**Lambda sinaktsa**

[*capture\_klauzula*] (*ulazni\_parameteri*) -> *povratni\_tip* { *tijelo\_funckije* }

**Lambda sinaktsa – dodjela lambda funkcije pokazivaču i poziv funkcije**

auto pok = [*capture\_klauzula*] (*ulazni\_param*) -> *pov\_tip* { *tijelo\_f* };

*pov\_tip* x = pok(*ulazne\_vrijednosti*);

**capture klauzula**

|  |  |
| --- | --- |
| [] | Capture nothing (or, a scorched earth strategy?) |
| [&] | Capture any referenced variable by reference |
| [=] | Capture any referenced variable by making a copy |
| [=, &foo] | Capture any referenced variable by making a copy, but capture variable foo by reference |
| [bar] | Capture bar by making a copy; don't copy anything else |
| [this] | Capture the this pointer of the enclosing class |

# Zadatak 1

1. Prepravite naredni program tako da u main-u koristite pokazivač na funkciju
   1. sa korištenjem varijable tipa **auto**.
   2. bez korištenja varijable tipa **auto**

|  |
| --- |
| int saberi(int a, int b)  {  return a + b;  }  void main()  {  int m, n;  cin >> m >> n;  int x = saberi(m, n);  cout << x << endl;  } |

|  |
| --- |
| **Rješenje** |
|  |

1. Prepravite rješenje zadataka, tako da se koriste lambda funkcija.
   1. sa korištenjem varijable tipa auto.
   2. bez korištenja varijable tipa auto

|  |
| --- |
| **Rješenje** |
|  |

# Zadatak 2 - Lambda i pokazivač na funkciju

1. Prepravite naredni program tako da u main-u koristite pokazivač na funkciju
   1. sa korištenjem varijable tipa **auto**.
   2. bez korištenja varijable tipa **auto**

|  |
| --- |
| float rjesenje\_kvadratne\_jednacine(float a, float b, float c)  {  float D = b\*b - 4 \* a\*c;  return (-b + sqrt(D)) / (2 \* a);  }  void main()  {  float a,b,c;  cin>>a>>b>>c;  float x1 = rjesenje\_kvadratne\_jednacine(a,b,c);  cout<<x1<<endl;  //pokazivač p na funkciju rjesenje\_kvadratne\_jednacine  //pozvati funkciju p  } |

|  |
| --- |
| **Rješenje** |
|  |

1. Prepravite prethodno rješenje tako da se koriste lambda funkcije
   1. sa korištenjem varijable tipa **auto**.
   2. bez korištenja varijable tipa **auto**

|  |
| --- |
| **Rješenje** |
|  |

# Zadatak 3 - Lambda i pokazivač na funkciju

Rješenje zadatka prepravite tako da umjesto funkcija zadatak-1, -2, -3 i -4 koristite lambda funkcije unutar **odaberi\_zadatak**.

|  |
| --- |
| #include <iostream>  using namespace std;  typedef void (FunkTip)();  void zadatak1()  {  //pauziranje na 1 sek  \_sleep(1000);  cout << "radni zadatak 1 zavrsen" << endl;  }  void zadatak2()  {  //pauziranje na 1 sek  \_sleep(1000);  cout << "radni zadatak 2 zavrsen" << endl;  }  void zadatak3()  {  //pauziranje na 1 sek  \_sleep(1000);  cout << "radni zadatak 3 zavrsen" << endl;  }  void zadatak4()  {  //pauziranje na 1 sek  \_sleep(1000);  cout << "radni zadatak 4 zavrsen" << endl;  }  FunkTip\* odaberi\_zadatak()  {  cout << "Odaberite zadatak 1 - 4: " << endl;  int r;  cin >> r;  if (r == 1)  return zadatak1;  if (r == 2)  return zadatak2;  if (r == 3)  return zadatak2;  if (r == 4)  return zadatak2;  return nullptr;  }  struct TodoRaspored  {  int brojacZadatakaNiz = 0;  FunkTip\* zadaci[10];  };  void dodaj\_zadatak(TodoRaspored\* todo, FunkTip\* newZadatak)  {  todo->zadaci[todo->brojacZadatakaNiz] = newZadatak;  todo->brojacZadatakaNiz++;  }  void ukloni\_zadatak(TodoRaspored\* todo)  {  int rb;  cin >> rb;  todo->brojacZadatakaNiz--;  todo->zadaci[rb] = todo->zadaci[todo->brojacZadatakaNiz];  }  void izvrsi\_zadatke(TodoRaspored\* todo)  {  for (int i = 0; i < todo->brojacZadatakaNiz; i++)  {  FunkTip\* p = todo->zadaci[i];  p();  }  }  void main()  {  TodoRaspored\* danas = new TodoRaspored;  cout << "TodoRaspored danas" << endl;  dodaj\_zadatak(danas, odaberi\_zadatak());  dodaj\_zadatak(danas, odaberi\_zadatak());  FunkTip\* pZ3 = odaberi\_zadatak();  dodaj\_zadatak(danas, pZ3);  TodoRaspored\* sutra = new TodoRaspored;  cout << "TodoRaspored sutra" << endl;  dodaj\_zadatak(sutra, odaberi\_zadatak());  dodaj\_zadatak(sutra, odaberi\_zadatak());  dodaj\_zadatak(sutra, odaberi\_zadatak());  izvrsi\_zadatke(danas);  system("pause");  izvrsi\_zadatke(sutra);  } |

