Zadaća 3

Ova zadaća nosi **6 poena**. Rok za predaju je **nedjelja**, **17. XII 2023.** do kraja dana. Zadaće se predaju putem Zamgera. Broj poena koji student dobija ovisi od broja korektno implementiranih stavki koje se traže u zadaći.

Zadatak 1 (5 poena)

U ovom zadatku, potrebno je sastaviti šest klasa za podršku interpolaciji. Ove klase nazvane su redom "`LinearInterpolator`" (za linearnu interpolaciju), "`PolynomialInterpolator`" (za polinomsku interpolaciju), "`PiecewisePolynomialInterpolator`" (za interpolaciju polinomima dio po dio), zatim "`SplineInterpolator`" (za interpolaciju pomoću kubnog splajna), "`BarycentricInterpolator`" (za baricentričnu racionalnu interpolaciju) i "`TrigonometricInterpolator`" (za Gaussovu trigonometrijsku interpolaciju). Sve ove klase trebaju da budu izvedene iz jedne zajedničke apstraktne bazne klase nazvane "`AbstractInterpolator`", tako da će svaka funkcija čiji je parametar deklariran kao referenca ili pokazivač na tip "`AbstractInterpolator`" moći primiti primjerak bilo koje klase izvedene iz ovog tipa, dakle bilo koji od objekata koji implementira neku konkretnu metodu interpolacije (uključujući i metode koje bi mogle biti eventualno implementirane u budućnosti).

Apstraktna klasa "`AbstractInterpolator`" treba da ima sljedeći interfejs:

```
AbstractInterpolator(const std::vector<std::pair<double, double>> &data);
virtual double operator()(double x) const = 0;
```

Konstruktor kao parametar prima vektor uređenih parova realnih brojeva koji predstavljaju tačke (x_i, y_i) na osnovu kojih se vrši interpolacija. Kako većina metoda za interpolaciju (mada ne i sve) zahtijevaju da tačke moraju biti sortirane po x-koordinati, a vektor "`data`" ne mora biti takav, ono što konstruktor treba uraditi je da pohrani u neki vektor ulazne podatke sortirane po x-koordinati (za tu svrhu, koristite funkciju "`sort`" iz biblioteke "`algorithm`" uz prikladnu funkciju kriterija, koju možete izvesti kao lambda funkciju ili kao privatnu statičku funkciju članicu). Sortiranje doduše nije potrebno za polinomsku ili trigonometrijsku interpolaciju, ali od njega neće biti ni velike štete. Ukoliko postoji više tačaka koji imaju jednake x-koordinate, potrebno je baciti izuzetak tipa "`domain_error`" uz prateći tekst "Invalid data set". Pored konstruktora, jedino što još sadrži interfejs ove klase je operatorska funkcija za operator "`()`", koja vrši izračunavanje vrijednosti interpolacije u zadanoj tački. Ova funkcija je apstraktna (čisto virtualna) funkcija, s obzirom da će njene konkretne realizacije biti implementirane tek u izvedenim klasama koje definiraju konkretne metode interpolacije.

Pored konstruktora i apstraktne operatorske funkcije za operator "`()`", ova klasa treba da sadrži i funkciju članicu s prototipom

```
int Locate(double x) const;
```

