

Міністерство освіти і науки України
Національний університет «Львівська політехніка»
кафедра інформаційних систем та мереж

Григорович Віктор

Об'єктно-орієнтоване програмування

Успадкування

Навчальний посібник

2021

Григорович Віктор Геннадійович

Об'єктно-орієнтоване програмування. Успадкування. Навчальний посібник.

Дисципліна «Об'єктно-орієнтоване програмування» вивчається після курсу «Алгоритмізація та програмування», цією дисципліною продовжується цикл предметів, що стосуються програмування та розробки програмного забезпечення.

В посібнику містяться теоретичні відомості, приклади, методичні вказівки з їх розв'язування, варіанти лабораторних завдань та питання і завдання з контролю знань з теми «Успадкування».

Розглядаються наступні роботи лабораторного практикуму:

Лабораторна робота № 3.1.

Відкрите успадкування

Лабораторна робота № 3.2.

Просте успадкування

Лабораторна робота № 3.3.

Успадкування замість композиції

Лабораторна робота № 3.4.

Демонстрація простого успадкування

Лабораторна робота № 3.5.

Масиви та успадкування

Лабораторна робота № 3.6.

Множинне успадкування

Відповідальний за випуск – Григорович В.Г.

Стислий зміст

Вступ.....	21
Тема 3. Успадкування	22
Стисло та головне про успадкування	22
Просте успадкування	23
Множинне успадкування.....	33
Теоретичні відомості.....	35
Проектування ієрархії класів	37
Успадкування та композиція.....	39
Просте успадкування	40
Множинне успадкування.....	69
Явні перетворення типів в мові C++	80
Лабораторний практикум	88
Оформлення звіту про виконання лабораторних робіт	88
Лабораторна робота № 3.1. Відкрите успадкування.....	90
Лабораторна робота № 3.2. Просте успадкування.....	105
Лабораторна робота № 3.3. Успадкування замість композиції	113
Лабораторна робота № 3.4. Демонстрація простого успадкування	281
Лабораторна робота № 3.5. Масиви та успадкування	285
Лабораторна робота № 3.6. Множинне успадкування	300
Питання та завдання для контролю знань	318
Просте успадкування	318
Множинне успадкування.....	319
Предметний покажчик	323
Література	324

Зміст

Вступ.....	21
Тема 3. Успадкування	22
Стисло та головне про успадкування	22
Просте успадкування	23
Ключі успадкування та директиви доступу	24
Директиви доступу <code>private</code> , <code>protected</code> та <code>public</code>	24
Просте відкрите успадкування.....	24
Конструктори і деструктори при успадкуванні	25
Конструктори.....	25
Деструктори	25
Поля та методи при успадкуванні	25
Поля	26
Методи.....	27
Статичні елементи класу при успадкуванні	28
Вкладені класи при успадкуванні	28
Принцип підстановки, операція присвоєння та розщеплення	28
Принцип підстановки.....	28
Операція присвоєння	29
Розщеплення та зрізування.....	29
Дружні функції при успадкуванні	30
Відкрите успадкування – успадкування інтерфейсу	30
Закрите успадкування. Делегування	30
Закрите успадкування та композиція	32
Закрите успадкування – успадкування реалізації	33
Відмінності структур та об'єднань від класів	33
Множинне успадкування.....	33
Теоретичні відомості.....	35
Проектування ієрархії класів	37
Послідовність дій	37
Приклад	37
Проектування класів	38
Правило проектування класів	39
Правило проектування методів.....	39
Успадкування та композиція.....	39

Просте успадкування	40
Ключі успадкування та директиви доступу	40
Директиви доступу <code>private</code> , <code>protected</code> та <code>public</code>	40
Ключі успадкування <code>private</code> , <code>protected</code> та <code>public</code>	41
Доступ до елементів класу при різних ключах успадкування	41
Просте відкрите успадкування	45
Конструктори і деструктори при успадкуванні	45
Конструктори	45
Деструктори	46
Поля та методи при успадкуванні	46
Поля	47
Методи	48
Статичні елементи класу при успадкуванні	54
Статичні поля	54
Статичні методи	55
Вкладені класи при успадкуванні	55
Принцип підстановки, операція присвоєння та розщеплення	56
Принцип підстановки	56
Операція присвоєння	57
Розщеплення та зрізування	59
Дружні функції при успадкуванні	59
Відкрите успадкування – успадкування інтерфейсу	60
Закрите успадкування. Делегування	60
Закрите успадкування та композиція	62
Закрите успадкування – успадкування реалізації	62
Патерн Adapter (адаптер об'єктів)	63
Відмінності структур та об'єднань від класів	69
Множинне успадкування	69
Неоднозначність	70
Патерн Adapter (адаптер класів)	73
Віртуальне успадкування	74
Принцип домінування	77
Фінальний клас	78
Розміри класів при множинному успадкуванні	79
Явні перетворення типів в мові C++	80

Перетворення <code>const_cast<></code>	80
Перетворення <code>static_cast<></code>	81
Перетворення <code>reinterpret_cast<></code>	81
Перетворення типів споріднених класів ієрархії <code>dynamic_cast<></code>	82
Операція <code>dynamic_cast<></code>	82
Перетворення, що підвищує рівень ієрархії	82
Перетворення, що понижує рівень ієрархії	83
Перетворення посилань	84
Перехресне перетворення	85
Операція <code>static_cast<></code>	87
Лабораторний практикум	88
Оформлення звіту про виконання лабораторних робіт	88
Вимоги до оформлення звіту про виконання лабораторних робіт №№ 3.1–3.6	88
Зразок оформлення звіту про виконання лабораторних робіт №№ 3.1–3.6	89
Лабораторна робота № 3.1. Відкрите успадкування	90
Мета роботи	90
Питання, які необхідно вивчити та пояснити на захисті	90
Зразок виконання завдання	90
Варіант 0	90
Умова завдання	90
Текст програми	91
Варіанти завдань	96
Варіант 1	96
Варіант 2	97
Варіант 3	97
Варіант 4	97
Варіант 5	97
Варіант 6	97
Варіант 7	98
Варіант 8	98
Варіант 9	98
Варіант 10	98
Варіант 11	98
Варіант 12	98
Варіант 13	99

Варіант 14.....	99
Варіант 15.....	99
Варіант 16.....	99
Варіант 17.....	99
Варіант 18.*	100
Варіант 19.....	100
Варіант 20.....	100
Варіант 21.....	100
Варіант 22.....	101
Варіант 23.....	101
Варіант 24.....	101
Варіант 25.....	101
Варіант 26.....	101
Варіант 27.....	101
Варіант 28.....	102
Варіант 29.....	102
Варіант 30.....	102
Варіант 31.....	102
Варіант 32.....	102
Варіант 33.....	102
Варіант 34.....	103
Варіант 35.....	103
Варіант 36.....	103
Варіант 37.....	103
Варіант 38.....	103
Варіант 39.....	104
Варіант 40.....	104
Лабораторна робота № 3.2. Просте успадкування	105
Мета роботи	105
Питання, які необхідно вивчити та пояснити на захисті.....	105
Варіанти завдань	105
Варіант 1.*	105
Варіант 2.....	106
Варіант 3.*	106
Варіант 4.....	106

Варіант 5.....	106
Варіант 6.*.....	106
Варіант 7.....	106
Варіант 8.....	107
Варіант 9.....	107
Варіант 10.....	107
Варіант 11.....	107
Варіант 12.....	107
Варіант 13.....	107
Варіант 14.....	107
Варіант 15.....	108
Варіант 16.....	108
Варіант 17.....	108
Варіант 18.....	108
Варіант 19.*.....	108
Варіант 20.*.....	108
Варіант 21.*.....	109
Варіант 22.*.....	109
Варіант 23.***.....	109
Варіант 24.***.....	109
Варіант 25.***.....	109
Варіант 26.*.....	109
Варіант 27.....	110
Варіант 28.*.....	110
Варіант 29.....	110
Варіант 30.....	110
Варіант 31.*.....	110
Варіант 32.....	111
Варіант 33.....	111
Варіант 34.....	111
Варіант 35.....	111
Варіант 36.....	111
Варіант 37.....	111
Варіант 38.....	111
Варіант 39.....	112

