

# Алгоритмизация и программирование

## Лекция 4

# Синтаксис лямбда-функций

C++11	<div>[CAPTURES] {BODY}</div> <div>[CAPTURES] -&gt; RETURN-TYPE {BODY}</div>					
C++23	<div>[CAPTURES] SPECIFIERS {BODY}</div> <div>[CAPTURES] SPECIFIERS -&gt; RETURN-TYPE {BODY}</div>					
C++11	<div>[CAPTURES] (PARAMETERS) {BODY}</div> <div>[CAPTURES] (PARAMETERS) -&gt; RETURN-TYPE {BODY}</div> <div>[CAPTURES] (PARAMETERS) SPECIFIERS {BODY}</div> <div>[CAPTURES] (PARAMETERS) SPECIFIERS -&gt; RETURN-TYPE {BODY}</div>					
C++20	<div>[CAPTURES] &lt;TYPE-PARAMETERS&gt; (PARAMETERS) {BODY}</div> <div>[CAPTURES] &lt;TYPE-PARAMETERS&gt; (PARAMETERS) -&gt; RETURN-TYPE {BODY}</div> <div>[CAPTURES] &lt;TYPE-PARAMETERS&gt; (PARAMETERS) SPECIFIERS {BODY}</div> <div>[CAPTURES] &lt;TYPE-PARAMETERS&gt; (PARAMETERS) SPECIFIERS -&gt; RETURN-TYPE {BODY}</div>					
SPECIFIERS	C++11			C++17	C++20	
	mutable	noexcept	[[attribute]]	constexpr	constexpr	requires...

# Синтаксис лямбда-функций

```
auto f1 = [] { return 47; };          auto x = f1();          // ⇔ int x = 47; C++11
auto f2 = [] (int x, int y) { return 0.5*(x+y); }; auto y = f2(2,3); // ⇔ double y = 2.5;
auto f3 = [] (int x) -> float { return x*x; }; auto z = f3(2); // ⇔ float z = 4.0f;
```

## Capturing of variables from the surrounding scope

### by reference

```
int x = 1;
auto f = [&] (int y) { x += 2; return x*y; };
cout << f(2); // 6
cout << x; // 3
```

### by value

```
int x = 1;
auto f = [=] (int y) { return x*y; };
cout << f(2); // 2
cout << x; // 1
```

[=]	capture all by value
[&]	capture all by reference
[=,&x]	x by reference, all others by value
[&,x]	x by value, all others by reference
[x,&y]	only x by value and y by reference
[&x,y]	only x by reference and y by value
[&x,y,&z]	only x by ref, y by value and z by ref

### init captures define lambda-local variables

```
auto f = [x=2] (int y) { return x*y; };
std::vector<char> v (1000, 'a');
auto g = [w=std::move(v)] () { /* use w */ };
```

C++14

## Immediately Invoked Function Expressions (IIFE)

```
int z = [] (int x) { return x*x; } (2); // create lambda and call it ⇔ int z = 4;
```

## Mutable Lambdas

```
int y = 1;
auto f = [=] () {
    y += 2;
    return y;
};
```

COMPILER ERROR:  
local variable  
'y' is const!

```
int y = 1;
auto f = [=] () mutable {
    y += 2;
    return y;
};
cout << f(); // 3
cout << f(); // 5
cout << y; // 1
```

```
auto f = [i=0] () mutable {
    ++i; return i;
};
cout << f(); // 1
cout << f(); // 2
```

## Generic Lambdas

C++14

```
auto print = [] (auto const& x) { std::cout << x; };
print(5); ✓
print(std::string{"it works!"}); ✓
```

```
[] (auto value, auto const& cref, auto& ref) { ... }
[] (auto... args) { return g(args...); }
```

### Perfect Forwarding preserving constness, l/r-valueness

```
[] (auto&& x) { g( std::forward<decltype(x)>(x) ); }
[] (auto&&... args) {
    g( std::forward<decltype(args)>(args)... ); }
```

