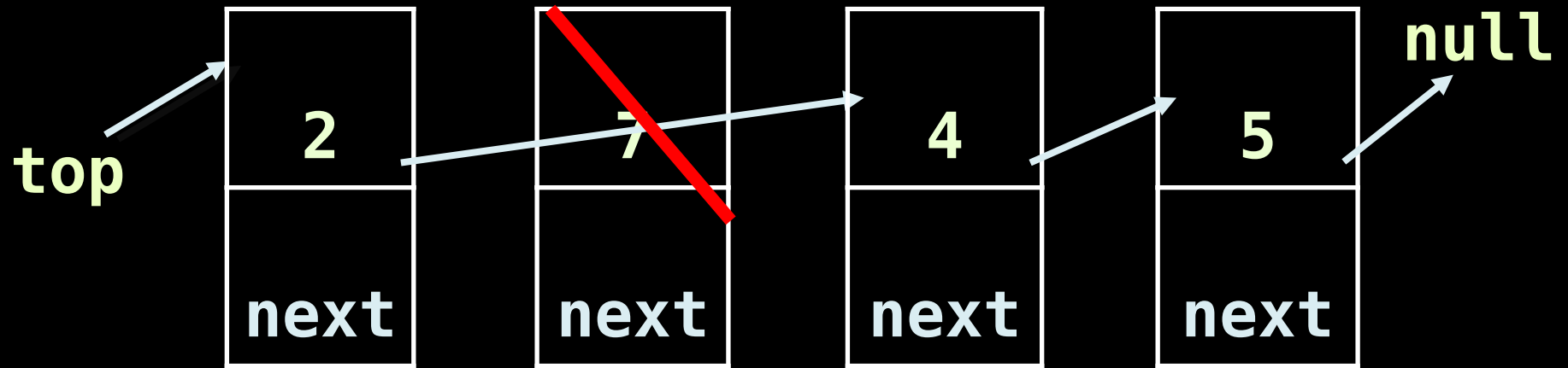
Úloha: vieš napísať kód, ktorý zapríčiní pretečenie zásobníka?

Zreťazený zoznam



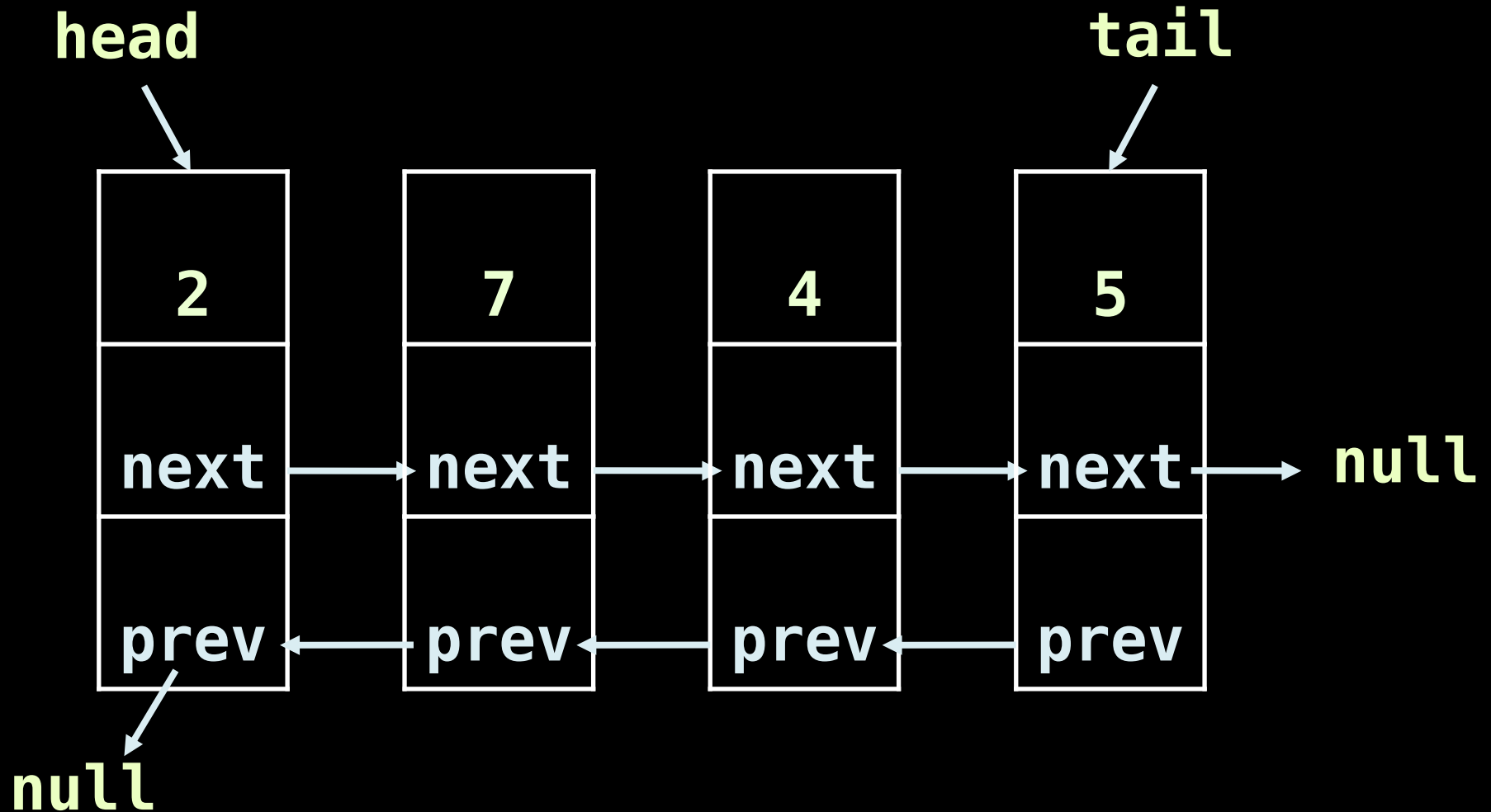
Každá položka má hodnotu a smerník na nasledujúcu položku