Варіант 40.....	112
Лабораторна робота № 3.3. Успадкування замість композиції	113
Мета роботи	113
Питання, які необхідно вивчити та пояснити на захисті.....	113
Зразок виконання завдання	113
Завдання Е.....	113
Умова завдання.....	113
Текст програми	115
Варіанти завдань	121
Завдання А	121
Варіант 1.....	122
Варіант 2.....	123
Варіант 3.....	123
Варіант 4.....	123
Варіант 5.*.....	123
Варіант 6.....	125
Варіант 7.....	126
Варіант 8.....	127
Варіант 9.....	127
Варіант 10.*.....	128
Варіант 11.....	128
Варіант 12.....	128
Варіант 13.....	128
Варіант 14.....	129
Варіант 15.....	129
Варіант 16.*.....	129
Варіант 17.....	131
Варіант 18.....	132
Варіант 19.....	133
Варіант 20.*.....	133
Варіант 21.*.....	134
Варіант 22.....	134
Варіант 23.....	134
Варіант 24.....	134
Варіант 25.....	135

Варіант 26.....	135
Варіант 27.....	135
Варіант 28.....	136
Варіант 29.....	136
Варіант 30.*.....	136
Варіант 31.....	138
Варіант 32.....	139
Варіант 33.....	140
Варіант 34.....	140
Варіант 35.*.....	140
Варіант 36.....	141
Варіант 37.....	141
Варіант 38.....	141
Варіант 39.....	142
Варіант 40.....	142
Завдання В.....	143
Варіант 1.....	144
Варіант 2.....	144
Варіант 3.....	144
Варіант 4.....	145
Варіант 5.*.....	145
Варіант 6.....	146
Варіант 7.....	148
Варіант 8.....	148
Варіант 9.....	149
Варіант 10.*.....	149
Варіант 11.....	149
Варіант 12.....	150
Варіант 13.....	150
Варіант 14.....	150
Варіант 15.....	151
Варіант 16.*.....	151
Варіант 17.....	152
Варіант 18.....	154
Варіант 19.....	155

Варіант 20.*.....	155
Варіант 21.*.....	155
Варіант 22.....	155
Варіант 23.....	156
Варіант 24.....	156
Варіант 25.....	156
Варіант 26.....	157
Варіант 27.....	157
Варіант 28.....	157
Варіант 29.....	158
Варіант 30.*.....	158
Варіант 31.....	159
Варіант 32.....	161
Варіант 33.....	161
Варіант 34.....	162
Варіант 35.*.....	162
Варіант 36.....	162
Варіант 37.....	163
Варіант 38.....	163
Варіант 39.....	163
Варіант 40.....	163
Завдання С.....	165
Варіант 1.....	166
Варіант 2.....	166
Варіант 3.....	167
Варіант 4.....	167
Варіант 5.*.....	167
Варіант 6.....	169
Варіант 7.....	170
Варіант 8.....	171
Варіант 9.....	171
Варіант 10.*.....	171
Варіант 11.....	172
Варіант 12.....	172
Варіант 13.....	172

Варіант 14.....	173
Варіант 15.....	173
Варіант 16.*.....	173
Варіант 17.....	175
Варіант 18.....	176
Варіант 19.....	177
Варіант 20.*.....	177
Варіант 21.*.....	177
Варіант 22.....	178
Варіант 23.....	178
Варіант 24.....	178
Варіант 25.....	179
Варіант 26.....	179
Варіант 27.....	179
Варіант 28.....	180
Варіант 29.....	180
Варіант 30.*.....	180
Варіант 31.....	182
Варіант 32.....	183
Варіант 33.....	184
Варіант 34.....	184
Варіант 35.*.....	184
Варіант 36.....	185
Варіант 37.....	185
Варіант 38.....	185
Варіант 39.....	186
Варіант 40.....	186
Завдання D	187
Варіант 1.....	188
Варіант 2.....	188
Варіант 3.....	188
Варіант 4.....	188
Варіант 5.....	189
Варіант 6.....	189
Варіант 7.....	189

Варіант 8.....	189
Варіант 9.....	189
Варіант 10.....	189
Варіант 11.....	190
Варіант 12.....	190
Варіант 13.....	190
Варіант 14.....	190
Варіант 15.....	190
Варіант 16.....	190
Варіант 17.....	191
Варіант 18.*.....	191
Варіант 19.....	191
Варіант 20.....	191
Варіант 21.....	192
Варіант 22.....	192
Варіант 23.....	192
Варіант 24.....	192
Варіант 25.....	192
Варіант 26.....	192
Варіант 27.....	193
Варіант 28.....	193
Варіант 29.....	193
Варіант 30.....	193
Варіант 31.....	193
Варіант 32.....	194
Варіант 33.....	194
Варіант 34.....	194
Варіант 35.....	194
Варіант 36.....	194
Варіант 37.....	194
Варіант 38.....	195
Варіант 39.....	195
Варіант 40.....	195
Завдання Е.....	196
Варіант 1.....	197

Варіант 2.....	197
Варіант 3.....	197
Варіант 4.....	197
Варіант 5.....	198
Варіант 6.....	198
Варіант 7.....	198
Варіант 8.....	198
Варіант 9.....	198
Варіант 10.....	198
Варіант 11.....	199
Варіант 12.....	199
Варіант 13.....	199
Варіант 14.....	199
Варіант 15.....	199
Варіант 16.....	199
Варіант 17.....	200
Варіант 18.*.....	200
Варіант 19.....	200
Варіант 20.....	200
Варіант 21.....	201
Варіант 22.....	201
Варіант 23.....	201
Варіант 24.....	201
Варіант 25.....	201
Варіант 26.....	201
Варіант 27.....	202
Варіант 28.....	202
Варіант 29.....	202
Варіант 30.....	202
Варіант 31.....	202
Варіант 32.....	203
Варіант 33.....	203
Варіант 34.....	203
Варіант 35.....	203
Варіант 36.....	203

Варіант 37.....	203
Варіант 38.....	204
Варіант 39.....	204
Варіант 40.....	204
Завдання F	205
Варіант 1.....	206
Варіант 2.....	207
Варіант 3.....	208
Варіант 4.....	209
Варіант 5*.....	210
Варіант 6*.....	211
Варіант 7.....	211
Варіант 8.....	212
Варіант 9.....	213
Варіант 10.....	214
Варіант 11.....	215
Варіант 12.....	216
Варіант 13*.....	217
Варіант 14*.....	217
Варіант 15*.....	218
Варіант 16*.....	219
Варіант 17.*.....	219
Варіант 18.*.....	221
Варіант 19.*.....	222
Варіант 20.*.....	223
Варіант 21.....	224
Варіант 22.....	225
Варіант 23.....	226
Варіант 24.....	227
Варіант 25*.....	228
Варіант 26*.....	229
Варіант 27.....	229
Варіант 28.....	230
Варіант 29.....	231
Варіант 30.....	232

Варіант 31.....	233
Варіант 32.....	234
Варіант 33*.....	235
Варіант 34*.....	235
Варіант 35*.....	236
Варіант 36*.....	237
Варіант 37.*.....	238
Варіант 38.*.....	239
Варіант 39.*.....	240
Варіант 40.*.....	241
Завдання G	243
Варіант 1.....	244
Варіант 2.....	245
Варіант 3.....	246
Варіант 4.....	247
Варіант 5*.....	248
Варіант 6*.....	249
Варіант 7.....	249
Варіант 8.....	250
Варіант 9.....	251
Варіант 10.....	252
Варіант 11.....	253
Варіант 12.....	254
Варіант 13*.....	255
Варіант 14*.....	255
Варіант 15*.....	256
Варіант 16*.....	257
Варіант 17.*.....	258
Варіант 18.*.....	259
Варіант 19.*.....	260
Варіант 20.*.....	261
Варіант 21.....	262
Варіант 22.....	263
Варіант 23.....	264
Варіант 24.....	265