Međutim, ova funkcija se neće nalaziti u interfejsu klase (tj. njenom javnom dijelu), nego u zaštićenom (tj. "**`protected`**") dijelu. Stoga će ovu funkciju moći samo interno koristiti funkcije članice klasa koje su izvedene iz apstraktne bazne klase "`AbstractInterpolator`", ali ne i korisnici ovih klasa. Uloga ove funkcije je da obavi jednu pomoćnu funkcionalnost koja je potrebna u mnogim metodama interpolacije. Naime, ona na osnovu argumenta x treba da postupkom binarne pretrage locira odgovarajući podinterval $(x_i, x_{i+1}]$ koji sadrži tačku x i da vrati vrijednost indeksa i kao rezultat. Pri tome, binarna pretraga se može koristiti jer su vrijednosti x_i prethodno sortirane, a za potrebe ovog zadatka, ne samo da se može, nego se i mora koristiti (za tu svrhu, od koristi Vam može biti funkcija "`lower_bound`" iz biblioteke "`algorithm`", ukoliko ne želite da implementirate binarnu pretragu "pješke"). Vodite računa da vrijednost i treba da bude vraćena u skladu s matematičkom konvencijom, prema kojoj prva tačka ima indeks 1, druga indeks 2, itd. iako će u memoriji prva tačka biti zapravo smještena u element vektora s indeksom 0, itd. Ukoliko je $x \leq x_1$ funkcija treba da vrati 0 kao rezultat, a ukoliko je $x > x_n$ gdje je n broj tačaka, funkcija treba da vrati n kao rezultat. Da bi se lociranje ubrzalo, nađenu poziciju intervala u sortiranom vektoru koji sadrži vrijednost argumenta x treba "keširati" u nekom privatnom atributu klase (koji mora biti označen sa "**`mutable`**" da bi se mogao mijenjati iz "**`const`**" funkcije), tako da kada se funkcija "`Locate`" bude pozivala ponovo za novu vrijednost x, treba prvo provjeriti da li se nova vrijednost x možda nalazi u istom intervalu u kojem se nalazila i prethodna, ili eventualno možda u

prvom sljedećem ili prvom prethodnom intervalu. Tek ukoliko to nije slučaj, treba ići na primjenu binarne pretrage za lociranje odgovarajućeg intervala.

S obzirom da je klasa "`AbstractInterpolator`" bazna klasa za sve ostale klase koje trebate razviti, nemojte ići dalje dok ne implementirate kako treba tražene funcionalnosti u ovoj klasi. Prilikom testiranja, učinite operatorsku funkciju za operator "`()`" konkretnom tako što ćete joj staviti neku bezveznu implementaciju (npr. "`return 0;`"), inače nećete moći kreirati primjerke ove klase. Isto tako, za potrebe testiranja prebacite prototip funkcije "`Locate`" u javni dio klase, jer je u suprotnom nećete moći pozivati iz testnog programa. Kada sve istestirate i uvjerite se da klasa "`AbstractInterpolator`" radi svoju funkcionalnost kako treba, vratite sve onako kako treba da bude.

Klasa "`LinearInterpolator`" nasljeđuje se iz bazne klase "`AbstractInterpolator`", a treba da ima sljedeći interfejs:

```
LinearInterpolator(const std::vector<std::pair<double, double>> &data);
double operator()(double x) const override;
```

Kao i kod svih ostalih klasa koje treba implementirati, parametar "`data`" je vektor uređenih parova koji predstavljaju tačke (x_i, y_i) na osnovu kojih se vrši interpolacija. U suštini, konstruktor ove klase ne treba uraditi ništa što ne radi konstruktor bazne klase, tako da je dovoljno da se parametar samo proslijedi konstruktoru bazne klase na obradu (sortiranje i detekciju duplikata). Operatorska funkcija za operator "`()`" vrši izračunavanje vrijednosti interpolacije u zadanoj tački. Pri tome, pozivom funkcije "`Locate`" treba locirati odgovarajući podinterval $(x_i, x_{i+1}]$ koji sadrži tačku x, a zatim primijeniti odgovarajući izraz koji vrijedi za taj interval. Ukoliko je $x \leq x_1$ ili $x > x_n$, proračun treba obaviti na osnovu izraza koji vrijedi za prvi odnosno posljednji interval, tj. koji vrijedi u intervalu (x_1, x_2) odnosno (x_{n-1}, x_n), čime se zapravo obavlja linearna ekstrapolacija.