|  |
| --- |
| Rješenje |
|  |

# Zadatak 3 - Capturing the Context

Koji je ispis naredna dva programa? Obrazložite rezultat programa.

|  |
| --- |
| void main()  {  int int\_var = 42;  auto lambda\_func = [int\_var]() {  cout << "lambda funkcija ima kopiju varijable int\_var: " << int\_var << endl;  };  lambda\_func();  for (int i = 0; i < 3; i++)  {  int\_var++;  lambda\_func();  }  } |

|  |
| --- |
| void main()  {  int int\_var = 42;  auto lambda\_func = [&int\_var]()  {  cout << "lambda funkcija ima referencu na int\_var: " << int\_var << endl;  };  lambda\_func();  for (int i = 0; i < 3; i++) {  int\_var++;  lambda\_func();  }  } |

# Zadatak 4

Prepravite rješenje zadatka 1b tako da se koristi lambda funkcija, a vrijednosti m i n se preuzimaju preko capture-klauzule.

# Zadatak 5- Implementirati kalkulator

1. naredbu za unos nove vrijednost (cin) smjestite u lambda funkciju
2. naredbu za unos nove vrijednost (cin) smjestite u main funkciju

|  |
| --- |
| #include <iostream>  using namespace std;  void main()  {  float result = 0;  float memory = 0;  auto f1\_dodaj = []() { };  auto f2\_oduzmi = []() { };  auto f3\_pomnozi = []() { };  auto f4\_korijen = []() { };  auto f5\_kvadrat = []() { };  auto f6\_stepenuj = []() { };  auto f7\_memory\_store = []() { };  auto f8\_memory\_recall = []() { };  auto f9\_memory\_clear = []() { };  auto f10\_result\_reset = []() { };  int x = 0;  do  {  system("cls");  cout << "RESULT = " << result << endl;  cout << "MEMORY = " << memory << endl;  cout << "1. saberi" << endl;  cout << "2. oduzmi" << endl;  cout << "3. pomnozi" << endl;  cout << "4. korijen" << endl;  cout << "5. kvadrat" << endl;  cout << "6. stepenuj" << endl;  cout << "7. memory\_store" << endl;  cout << "8. memory\_recall" << endl;  cout << "9. memory\_clear " << endl;  cout << "10. result reset " << endl;  cout << "0. EXIT " << endl;  cin >> x;  switch (x)  {  case 1:  f1\_dodaj(); break;  case 2:  f2\_oduzmi(); break;  case 3:  f3\_pomnozi(); break;  case 4:  f4\_korijen(); break;  case 5:  f5\_kvadrat(); break;  case 6:  f6\_stepenuj(); break;  case 7:  f7\_memory\_store(); break;  case 8:  f8\_memory\_recall(); break;  case 9:  f9\_memory\_clear(); break;  case 10:  f10\_result\_reset(); break;  default:  break;  }  system("pause");  } while (x != 0);  } |