Варіант 25*.....	266
Варіант 26*.....	267
Варіант 27.....	267
Варіант 28.....	268
Варіант 29.....	269
Варіант 30.....	270
Варіант 31.....	271
Варіант 32.....	272
Варіант 33*.....	273
Варіант 34*.....	274
Варіант 35*.....	274
Варіант 36*.....	275
Варіант 37.*.....	276
Варіант 38.*.....	277
Варіант 39.*.....	278
Варіант 40.*.....	279
Лабораторна робота № 3.4. Демонстрація простого успадкування	281
Мета роботи	281
Питання, які необхідно вивчити та пояснити на захисті.....	281
Варіанти завдань	281
Загальна частина завдань для варіантів 1-20	281
Варіанти 1-24	281
Варіанти 25-54	283
Лабораторна робота № 3.5. Масиви та успадкування	285
Мета роботи	285
Питання, які необхідно вивчити та пояснити на захисті.....	285
Варіанти завдань	285
Варіант 1.....	286
Варіант 2.....	287
Варіант 3.....	287
Варіант 4.....	287
Варіант 5.....	287
Варіант 6.....	288
Варіант 7.....	288
Варіант 8.....	288

Варіант 9.....	288
Варіант 10.....	288
Варіант 11.....	289
Варіант 12.....	289
Варіант 13.....	289
Варіант 14.*.....	289
Варіант 15.*.....	290
Варіант 16.*.....	290
Варіант 17.*.....	291
Варіант 18.*.....	291
Варіант 19.*.....	291
Варіант 20.*.....	292
Варіант 21.*.....	293
Варіант 22.*.....	293
Варіант 23.*.....	294
Варіант 24.....	294
Варіант 25.....	294
Варіант 26.....	295
Варіант 27.....	295
Варіант 28.....	295
Варіант 29.....	295
Варіант 30.....	295
Варіант 31.....	296
Варіант 32.....	296
Варіант 33.....	296
Варіант 34.....	296
Варіант 35.....	296
Варіант 36.....	297
Варіант 37.*.....	297
Варіант 38.*.....	297
Варіант 39.*.....	298
Варіант 40.*.....	298
Лабораторна робота № 3.6. Множинне успадкування	300
Мета роботи	300
Питання, які необхідно вивчити та пояснити на захисті.....	300

Зразок виконання завдання	300
Варіант 0.....	300
Умова завдання.....	300
Текст програми	301
Варіанти завдань	304
Варіант 1.....	304
Варіант 2.....	305
Варіант 3.....	305
Варіант 4.....	305
Варіант 5.....	306
Варіант 6.....	306
Варіант 7.....	306
Варіант 8.....	307
Варіант 9.....	307
Варіант 10.....	307
Варіант 11.....	308
Варіант 12.....	308
Варіант 13.....	308
Варіант 14.....	308
Варіант 15.....	309
Варіант 16.....	309
Варіант 17.....	309
Варіант 18.....	310
Варіант 19.....	310
Варіант 20.....	310
Варіант 21.....	311
Варіант 22.....	311
Варіант 23.....	311
Варіант 24.....	311
Варіант 25.....	312
Варіант 26.....	312
Варіант 27.....	312
Варіант 28.....	313
Варіант 29.....	313
Варіант 30.....	313

Варіант 31.....	314
Варіант 32.....	314
Варіант 33.....	314
Варіант 34.....	315
Варіант 35.....	315
Варіант 36.....	315
Варіант 37.....	316
Варіант 38.....	316
Варіант 39.....	316
Варіант 40.....	317
Варіант 41.....	317
Варіант 42.....	317
Варіант 43.....	317
Питання та завдання для контролю знань	318
Просте успадкування	318
Множинне успадкування.....	319
Предметний покажчик	323
Література	324

Вступ

Дисципліна «Об'єктно-орієнтоване програмування» вивчається після курсу «Алгоритмізація та програмування», цією дисципліною продовжується цикл предметів, що стосуються програмування та розробки програмного забезпечення.

В посібнику містяться теоретичні відомості, приклади, методичні вказівки з їх розв'язування, варіанти лабораторних завдань та питання і завдання з контролю знань з теми «Успадкування».

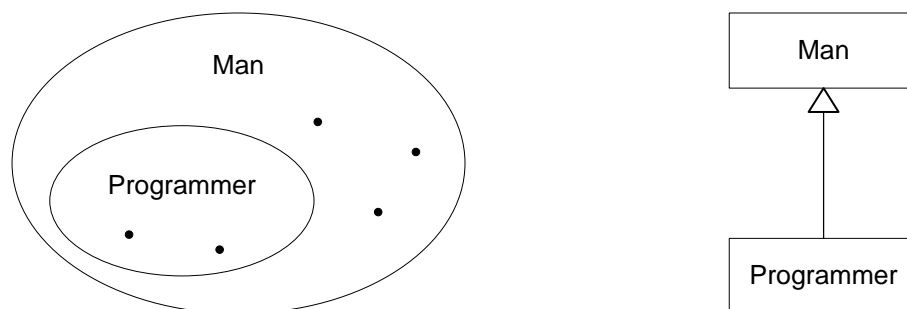
Тема 3. Успадкування

Стисло та головне про успадкування

Успадкування – один з трьох «житів» (разом з інкапсуляцією та поліморфізмом), на якому стоїть об'єктно-орієнтоване програмування. При успадкуванні обов'язково є *клас-предок* (батьківський клас; той, що породжує) та *клас-нащадок* (дочірній клас; той, який породжений). В C++ клас-предок називається *базовим*, а клас-нащадок – *похідним*. Всі терміни еквівалентні.

На UML-class diagram (UML-діаграмі класів) зв'язок успадкування (його ще називають *генералізація*) позначається трикутною незаповненою стрілкою, яка закінчує суцільну лінію, проведену від похідного класу до базового:

Наприклад, розглянемо класи **Man** (Людина) та **Programmer** (Програміст). Вони описують сукупності – множину **Люди** та множину **Програмісти**. Прийнято вважати, що кожний програміст є людиною (кодерів з Марсу та хакерів з планети Нібіру зараз не враховуємо), проте не кожна людина є програмістом: **Програмісти** – підмножина **Людей**:



На UML-діаграмі класів такий зв'язок позначаємо трикутною незаповненою стрілкою від похідного класу **Programmer** до базового класу **Man**.

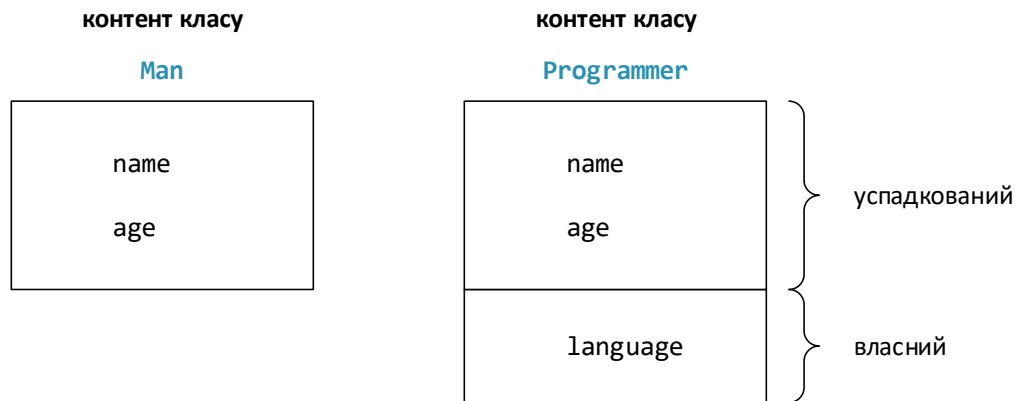
Зв'язок успадкування означає, що при визначенні похідного класу ми вказуємо лише його власні характеристики. Характеристики, спільні з базовим класом, при визначенні похідного класу вказувати не потрібно. Директива успадкування означає, що весь контент (всі характеристики) базового класу будуть присутні в похідному класі.

Термін *успадкування* (англ. *inheritance*) використовується саме тому, що похідний клас – нащадок отримує у спадщину (англ. *inheritage*) весь контент (всі характеристики) базового класу – предка.