Klasa "`PolynomialInterpolator`" se također nasljeđuje iz bazne klase "`AbstractInterpolator`", a treba da ima sljedeći interfejs:

```
PolynomialInterpolator(const std::vector<std::pair<double, double>> &data);
double operator()(double x) const override;
void AddPoint(const std::pair<double, double> &p);
std::vector<double> GetCoefficients() const;
```

Konstruktor ove klase također kao parametar prima vektor uređenih parova koji predstavljaju tačke (x_i, y_i) na osnovu kojih se vrši interpolacija, tj. tačke kroz koje interpolacioni polinom treba da prođe. Kako konstruktor izvedene klase mora pozvati konstruktor bazne klase, to se i ovdje mora uraditi, mada će konstruktor bazne klase obaviti sortiranje podataka koje za ovu vrstu interpolacije nije potrebno. Dobra stvar je što će se tom prilikom provjeriti i da li ima tačaka sa jednakim x koordinatama i baciti odgovarajući izuzetak ukoliko ima. Za potrebe ove klase, interpolacioni polinom treba interno čuvati kao vektor koeficijenata Newtonove interpolacione formule, odnosno za računanje interpolacije koristiće se Newtonova formula. Odgovarajući koeficijenti za Newtonovu interpolacionu formulu proračunavaju se unutar konstruktora, koristeći pristup zasnovan na trougaonoj šemi formiranoj od podijeljenih razlika (pri tome će biti potrebno čuvati još neke informacije radi efikasne podrške funkciji "`AddPoint`", o čemu ćemo govoriti malo kasnije). Operatorska funkcija za operator "`()`" vrši izračunavanje interpolacionog polinoma u zadanoj tački pomoću Newtonove formule (u linearnom vremenu), koristeći koeficijente koji su prethodno izračunati u konstruktoru i pohranjeni negdje u internim atributima klase. Funkcija "`AddPoint`" dodaje novu tačku u skup tačaka na osnovu kojeg se vrši interpolacija. Nakon dodavanja nove tačke, treba izračunati novi koeficijent za Newtonovu interpolacionu formulu (dotadašnji koeficijenti moraju ostati isti). Da bi se ovo moglo obaviti a da se cijeli račun ne ponavlja ispočetka, prilikom prvog proračunavanja koeficijenata (koje se vrši u konstruktoru) potrebno je zapamtiti i neophodne informacije koje će kasnije biti potrebne za proračunavanje novog koeficijenta. U slučaju da je tačka koju dodajemo ista kao neka od tačaka koje su već pohranjene, funkcija treba baciti izuzetak tipa "`domain_error`" uz prateći tekst "Invalid point". Konačno, funkcija "`GetCoefficients`" vraća vektor koeficijenata interpolacionog polinoma (vodite računa da to nisu koeficijenti koji se javljaju u Newtonovoj interpolacionoj formuli, nego koeficijenti koji stoje uz stepene promjenljive x u analitičkom zapisu polinoma). Za proračun ovih koeficijenata treba koristiti algoritam zasnovan na upotrebi master polinoma koji radi u vremenu $O(n^2)$.

Sljedeća klasa koju treba naslijediti iz apstraktne bazne klase "`AbstractInterpolator`" je klasa "`PiecewisePolynomialInterpolator`", a treba da ima sljedeći interfejs:

```
PiecewisePolynomialInterpolator(const std::vector<std::pair<double, double>> &data,
  int order);
double operator()(double x) const override;
```

Ova klasa je veoma slična klasi "`LinearInterpolator`", samo što umjesto linearne interpolacije vrši interpolaciju polinonima dio po dio. Konstruktor ove klase, pored vektora uređenih parova koji predstavljaju tačke (x_i, y_i) na osnovu kojih se vrši interpolacija, prima još jedan parametar koji predstavlja red interpolacije k, odnosno red (stepen) polinoma pomoću kojih se vrši interpolacija između čvorova. U tom smislu, klasa "`LinearInterpolator`" može da se shvati i kao specijalan slučaj ove klase kod koje se koriste polinomi prvog reda (tj. za $k = 1$). Red k mora biti veći ili jednak od 1 kao i strogo manji od broja čvorova, u suprotnom treba baciti izuzetak tipa "`domain_error`" uz prateći tekst "Invalid order". Unutar operatorske funkcije za operator "`()`", prvo treba pozivom funkcije "`Locate`" locirati odgovarajući podinterval $(x_i, x_{i+1}]$ koji sadrži tačku x, a nakon toga se vrijednost interpolacije u tački x određuje na osnovu polinoma zadanog reda koji prolazi ne samo kroz čvorove (x_i, y_i) i (x_{i+1}, y_{i+1}), nego i još nekoliko čvorova s lijeve i desne strane (tačan način odabira čvorova opisan je na predavanjima). Sâmu vrijednost polinoma u tački x treba računati pomoću Lagrangeove formule (samo sa izmijenjenim granicama u sumi i produktu, jer u igri neće biti svi čvorovi, već samo k čvorova u okolici tačke x). U slučaju da je $x \leq x_1$ ili $x > x_n$ (ekstrapolacija), proračun treba obaviti na osnovu izraza koji vrijedi za prvi odnosno posljednji interval, tj. koji vrijedi u intervalu (x_1, x_2) odnosno (x_{n-1}, x_n).