|  |
| --- |
| **rješenje - a** |
|  |

|  |
| --- |
| **rješenje - b** |
|  |

# Zadatak 6 – Lambda i pokazivač na funkciju

U nastavku su prikazana tri načina deklarisana pokazivača na funkciju (f1, f2 i f3)

|  |
| --- |
| #include <iostream>  #include <functional>  using namespace std;  bool NekaF(int v)  {  return (v > 0);  }  typedef bool(\*Tip)(int); // samo za f2  void main()  {  int c;  cin >> c;  //1. pokazivač na funkciju f1  bool(\*f1)(int) = NekaF;  if (f1(c))  cout << "Unjeli ste pozitivan broj" << endl;    //2. pokazivač na funkciju f2 uz korištenje typedef  Tip f2 = NekaF;  if (f2(c))  cout << "Unjeli ste pozitivan broj" << endl;  //3. pokazivač na funkciju f3 uz korištenje generičke klase function  function<bool(int)> f3 = NekaF;  if (f3(c))  cout << "Unjeli ste pozitivan broj" << endl;  system("pause");  } |

Prepravite prethodni primjer tako da koristite lambda funkcije

1. bez korištenja **auto** kaotip podatka i bez korištenja *capture*-klauzule.

|  |
| --- |
| **Rješenje a** |
| //3. pokazivač na funkciju f3 uz korištenje generičke klase function  function<bool()> f3 = [c]() { return (c > 0); };  if (f3())  cout << "Unjeli ste pozitivan broj" << endl;  //4. pokazivač na funkciju f4 uz korištenje auto  auto f4 = [c](){ return (c > 0); };  if (f4())  cout << "Unjeli ste pozitivan broj" << endl; |

1. bez korištenja **auto** kaotip podatka ali sa korištenja *capture*-klauzule sa unos **c**.

Napomena: Moguće samo rješenje sa uz korištenje generičke klase function – f3 ???

|  |
| --- |
| #include <iostream>  #include <functional>  using namespace std;  typedef bool(\*Tip)(); // samo za f2  void main()  {  int c;  cin >> c;  //1. pokazivač na funkciju f1  // no suitable conversion function from "lambda []bool ()->bool" to "bool (\*)(int)"  ~~bool(\*f1)(int) = [&c]()->bool{return (c > 0); };~~  ~~if (f1())~~  ~~cout << "Unjeli ste pozitivan broj" << endl;~~    //2. pokazivač na funkciju f2 uz korištenje typedef  // no suitable conversion function from "lambda []bool ()->bool" to "Tip"  ~~Tip f2 = [&c]()->bool {return (c > 0); };~~  ~~if (f2())~~  ~~cout << "Unjeli ste pozitivan broj" << endl;~~  //3. pokazivač na funkciju f3 uz korištenje generičke klase function  function<bool()> f3 = [&c]()->bool {return (c > 0); };  if (f3())  cout << "Unjeli ste pozitivan broj" << endl;  system("pause");  } |

# Zadatak 7 -

1. Dovršiti program bez korištenja lambda funkcija
2. Prepravite program tako da koristite lambda funkcije

|  |
| --- |
| #include <iostream>  #include <functional>  using namespace std;  const int max\_size = 30;  struct Kolekcija  {  int podaci[max\_size];  int brojac = 0;  };  void kDodaj(Kolekcija& k, int newV)  {  //dodaj newV u podaci na poziciju brojac  //uvecati brojac  }  void kGenerisiPodatke(Kolekcija& k)  {  for (int i = 0; i < max\_size / 2; i++)  {  int newV = rand() - rand();  //newV dodati u kolekciju k  }  }  void kPrinter(Kolekcija& k, char\* opis)  {  cout << opis << endl;  for (int i = 0; i < k.brojac; i++)  {  int& p = k.podaci[i];  cout << i << ": " << p << endl;  }  }  Kolekcija kFilter(Kolekcija& k, function<bool(int&)> f)  {  Kolekcija newK;  //iz kolekcije k u kolekciju newK kopirati elemente koji zadovoljavaju uslov f  return newK;  }  void main()  {  //a. Instaciraj Kolekciju k1 i generiši podatke    //b. Iz kolekcije k1 filtiraj parne brojeve u kolekciju k2    //c. Iz kolekcije k2 filtiraj pozitivne brojeve u kolekciju k3    //d. Ispisati na ekran sve tri kolekcije  } |