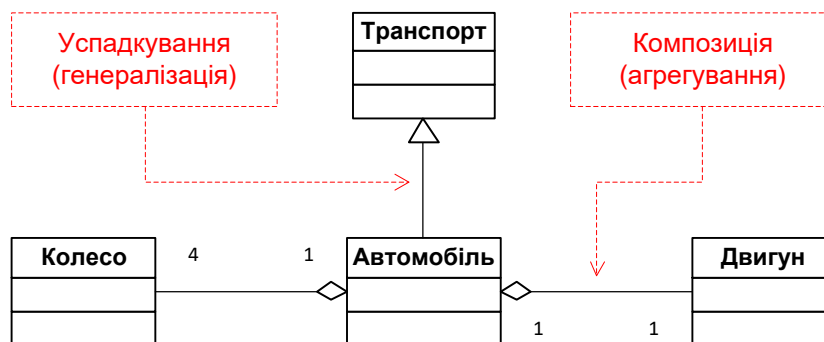
```
class Man
{
    string name;
    int age;
};

class Programmer : Man
{
    string language;
};
```

директива
успадкування



Інший приклад: класи Транспорт та Автомобіль: автомобілі є різновидом (підмножиною) транспорту. Наступна UML-діаграма показує зв'язки успадкування (транспорт – предок, автомобіль – нащадок) та агрегування (автомобіль містить 4 колеса та 1 двигун):



Відношення між батьківським класом та його нащадками називається *ієрархією успадкування*. Глибина (тобто, кількість рівнів) успадкування не обмежена.

Просте успадкування

Простим (або *одиначним*) називається успадкування, при якому похідний клас має лише одного предка. Просте успадкування визначається наступною конструкцією:

```
class ім'я_похідного_класу: [ключ_успадкування] ім'я_базового_класу
{
    тіло_похідного_класу;
};
```

ключ_успадкування — необов'язковий елемент синтаксичної конструкції (тому позначений в квадратних дужках), — це модифікатор доступу, який визначає доступність елементів базового класу в похідному класі. Квадратні дужки — не елемент синтаксису, а показують, що ключ успадкування може бути відсутнім.

Ключі успадкування та директиви доступу

При розробленні окремих класів використовуються два модифікатори доступу до елементів класу: **public** (доступні елементи) та **private** (приховані елементи). При успадкуванні використовується ще один: **protected** (захищені елементи):

```
class і'мя_класу
{
    private:
        приховані_елементи_класу;

    protected:
        захищені_елементи_класу;

    public:
        доступні_елементи_класу;
};
```

Директиви доступу private, protected та public

Директиви доступу можна вказувати в довільному порядку, їх кількість не обмежена.

Директива **private:** починає секцію прихованих елементів, які доступні *лише методам цього класу та його друзям* (дружнім класам та дружнім функціям).

Директива **protected:** починає секцію захищених елементів, які доступні *лише методам цього класу, його друзям* (дружнім класам та дружнім функціям) та *нащадкам*.

Директива **public:** починає секцію доступних елементів, які *повністю доступні поза класом* (в тому числі і методам цього класу, його друзям та його нащадкам).

Просте відкрите успадкування

Найбільш поширеним є просте відкрите успадкування. Багато об'єктно-орієнтованих мов програмування (зокрема, java, Delphi та ін.) лише його і реалізують. Тому будемо розглядати нюанси простого відкритого успадкування, а потім зазначимо відмінності для випадку закритого успадкування.

Просте відкрите успадкування описується наступною конструкцією:


```
class ім'я_похідного_класу: public ім'я_базового_класу
{
    тіло_похідного_класу;
};
```

Конструктори і деструктори при успадкуванні

Конструктори

Конструктори не успадковуються – вони створюються в похідному класі (якщо не визначені програмістом явно). Конструктори опрацьовуються наступним чином:

- якщо в базовому класі немає явно визначених конструкторів або є конструктор без аргументів (або аргументи присвоюються за умовчанням), то в похідному класі конструктор можна не писати – автоматично будуть створені конструктор копіювання та конструктор без аргументів;
- якщо в базовому класі всі конструктори – з аргументами, то похідний клас має мати конструктор, в якому потрібно явно викликати конструктор базового класу;
- при створенні об'єкта похідного класу спочатку викликається конструктор базового класу, а потім – похідного.

Деструктори

Деструктор, як і конструктори, не успадковується, а створюється в похідному класі. Деструктори опрацьовуються наступним чином:

- при відсутності явно визначеного деструктора в похідному класі система автоматично створює деструктор за умовчанням;
- деструктор базового класу викликається в деструкторі похідного класу автоматично – незалежно від того, чи він визначений явно, чи створений автоматично;
- деструктори викликаються для знищення об'єктів в порядку, зворотному до порядку виклику конструкторів при створенні об'єктів: за принципом LIFO (last input – first output, останній зайшов – перший вийшов), – тобто, при створенні об'єкта похідного класу спочатку викликаються конструктори базових класів, а потім – похідного, а при знищенні – спочатку викликається деструктор похідного класу, а потім – базових.

Поля та методи при успадкуванні

Похідний клас успадковує структуру (всі поля) та поведінку (всі методи) базового класу. Тобто, похідний клас буде мати всі поля і всі методи базового класу (хоча, якщо вони були приватні, доступу до них мати не буде).

Поля

Якщо нові поля в похідному класі не добавляються, то розмір похідного класу збігається з розміром базового.

Похідний клас може добавити власні поля:

```
class Point2
{
    int x, y;

public:
    // ...
};

class Point3: public Point2
{
    int z;

public:
    // ...
};
```

Добавлені поля має ініціалізувати конструктор похідного класу.

Добавлені у похідному класі поля можуть збігатися за іменем та за типом з полями базового класу – в цьому випадку нове поле *приховує* поле базового класу, тому для доступу до зазначеного поля базового класу з методів похідного класу слід використовувати префікс – кваліфікатор базового класу (при цьому це поле має бути доступним у похідному класі):

```
class A
{
protected:
    int a, b;    // поля будуть доступні в похідних класах

public:
    // ...
};

class B: public A
{
    int a, c;    // поле a приховує однойменне поле базового класу

public:
    // ...
    void f()
    {
        a = 1;    // власне поле
        A::a = 2; // успадковане поле
    }
};
```

Методи

1. Похідний клас успадковує всі методи базового класу, крім конструкторів, деструктора та операції присвоєння – вона створюється для нового класу автоматично, якщо не визначена явно.

2. У похідному класі за потреби необхідно явно визначати свої конструктори, деструктор та перевантажені операції присвоєння – бо вони не успадковуються від базового класу. Проте їх можна викликати явно при визначенні конструкторів, деструктора чи перевантаженні операції присвоєння в похідному класі.

3. У похідному класі доступні операції присвоєння, визначені в базовому класі. Це правило діє не лише на операцію присвоєння, а і на всі методи, які перевизначаються в похідному класі, – тобто, методи похідного класу, сигнатура яких (ім'я та список параметрів) збігається із сигнатурою відповідного методу базового класу.

Для цього в тілі похідної операції присвоєння слід викликати базову операцію, використовуючи її функціональну форму та повне ім'я (з префіксом – іменем класу):

```
A::operator = (r); // явний виклик базової операції присвоєння
```

4. Механізм викликів методів у похідному класі:

Коли деякий об'єкт викликає певний метод, починається пошук адреси коду цього метода. Для цього переглядається таблиця методів відповідного класу. Якщо адреса цього метода є в цій таблиці – управління передається відповідному методу і виконується його код. Якщо ж адреси цього метода в таблиці немає – пошук продовжується в таблиці методів батьківського класу. Якщо адреса методу є в таблиці методів батьківського класу – передається управління цьому методу і виконується його код. Якщо адреси метода немає в таблиці методів батьківського класу – пошук продовжується в таблицях базових класів, вгору по ієрархії. Якщо на деякому рівні буде знайдено цей метод – йому буде передано управління. Якщо такого метода немає ніде в усій ієрархії класів – буде помилка.

Таким чином, успадкування дозволяє звертатися «вгору» по ієрархії класів.

5. У похідному класі можна визначати нові методи. В нових методах можна викликати будь-які доступні методи базового класу.

6. Успадкування вносить нові елементи в перевантаження функцій. Якщо за звичних обставин перевантажені функції мають відрізнятися набором параметрів, то при успадкуванні похідний клас може визначати методи, однойменні методам базового класу з тим самим набором параметрів, що і у відповідного метода базового класу.

7. Якщо в похідному класі ім'я методу збігається з іменем метода базового класу (параметри та тип результату можуть відрізнятися!), то цей метод похідного класу *приховує* відповідний метод базового класу. Для виклику зазначеного метода базового класу слід вказати префікс-кваліфікатор класу, при цьому цей метод має бути доступним у похідному класі.