Sljedeća klasa naslijeđena iz bazne klase "`AbstractInterpolator`" je klasa "`SplineInterpolator`", koja treba da ima sljedeći interfejs:

```
SplineInterpolator(const std::vector<std::pair<double, double>> &data);
double operator()(double x) const override;
```

Kao što je uobičajeno, konstruktor ove klase kao parametar prima vektor uređenih parova koji predstavljaju tačke (x_i, y_i) na osnovu kojih se vrši interpolacija, tj. tačke kroz koje kubni splajn treba da prođe. Slično kao kod linearne interpolacije odnosno interpolacije polinomima dio po dio, interpolacija splajnovima također zahtijeva da tačke moraju biti sortirane po x-koordinati, što će svakako obaviti konstruktor bazne klase (uz provjeru da li možda ima tačaka s jednakim x-koordinatama i bacanje odgovarajućih izuzetaka ako ima). Nakon toga, konstrukor ove klase treba da obavi konstrukciju splajna, odnosno da izvrši proračun odgovarajućih koeficijenata koji tvore splajn i da ih pohrani negdje unutar klase. Pretpostavite da je u pitanju prirodni splajn, odnosno za granične uvjete uzmite da su drugi izvodi u krajnjim tačkama jednaki nuli. Operatorska funkcija za operator "`()`" vrši izračunavanje splajna u zadanoj tački. Slično kao u prethodne dvije klase, prvo treba pozivom funkcije "`Locate`" iz bazne klase locirati odgovarajući podinterval $(x_i, x_{i+1}]$ koji sadrži tačku x, a zatim primijeniti odgovarajući izraz koji vrijedi za taj interval. Ukoliko je $x \leq x_1$ ili $x > x_n$, proračun treba obaviti na osnovu polinoma koji vrijedi u intervalu $(x_1, x_2]$ odnosno $(x_{n-1}, x_n]$.

Sljedeća klasa koja se također nasljeđuje iz apstraktne bazne klase "`AbstractInterpolator`" je klasa "`BarycentricInterpolator`". Ona treba da ima sljedeći interfejs:

```
BarycentricInterpolator(const std::vector<std::pair<double, double>> &data, int order);
double operator()(double x) const override;
std::vector<double> GetWeights() const;
```

Slično kao kod klase "`PiecewisePolynomialInterpolator`", konstruktor ove klase pored vektora uređenih parova koji predstavljaju tačke (x_i, y_i) na osnovu kojih se vrši interpolacija zahtijeva i dodatni cjelobrojni parametar koji određuje red baricentrične interpolacije. Ovaj parametar mora biti u opsegu od 0 do broja tačaka uključivo, Ukoliko to nije slučaj, treba baciti izuzetak tipa "`domain_error`" uz prateći tekst "Invalid order". Pored sortiranja i provjere da li možda ima tačaka sa jednakim x koordinatama, što će uraditi konstruktor bazne klase, unutar konstruktora ove klase treba obaviti i proračun težinskih koeficijenata za baricentričnu racionalnu interpolaciju, koji se potom čuvaju u nekom internom atributu klase. Kao i obično, operatorska funkcija za operator "`()`" vrši izračunavanje vrijednosti interpolacije u zadanoj tački. Treba podržati i funkciju "`GetWeights`" pomoću koje se može dobiti vektor koji sadrži vrijednosti proračunatih težinskih koeficijenata. Kako su težinski koeficijenti indeksirani od jedinice

($w_i, i = 1, 2, \ldots, n$) a ova funkcija vraća klasični C++-ovski vektor indeksiran od nule, koeficijente w_i, $i = 1, 2, \ldots, n$ treba smjestiti u elemente vraćenog vektora sa indeksima $0, 1, \ldots, n-1$ respektivno.