Zmazanie položky



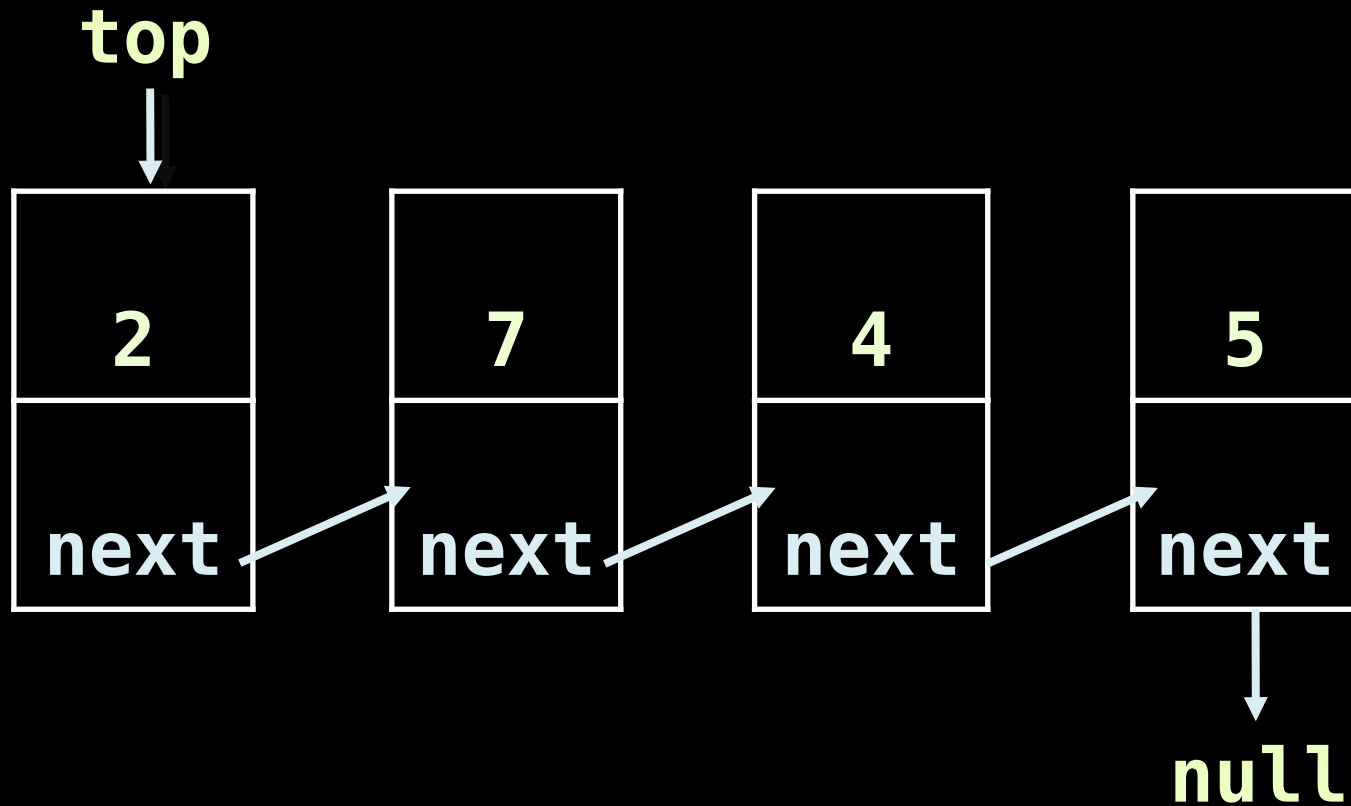
Ako pridať novú položku?