Причина такого приховування базових методів для об'єктів похідного класу полягає в тому, що похідний об'єкт зв'язаний з таблицею методів похідного класу, і пошук метода в цій таблиці починається з адрес методів саме похідного класу (пошук виконується «знизу вгору»).

8. Об'єкти похідного класу можуть використовувати як операції базового класу, так і операції похідного.

Статичні елементи класу при успадкуванні

Статичні поля успадковуються, як і звичайні поля. Проте всі нащадки спільно використовують єдиний екземпляр статичних полів.

Статичні методи успадковуються як і звичайні.

При звертанні до успадкованих статичних елементів класу можна використовувати або префікс базового класу, або префікс похідного.

Вкладені класи при успадкуванні

Ніяких обмежень при успадкуванні вкладених класів немає: зовнішній клас може успадковувати від вкладеного і навпаки, вкладений клас може успадковувати від зовнішнього.

Слід подбати про доступність базового класу в точці успадкування.

Принцип підстановки, операція присвоєння та розщеплення

Принцип підстановки

Відкрите успадкування встановлює між класами відношення «є різновидом» (іншими словами, «є підмножиною», «*is a*»): похідний клас є різновидом (підмножиною) базового класу. Це означає, що скрізь, де може бути використаний об'єкт базового класу (при присвоєнні, при передаванні параметрів та поверненні результату функції), замість нього можна підставляти об'єкт похідного класу. Така можливість підстановки об'єктів похідного класу замість об'єктів базового класу називається *принципом підстановки*, вона автоматично забезпечується компілятором. Цей принцип працює не лише для об'єктів, а і для посилань та

для вказівників: замість посилання (вказівника) на об'єкт базового класу можна підставити посилання (вказівник) на об'єкт похідного класу.

Принцип підстановки встановлює односторонній зв'язок: не можна замість об'єкта (посилання чи вказівника) похідного класу підставляти об'єкт (посилання чи вказівник) базового класу, наприклад: програміст є людиною, проте не кожна людина – програміст (тут людина – базовий клас, а програміст – похідний).

Операція присвоєння

Окрім конструкторів та деструктора, не успадковуються два види функцій: операція присвоєння та дружні функції. Операція присвоєння, як і конструктори та деструктор, для кожного класу створюється автоматично і має наступний прототип:

```
клас& клас::operator = (const клас &r);
```

Операція бінарна, лівим аргументом є поточний об'єкт. Ця стандартна операція забезпечує копіювання полів об'єкта r в поля поточного об'єкта.

Для будь-якого класу операцію присвоєння можна перевантажувати багато разів. При цьому допускається, щоб ні аргумент, ні результат не були того класу, в якому визначається така операція присвоєння. Єдиним обмеженням є видимість необхідних класів в точці визначення операції присвоєння.

Розщеплення та зрізування

Якщо в похідному класі визначено додаткові поля, то при підстановці об'єктів відбувається зрізування (розщеплення): об'єкту базового класу будуть присвоєні лише успадковані похідним класом поля:

```
class Point2
{
    int x, y;

public:
    // ...
};

class Point3: public Point2
{
    int z;

public:
    // ...
};

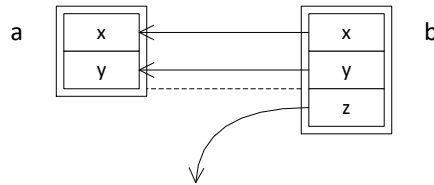
int main()
{
    Point2 a;
    Point3 b;
```

```

a = b;      // підстановка в присвоєнні
            // - розщеплення та зрізування

return 0;
}

```



Зрізування та розщеплення може відбутися лише при передаванні параметрів за значенням та при присвоєнні об'єктів, коли похідний клас визначає власні поля.

При передаванні параметрів за посиланням (чи за вказівником), а також при присвоєнні вказівників, – зрізування та розщеплення не відбувається.

Дружні функції при успадкуванні

Дружні функції не успадковуються, оскільки не є методами базового класу, хоча і мають доступ до внутрішньої структури класу. При відкритому успадкуванні можна не повторювати дружні функції для похідного класу, оскільки принцип підстановки забезпечує підстановку аргументів похідного класу на місце параметрів базового класу.

Відкрите успадкування – успадкування інтерфейсу

Просте відкрите успадкування означає, що *похідний клас є різновидом (підмножиною) базового класу*: кожний об'єкт похідного класу також є об'єктом базового класу. При відкритому успадкуванні інтерфейс базового класу буде доступний програмі-клієнту, яка використовує похідний клас: доступні елементи базового класу залишатимуться доступними у похідному класі. Таким чином, відкрите успадкування – це успадкування інтерфейсу.

Згідно принципу підстановки, скрізь, де можна використовувати об'єкт базового класу, замість нього можна підставляти об'єкт похідного класу. Цей принцип працює і для посилань, і для вказівників: замість посилання (вказівника) на об'єкт базового класу можна підставити посилання (вказівник) на об'єкт похідного класу. Перетворення типу при цьому вказувати не потрібно – воно виконується автоматично. Таке приведення типу називається *приведення, що підвищує* або *приведення вгору по ієрархії*.

Закрите успадкування. Делегування

Закрите успадкування – це *успадкування реалізації* (на відміну від успадкування інтерфейсу, яке здійснюється при відкритому успадкуванні): похідний клас *реалізується* за допомогою базового класу.

Просте закрите успадкування описується наступною конструкцією:

```
class ім'я_похідного_класу: private ім'я_базового_класу
{
    тіло_похідного_класу;
};
```

Закрите успадкування принципово відрізняється від відкритого: принцип підстановки не виконується. Це означає, що не можна присвоїти (без явного перетворення типу) об'єкт похідного класу базовому. Закрите успадкування зазвичай використовують, коли потрібно отримати функціональність базового класу, але не потрібні ні копіювання, ні присвоєння.

При закритому успадкуванні всі успадковані елементи базового класу стають приватними та недоступними програмі-клієнту:

```
class A
{
public:
    void f1() { /*...*/ }
    void f2() { /*...*/ }
};

class B: private A // успадковані методи недоступні клієнту
{ };
```

Програма, яка використовує клас B, не може викликати ні метод f1(), ні метод f2().

Щоб надати програмі таку можливість, потрібно в похідному класі заново реалізувати необхідні методи, які будуть *делегувати* (тобто, передавати) виклик відповідним методам базового класу:

```
class B: private A // успадковані методи недоступні клієнту
{
public:
    void f1() { A::f1(); } // делегування
    void f2() { A::f2(); } // делегування
};
```

Префікс при делегуванні вказувати обов'язково, бо інакше виникне рекурсія.

Успадковані методи можна відкрити в похідному класі також за допомогою конструкції `using`, яка має наступний синтаксис:

```
using ім'я_базового_класу::ім'я_елемента_базового_класу;
```

Для наведеного прикладу це буде виглядати так:

```
class B: private A // успадковані методи недоступні клієнту
{
public:
    using A::f1(); // відкрили успадкований метод
    using A::f2(); // відкрили успадкований метод
};
```

Таким чином, закрите успадкування обмежує функціональність, яку надає базовий клас; проте це обмеження можна зняти за допомогою *делегування* (передавання виклику) або конструкції `using`.

Закрите успадкування та композиція

Розглянемо попередній приклад з делегуванням:

```
class A
{
public:
    void f1() { /*...*/ }
    void f2() { /*...*/ }
};

class B: private A // успадковані методи недоступні клієнту
{
public:
    void f1() { A::f1(); } // делегування
    void f2() { A::f2(); } // делегування
};
```

При закритому успадкуванні ні структура (поля), ні поведінка (методи) базового класу не доступні програмі, яка використовує похідний клас. Лише за допомогою створення методів-оболонок, які *делегують* (тобто, передають) виклики відповідним методам базового класу, можна забезпечити доступ до функціональності базового класу (ще один спосіб, використання конструкції `using`, описаний в попередньому параграфі, тут не розглядається).