Posljednja iz porodice klasa koje se naslijeđuju iz apstraktne bazne klase "`AbstractInterpolator`" je klasa "`TrigonometricInterpolator`". Ona treba da ima sljedeći interfejs:

```
TrigonometricInterpolator(const std::vector<std::pair<double, double>> &data);
double operator()(double x) const override;
```

Kao i kod svih dosadašnjih klasa, konstruktor ove klase kao parametar prima vektor uređenih parova koji predstavljaju tačke (x_i, y_i) tačke kroz koje interpolacioni trigonometrijski polinom treba da prođe. Slično kao kod klase "`PolynomialInterpolator`", konstruktor bazne klase (čije je pozivanje neophodno) i ovdje će obaviti sortiranje podataka koje za ovu vrstu interpolacije nije neophodno, uz pozitivan prateći efekat provjere postojanja tačaka sa jednakim x koordinatama praćen bacanjem odgovarajućeg izuzetka ukoliko takvih tačaka ima. Međutim, nakon obavljenog sortiranja, treba još dodatno testirati da li je vrijednost funkcije u prvom čvoru interpolacije jednaka vrijednosti funkcije u posljednjem čvoru (što bi trebalo biti, ukoliko interpoliramo periodičnu funkciju čiji je period jednak širini intervala na kojem se vrši interpolacija). Ukoliko to nije ispunjeno, teba baciti izuzetak tipa "`domain_error`" uz prateći tekst "Function is not periodic". Pored konstruktora, ova klasa posjeduje još samo operatorsku funkciju za operator "`()`", koja vrši računanje vrijednosti interpolacije u zadanoj tački. Računanje treba vršiti pomoću Gaussove ili modificirane Gaussove formule za trigonometrijsku interpolaciju, ovisno od toga da li je broj čvorova interpolacije paran ili neparan (činjenica je da ove formule imaju vrijeme izvršavanja reda $O(n^2)$ i da postoje formule Newtonovog tipa čije je vrijeme izvršavanja reda $O(n)$, ali za potrebe ovog zadatka u to nećemo ulaziti). U slučaju neparnog broja čvorova, treba koristiti varijantu formule koja obezbjeđuje da posljednji član trigonometrijskog polinoma ima fazu 0, tj. da ne sadrži funkciju sinus.

Obavezno napišite i testni glavni program u kojem ćete testirati napisane klase. Pri tome, klasu "`PolynomialInterpolator`" najbolje ćete testirati tako što ćete uzeti testne podatke uzete sa neke funkcije koja zaista jeste polinom (interpolacioni polinom tada treba da bude upravo taj polinom). Za testiranje klase "`SplineInterpolator`" konstruirajte splajn na osnovu uzoraka neke poznate i relativno pravilne funkcije, recimo $f(x) = \sin x$ u tačkama koje nisu previše razmaknute, i uporedite vrijednosti koje se dobiju interpolacijom u nekim međutačkama sa pravom vrijednošću funkcije. Vrijednosti bi trebale biti posve bliske. Slično možete postupiti i za klase "`LinearInterpolator`", "`PiecewisePolynomialInterpolator`" odnosno "`BarycentricInterpolator`", jedino što u slučaju linearne interpolacije uzorci moraju biti znatno gušći za postizanje dobre tačnosti. Konačno, testiranje klase "`TrigonometricInterpolator`" najbolje ćete testirati tako što ćete uzeti testne podatke uzete sa neke funkcije koja zaista jeste trigonometrijski polinom.