Dvojito zreťazený zoznam



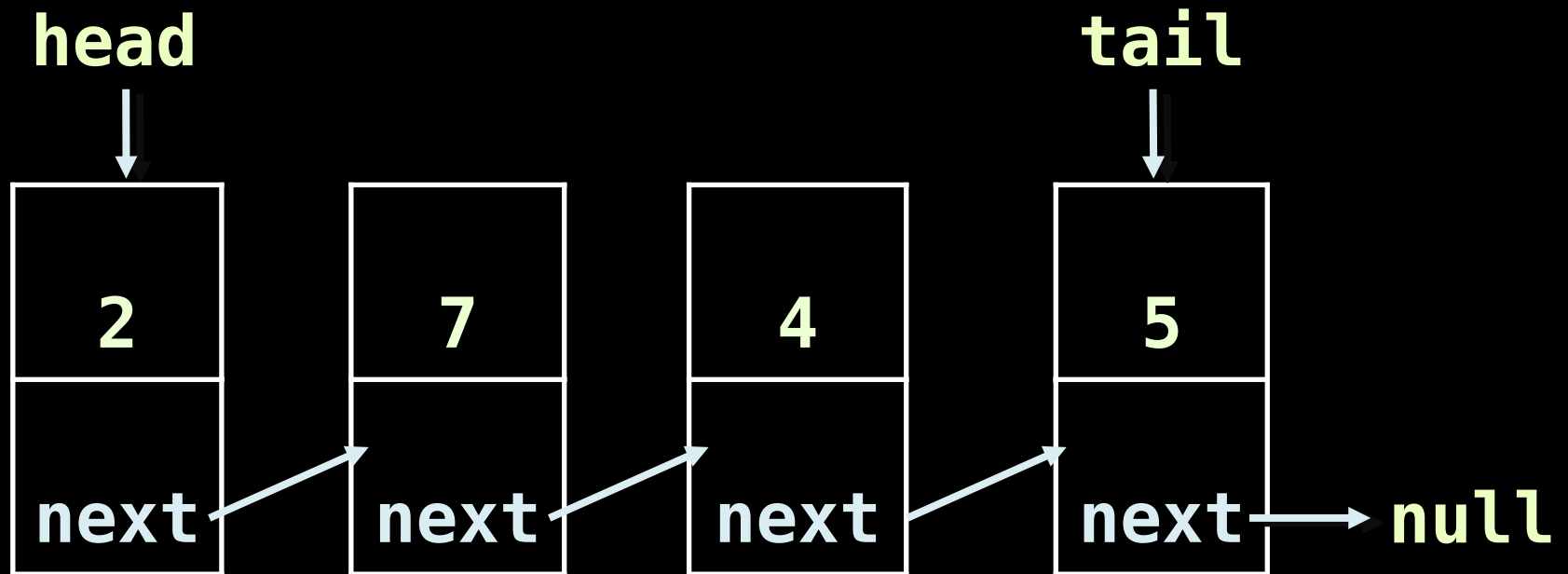
Každá položka má hodnotu a smerník na nasledujúcu a predchádzajúcu položku

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?

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

Naprogramuj zreťazený zoznam!

(tutoriál)

Chvostová rekurzia

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

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

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