Перепишемо цей приклад з використанням композиції:

```
class A
{
public:
    void f1() { /*...*/ }
    void f2() { /*...*/ }
};

class B
{
    A a; // закрите поле-об'єкт класу A
public:
    void f1() { a.f1(); } // делегування
    void f2() { a.f2(); } // делегування
};
```

Зміст при такій організації зв'язків між класами **A** та **B** – той самий, що і при закритому успадкуванні класу **B** від класу **A**: клас **B** містить (успадковує) реалізацію класу **A**, але не надає програмі-клієнту його інтерфейсу.

Таким чином, закрите успадкування – це різновид композиції.

Закрите успадкування – успадкування реалізації

При закритому успадкуванні *похідний клас реалізується за допомогою базового*, тобто, – закрите успадкування – це *успадкування реалізації*.

Оскільки принцип підстановки при закритому успадкуванні не виконується, то не можна присвоїти (без явного перетворення типу) об'єкт похідного класу базовому. Тому закрите успадкування слід використовувати в тих випадках, коли потрібно зберегти функціональність базового класу, але не потрібні ні присвоєння, ні копіювання.

Відмінності структур та об'єднань від класів

Структури – теж класи, всі елементи структур за умовчанням доступні (в класах – приватні); ключ успадкування за умовчанням – `public` (для класів – `private`), тобто для структур за умовчанням реалізується відкрите успадкування (а для класів – закрите).

Множинне успадкування

Механізм множинного успадкування з концептуальної точки зору є цілком природнім: якщо кожний із базових класів містить лише частину необхідних властивостей, то чому би не об'єднати ці властивості у їх спільному класі-нащадку? Адже і в реальному житті кожна дитина має двох батьків:



Множинне успадкування, реалізоване в C++, не позбавлене недоліків і часто є джерелом різних проблем. Недаремно в більш нових об'єктно-орієнтованих мовах java та C# множинне успадкування класів заборонене, дозволяється лише множинна реалізація інтерфейсів (інтерфейси будуть вивчатися в темі «Віртуальні функції. Поліморфізм»).

Множинне успадкування відрізняється від простого (одиначного) наявністю кількох базових класів, наприклад:

```
class A {};  
  
class B {};  
  
class C: public A, public B  
{};
```

Базові класи перелічуються через кому, їх кількість не обмежена. Ключ успадкування для кожного базового класу може бути різним: від одного класу можна успадковувати відкрито, від іншого – закрито, від третього – захищено.

При множинному успадкуванні виконується все те ж саме, що і при одиночному: похідний клас успадковує структуру (всі елементи даних – поля) та поведінку (всі методи) всіх своїх базових класів.

При цьому:

- 1) З конструкторами при множинному успадкуванні слід поступати так ж само, як і при одиночному: якщо в базових класах визначені конструктори, то і в похідному класі слід визначити конструктор. При цьому конструктори-ініціалізатори базових класів слід явно викликати в конструкторі ініціалізації похідного класу.
- 2) При відкритому множинному успадкуванні виконується принцип підстановки: замість об'єкта одного із базових класів можна підставити об'єкт похідного класу.

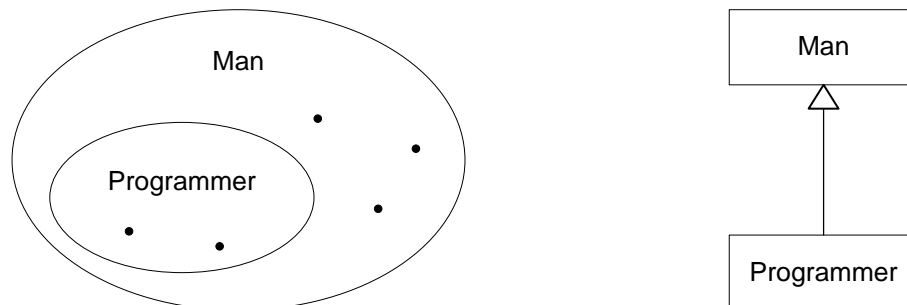
Теоретичні відомості

Відношення *композиції / агрегування* – це відношення «*ціле – частина*», тобто відношення «*має*» (*has a*): двигун – це частина автомобіля. Існує ще один важливий вид зв'язку між класами – відношення «*є різновидом*», тобто «*є підмножиною*» (*is a*): автомобіль – це різновид (підмножина) транспорту. Таке відношення називається *успадкуванням*.

Успадкування – один з трьох «*китів*» (разом з інкапсуляцією та поліморфізмом), на якому стоїть об'єктно-орієнтоване програмування. При успадкуванні обов'язково є *клас-предок* (батьківський клас; той, що породжує) та *клас-нащадок* (дочірній клас; той, який породжений). В C++ клас-предок називається *базовим*, а клас-нащадок – *похідним*. Всі терміни еквівалентні.

На UML-class diagram (UML-діаграмі класів) зв'язок *успадкування* (його ще називають *генералізація*) позначається трикутною незаповненою стрілкою, яка закінчує суцільну лінію, проведenu від похідного класу до базового:

Наприклад, розглянемо класи **Man** (Людина) та **Programmer** (Програміст). Вони описують сукупності – множину **Люди** та множину **Програмісти**. Прийнято вважати, що кожний програміст є людиною (кодерів з Марсу та хакерів з планети Нібіру зараз не враховуємо), проте не кожна людина є програмістом: **Програмісти** – підмножина **Людей**:



На UML-діаграмі класів такий зв'язок позначаємо трикутною незаповненою стрілкою від похідного класу **Programmer** до базового класу **Man**.

Зв'язок успадкування означає, що при визначенні похідного класу ми вказуємо лише його власні характеристики. Характеристики, спільні з базовим класом, при визначенні похідного класу вказувати не потрібно. Директива успадкування означає, що весь контент (всі характеристики) базового класу будуть присутні в похідному класі.

Термін *успадкування* (англ. *inheritance*) використовується саме тому, що похідний клас – нащадок отримує у спадщину (англ. *inheritage*) весь контент (всі характеристики) базового класу – предка.

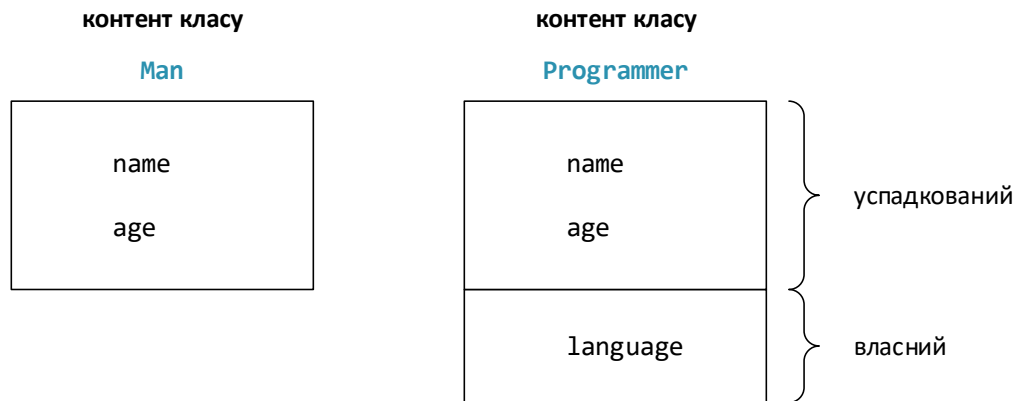
```

class Man
{
    string name;
    int age;
};

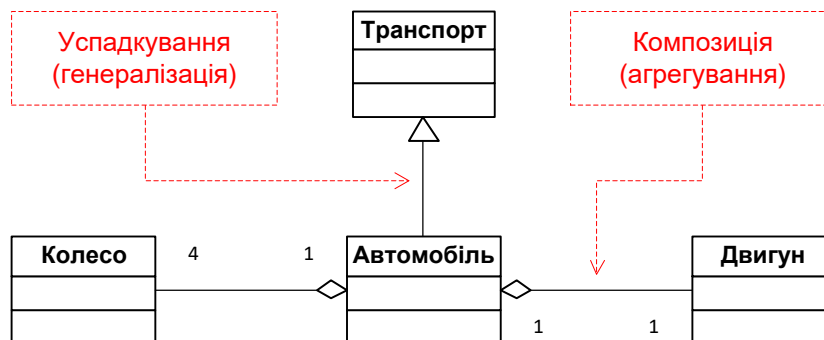
class Programmer : Man
{
    string language;
};

```

директива
успадкування



Інший приклад: класи **Транспорт** та **Автомобіль**: автомобілі є різновидом (підмножиною) транспорту. Наступна UML-діаграма показує зв'язки успадкування (транспорт – предок, автомобіль – нащадок) та агрегування (автомобіль містить 4 колеса та 1 двигун):



Відношення між батьківським класом та його нащадками називається *ієрархією успадкування*. Глибина (тобто, кількість рівнів) успадкування не обмежена.