Zadatak 2 (1 poen)

U ovom zadatku, potrebno je napraviti globalnu funkciju "`Limit`" za numeričko nalaženje graničnih vrijednosti postupkom Richardsonove ekstrapolacije. Ova funkcija treba da ima sljedeći prototip:

```
template <typename FunType>
  std::pair<double, bool> Limit(FunType f, double x0, double h = 0,
    double eps = 1e-8, double nmax = 20);
```

Funkcija treba da vrati kao rezultat numerički izračunatu vrijednost granične vrijednosti funkcije f zadane prvim parametrom kada x teži ka x_0, gdje se x_0 zadaje kao drugi parametar. Funkcija je napisana kao generička, da bi se omogućilo da prvi parametar može biti ne samo funkcija, nego i bilo šta što se može koristiti poput funkcije, npr. neki funkcijski objekat (alternativno bi prvi parametar mogao biti i polimorfni funkcijski omotač tipa "`std::function<double(double)>`", ali su generičke funkcije za ovu svrhu efikasnije). Računanje treba obaviti Richardsonovim postupkom ekstrapolacije ka granici, koristeći Nevilleov algoritam. Treći parametar predstavlja početnu vrijednost koraka h. Ova vrijednost može biti kako pozitivna, tako i negativna (negativna vrijednost se uzima ukoliko želimo računati limes slijeva). Ukoliko se kao h zada nula (primijetimo da je besmisleno da korak zaista bude nula), korak se automatski određuje kao $0.001 \cdot \max\{1, |x_0|\}$, što se u praksi pokazalo kao vrlo dobra vrijednost koraka. Primijetimo da je podrazumijevana vrijednost trećeg parametra upravo nula, tako da će se u slučaju da se ovaj parametar izostavi, koristiti automatsko biranje koraka. Četvri parametar je zadana toleranciju ε (pri čemu je podrazumijevana vrijednost 10^{-8}). Račun se prekida ukoliko se postigne da u

dvije susjedne iteracije modul razlike proračunatih vrijednosti postane manji od ε. Konačno, posljednji parametar je maksimalno dozvoljeni broj iteracija $n_{\max}$. Računanje se prekida kada se premaši ovaj broj iteracija, čak i ukoliko pri tome nije postignuta odgovarajuća tačnost. Kao rezultat funkcije, vraća se uređeni par čija je prva koordinata procijenjena vrijednost limesa, a druga koordinata logička vrijednost "tačno" ili "netačno", ovisno da li je tražena tačnost postignuta ili ne. Parametar ε mora biti pozitivan, dok parametar $n_{\max}$ mora biti između 3 i 30 uključivo. U suprotnom, treba baciti izuzetak tipa "`domain_error`" uz prateći tekst "Invalid parameters".

Funkcija "`Limit`" treba podržavati i mogućnost da parametar x_0 ima vrijednost ∞ ili $-\infty$. Podsjetimo se da se granična vrijednost funkcije $f(x)$ kada x teži ka ∞ ili $-\infty$ može dobiti kao granična vrijednost kad x teži nuli zdesna ili slijeva funkcije $f(1/x)$. Isto tako, podsjetimo se da je legalan način da se zada vrijednost ∞ pomoću konstrukcije "`std::numeric_limits<double>::infinity()`", mada postoje i neki drugi manje rogobatni načini (vjerovatno je konstrukcija "`1 / 0.`" najkraći način da se to postigne), ali svi ti alternativni načini imaju i svoje nuspojave (recimo, konstrukcija "`1 / 0.`" će vjerovatno dovesti do prijave upozorenja od strane kompajlera, a također će postaviti indikator da je došlo do dijeljenja s nulom, što obično ne smeta, ali može u nekim kontekstima smetati).

Obavezno napišite i testni glavni program u kojem ćete testirati napisanu funkciju. Funkciju ćete najbolje testirati na nekoliko tipičnih limesa, čija vam je vrijednost poznata.

NAPOMENA:

Naučite se da testirate programe sami, nemojte čekati na autotestove da vidite da li Vam je nešto ispravno ili ne. Pregledanje zadaće će se vršiti kombinacijom autotestova i ručnog pregledanja. Za metodu, funkciju ili operator koji nisu temeljito testirani u glavnom programu smatraće se da nisu urađeni, te se za njih neće dobiti nikakvi bodovi!