### Constrained auto Parameters

C++20

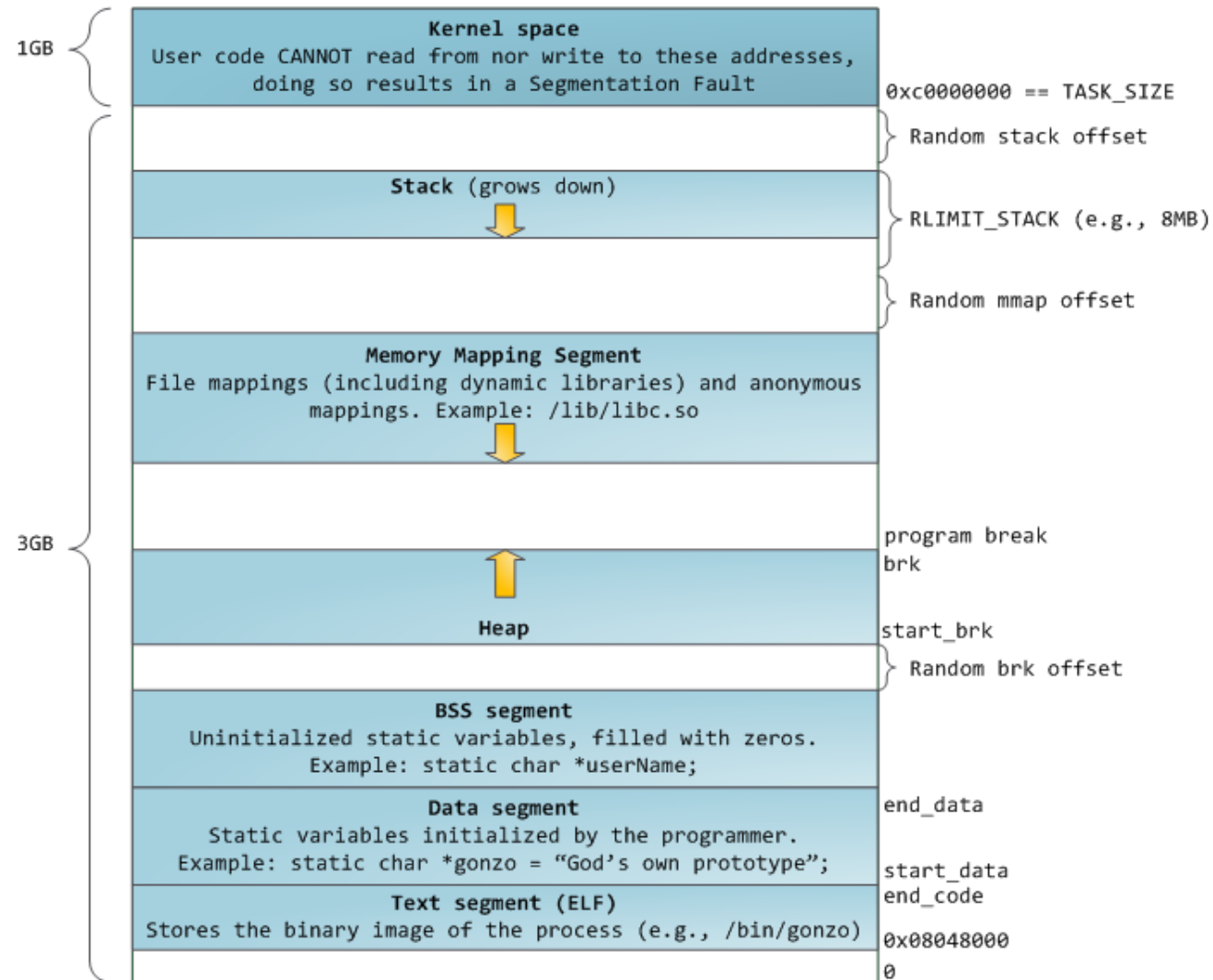
```
#include <concepts>
[] (std::copyable auto x) { ... }
```

### Explicit Template Parameters

C++20

```
[] <typename T> (T x, T y) { ... }
```

# Анатомия программы в памяти



# Стековый кадр