Проектування ієрархії класів

Послідовність дій

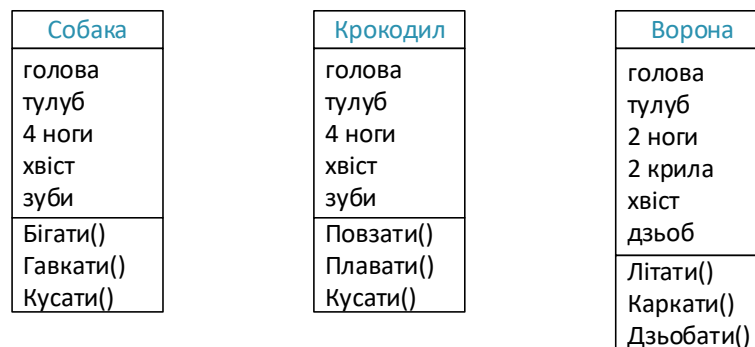
- Виконується «знизу догори»
- Починаємо від конкретних класів (вони будуть на нижніх рівнях ієрархії)
- Всі спільні характеристики та спільні дії об'єднуємо та виносимо у більш загальні базові класи, які утворюють вищі рівні ієрархії
- Продовжуємо, поки не дійдемо до найбільш загальних класів

Приклад

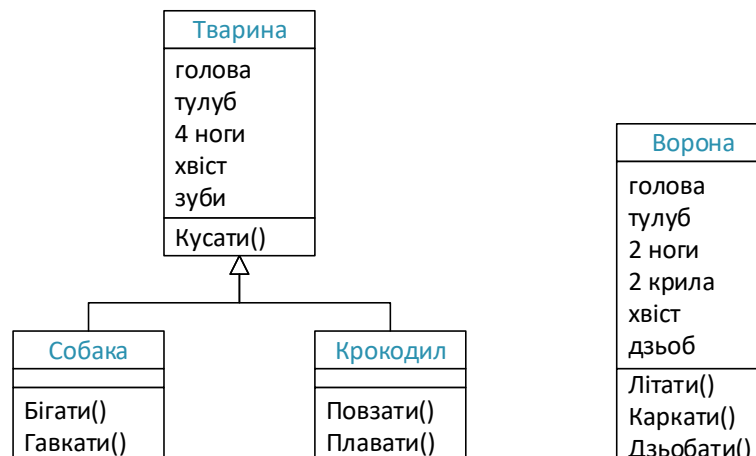
Припустимо, є завдання: спроектувати програму, яка керуватиме охоронною системою з роботів трьох видів

- «Собака» – охороняє на землі; має тулуб, 4 ноги, хвіст, голову, зуби; бігає, гавкає, кусає;
- «Ворона» – охороняє в повітрі; має тулуб, 2 ноги, 2 крила, хвіст, голову, дзьоб; літає, каркає, дзьобає;
- «Крокодил» – охороняє у воді (довкола об'єкта охорони – рів з водою); має тулуб, 4 ноги, хвіст, голову, зуби; повзає, плаває, кусає.

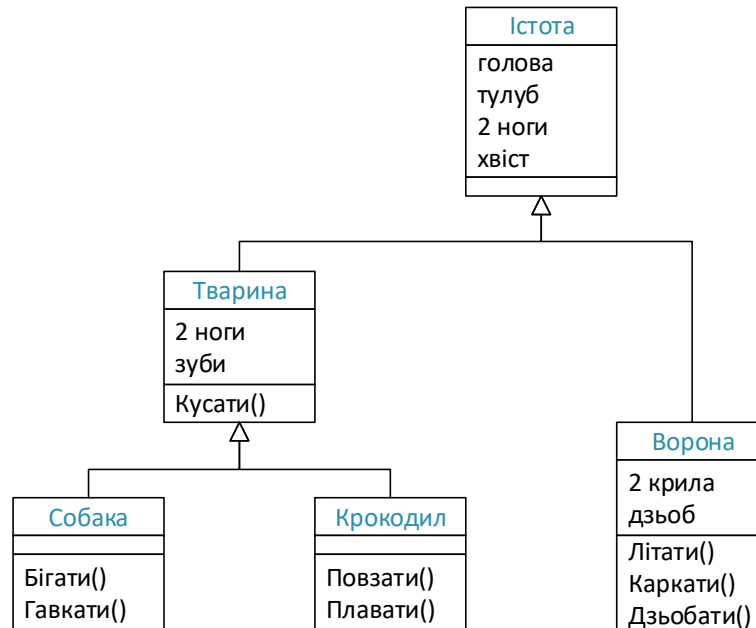
Маємо початкову систему класів:



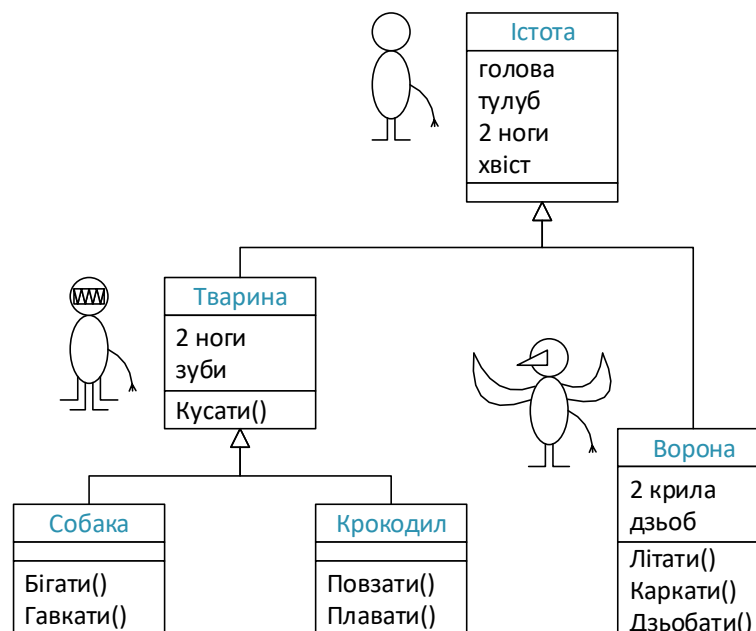
На основі аналізу класів Собака та Крокодил утворимо клас Тварина:



На основі аналізу класів **Тварина** та **Ворона** утворимо клас **Істота**:



Остаточно маємо:



Проектування класів

При проектуванні класів ми розглядаємо не статичну картину предметної області (фотознімок), як це робиться при структурному програмуванні, а динамічну (потік відео). Це означає, що ми маємо враховувати не лише дані, які описують предметну область, а і дії над цими даними. Тобто, слід не лише спроектувати структуру даних, а (що не менш важливо) – структуру функцій, які опрацьовують ці дані.

При структурному проектуванні ми розподіляли дані між сутностями як атрибути відповідних сутностей.

При об'єктно-орієнтованому проектуванні ми маємо розподілити дані та функції як поля та методи відповідних класів.

Правило проектування класів

Принцип **KISS: Keep it simple stupid!** (не потребує перекладу)

- Кожний клас має бути якомога простіший і має представляти лише одну сутність (тобто, лише один набір однотипних екземплярів предметної області);
- Краще багато простих класів, ніж один чи кілька складних. Слід уникати класу, який поєднує у собі все (всі сутності предметної області) – тобто, уникати «класу-Бога»;
- Класи мають мати значущі імена (ім'я класу – це ім'я відповідної сутності) та очевидну будову і функціонал.

Правило проектування методів

Принцип **KISS: Keep it simple stupid!** (не потребує перекладу)

- Кожний метод має бути якомога простішим і має виконувати лише одну роль;
- Краще багато простих методів, ніж один чи кілька складних. Слід уникати методу, який виконує всі дії – тобто, уникати «методу-Бога»;
- Методи мають мати значущі імена (ім'я методу – це ім'я відповідної ролі) та очевидний функціонал.