|  |
| --- |
| **Rješenje** |
|  |

# Zadatak 8

U prethodni program dodati funkciju kForEach

|  |
| --- |
| #include <iostream>  #include <functional>  using namespace std;  ...  void kForEach(Kolekcija& k, function<void(int&)> f)  {  //za svaki element iz kolekcije k izvrsi zadatak f  }  ...  void main()  {  //a. Instaciraj Kolekciju kSviBrojevi i generiši podatke  Kolekcija kSviBrojevi;  ...  //f. koristeci kForEach uvecati sve elemente kolekcije kParniBrojevi  ...  //g. koristeci kForEach kopirati sve elemente kolekcije kParniBrojevi u novu kolekciju  Kolekcija kopijaParniBrojevi;  ...    } |

|  |
| --- |
| #include <iostream>  #include <functional>  #include <algorithm>  using namespace std;  const int max\_size = 30;  struct Kolekcija  {  int podaci[max\_size];  int brojac = 0;  };  void kDodaj(Kolekcija& k, int newV)  {  //dodaj newV u podaci na poziciju brojac  //uvecati brojac  k.podaci[k.brojac++] = newV;  }  void kGenerisiPodatke(Kolekcija& k)  {  for (int i = 0; i < max\_size / 2; i++)  {  int newV = rand() - rand();  //newV dodati u kolekciju k  kDodaj(k, newV);  }  }  void kPrinter(Kolekcija& k, char\* opis)  {  cout << opis << endl;  for (int i = 0; i < k.brojac; i++)  {  int& p = k.podaci[i];  cout << i << ": " << p << endl;  }  }  Kolekcija kFilter(Kolekcija& k, function<bool(int&)> f)  {  Kolekcija newK;  for (size\_t i = 0; i < k.brojac; i++)  {    int &x = k.podaci[i];  if (f(x)) {  kDodaj(newK, x);  }  }  //iz kolekcije k u kolekciju newK kopirati elemente koji zadovoljavaju uslov f  return newK;  }  void kForEach(Kolekcija& k, function<void(int&)> f)  {  //za svaki element iz kolekcije k izvrsi zadatak f  for (int i = 0; i < k.brojac; i++)  {  int &x = k.podaci[i];  f(x);  }  }  void main()  {  //a. Instaciraj Kolekciju k1 i generiši podatke  Kolekcija k1;  kGenerisiPodatke(k1);  //b. Iz kolekcije k1 filtiraj parne brojeve u kolekciju k2  Kolekcija k2 = kFilter(k1, [](int &b)  {  return b % 2 == 0;  });  //c. Iz kolekcije k2 filtiraj pozitivne brojeve u kolekciju k3  Kolekcija k3 = kFilter(k2, [](int &b)  {  return b>0;  });  //d. Ispisati na ekran sve tri kolekcije    kPrinter(k1, "kolekcija 1 ");  kPrinter(k2, "kolekcija 2 ");  kPrinter(k3, "kolekcija 3 ");  //f. koristeci kForEach uvecati sve elemente kolekcije k3  kForEach(k3, [](int &b) {  b++;  });  //g. koristeci kForEach kopirati sve elemente kolekcije k2 u novu kolekciju  Kolekcija k4;  kForEach(k2, [&k4](int &v) {  kDodaj(k4, v);  });  } |

# Zadatak 9 – dodatna lekcija

Doradite prethodni program:

* u funkciju main dodajte poziv funkcije **kSortiraj** za sve tri kolekcije.

|  |
| --- |
| void rotiraj(int& m, int& n)  {  int t = m;  m = n;  n = t;  }  void kSortiraj(Kolekcija& k, function<bool(int&,int&)> t)  {  int n = k.brojac;  //granica između nesortiranog i sortiranog dijela niza  for (int granica = n - 1; granica >= 0; granica--)  {  for (int j = 0; j < granica; j++)  {  int& a = k.podaci[j];  int& b = k.podaci[j + 1];  if (t(a, b))  rotiraj(a, b);  }  }  }  void main()  {  ...  //h. Sortirati sve tri kolekcije i ispisati na ekran  } |

|  |
| --- |
| **Rješenje** |
|  |