Успадкування та композиція

Механізм успадкування класів дозволяє будувати ієрархії, в яких похідні класи отримують елементи батьківських (базових) класів та можуть доповнювати їх чи змінювати їхні властивості. При великій кількості ніяк не пов'язаних класів управляти ними стає неможливим. Успадкування дозволяє вирішити цю проблему шляхом впорядкування та ранжирування класів, тобто об'єднання загальних для кількох класів властивостей в одному класі та використання його в якості базового.

Класи, які знаходяться ближче до початку ієрархії, поєднують в собі найбільш загальні риси для всіх класів, що знаходяться нижче в ієрархії. По мірі просування вниз по ієрархії класи отримують все більше конкретних рис.

При успадкуванні між класами встановлюються набагато тісніші зв'язки, ніж при композиції.

Просте успадкування

Простим (або *одиначним*) називається успадкування, при якому похідний клас має лише одного предка. Просте успадкування визначається наступною конструкцією:

```
class ім'я_похідного_класу: [ключ_успадкування] ім'я_базового_класу
{
    тіло_похідного_класу;
};
```

ключ_успадкування – необов'язковий елемент синтаксичної конструкції (тому позначений в квадратних дужках), – це модифікатор доступу, який визначає доступність елементів базового класу в похідному класі. Квадратні дужки – не елемент синтаксису, а показують, що ключ успадкування може бути відсутнім.

Ключі успадкування та директиви доступу

При розробленні окремих класів використовуються два модифікатори доступу до елементів класу: **public** (доступні елементи) **private** (приховані елементи). При успадкуванні використовується ще один: **protected** (захищені елементи):

```
class і'мя_класу
{
    private:
        приховані_елементи_класу;
    protected:
        захищені_елементи_класу;
    public:
        доступні_елементи_класу;
};
```

Директиви доступу private, protected та public

Директиви доступу можна вказувати в довільному порядку, їх кількість не обмежена.

Директива **private:** починає секцію прихованих елементів, які доступні *лише методам цього класу та його друзям* (дружнім класам та дружнім функціям).

Директива **protected:** починає секцію захищених елементів, які доступні *лише методам цього класу, його друзям* (дружнім класам та дружнім функціям) *та нащадкам*.

Директива **public:** починає секцію доступних елементів, які *повністю доступні поза класом* (в тому числі і методам цього класу, його друзям та його нащадкам).

Модифікатор доступу	Доступність		
	У цьому ж класі: його методам та його друзям	Нащадкам цього класу	Скрізь поза класом
private	+		
protected	+	+	
public	+	+	+

Доступність елементів класу залежить від модифікатора доступу – від мінімальної (**private**) до максимальної (**public**).

Ключі успадкування **private**, **protected** та **public**

Ключ успадкування (вказується в директиві успадкування при визначенні похідного класу) разом з модифікатором доступу (вказаним в базовому класі) визначає доступність успадкованих елементів базового класу при використанні похідного класу.

Якщо ключ успадкування – **public**, то успадкування називається *відкритим*; ключ успадкування **protected** визначає *захищене* успадкування; а ключ **private** визначає *закрите* успадкування.

Доступ до елементів класу при різних ключах успадкування

Припустимо, є наступна ієрархія класів:

```
class імя_базового_класу
{
    private:
        приховані_елементи_базового_класу;
    protected:
        захищені_елементи_базового_класу;
    public:
        доступні_елементи_базового_класу;
};

class імя_похідного_класу: [ключ_успадкування] імя_базового_класу
{
    тіло_похідного_класу;
};
```

Доступність успадкованих елементів базового класу в об'єкті похідного класу залежить від модифікатора доступу в базовому класі та ключа успадкування. Яким би не був ключ успадкування, приватні елементи базового класу недоступні поза класом.

В наступній таблиці наведено всі варіанти доступності елементів базового класу в об'єктах похідного класу при будь-яких значеннях ключа успадкування:

Таблиця

доступності елементів базового класу в об'єктах похідного класу

Модифікатор доступу в базовому класі	Ключ успадкування			
	відсутній	private	protected	public
	закрите успадкування		захищене успадкування	відкрите успадкування
private	нема доступу	нема доступу	нема доступу	нема доступу
protected	private	private	protected	protected
public	private	private	protected	public

За відсутності явного ключа успадкування для класів за умовчанням реалізується закрите успадкування – таке ж, як і при ключі `private` (для структур – за умовчанням реалізується відкрите, – як при ключі `public`).

Розглянемо приклад:

```
class A
{
private:
    int x;
protected:
    int y;
public:
    int z;
};

class B: private A
{
    void f()
    {
        x = 1; // помилка: немає доступу (A::x - приватне поле)
        y = 2; // вірно: є доступ (A::y - захищене поле)
        z = 3; // вірно: є доступ (A::z - доступне поле)
    }
};

class C: protected A
{
};

class D: public A
{
};

int main()
{
    A a;
    a.x = 1; // помилка: немає доступу (A::x - приватне поле)
    a.y = 2; // помилка: немає доступу (A::y - захищене поле)
    a.z = 3; // вірно: є доступ (A::z - доступне поле)

    B b;
    b.x = 1; // помилка: немає доступу (A::x - приватне поле)
    b.y = 2; // помилка: немає доступу (A::y - захищене поле)
    b.z = 3; // помилка: немає доступу (B - закритий нащадок A)

    C c;
    c.x = 1; // помилка: немає доступу (A::x - приватне поле)
    c.y = 2; // помилка: немає доступу (A::y - захищене поле)
    c.z = 3; // помилка: немає доступу (C - захищений нащадок A)

    D d;
    d.x = 1; // помилка: немає доступу (A::x - приватне поле)
    d.y = 2; // помилка: немає доступу (A::y - захищене поле)
    d.z = 3; // вірно: є доступ (A::z - доступне поле)

    return 0;
}
```

Яким би не був ключ успадкування, приватні елементи базового класу недоступні його нащадкам (нащадкам доступні лише захищені та відкриті елементи базового класу):

```
class B: private A
{
    void f()
    {
        x = 1; // помилка: немає доступу (A::x - приватне поле)
        y = 2; // вірно: є доступ (A::y - захищене поле)
        z = 3; // вірно: є доступ (A::z - доступне поле)
    }
};
```

Із-зовні (зі сторони) доступ можливий лише до відкритих елементів класу:

```
A a;
a.x = 1; // помилка: немає доступу (A::x - приватне поле)
a.y = 2; // помилка: немає доступу (A::y - захищене поле)
a.z = 3; // вірно: є доступ (A::z - доступне поле)
```

Закрите поле *x* доступне лише методам класу *A* та його друзям (зобразимо це «закритим» прямокутником, нарисованим суцільною лінією). Захищене поле *y* доступне лише методам класу *A*, його друзям та методам його нащадків (його зобразимо «напіввідкритим» прямокутником, одна із сторін якого – пунктирна). Відкрите поле *z* доступне скрізь: методам класу *A* та його друзям, методам його нащадків, стороннім функціям (зокрема, *main()*). Зобразимо його «відкритим» прямокутником, без одної сторони:

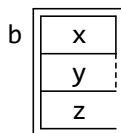
a	x
	y
	z

Ключ успадкування звужує доступність елементів базового класу.

При ключі *private* (закрите успадкування) захищені та доступні елементи базового класу стають прихованими в похідному, – вони стають недоступними поза похідним класом. Тобто, при звертанні до цих елементів базового класу за допомогою похідного, вони будуть доступні лише методам похідного класу та його друзям – доступ до захищених та відкритих елементів базового класу звужується в похідному класі до рівня «закриті елементи»:

```
B b;
b.x = 1; // помилка: немає доступу (A::x - приватне поле)
b.y = 2; // помилка: немає доступу (A::y - захищене поле)
b.z = 3; // помилка: немає доступу (B - закритий нащадок A)
```

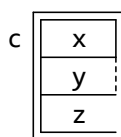
Об'єкт *b* – це «закрита обгортка» для успадкованих від класу *A* полів. Зобразимо це «закритим» прямокутником, нарисованим суцільною лінією, довкола прямокутників, що зображують успадковані поля. Всі поля отримали «закриту» обгортку і стали недоступними поза класом:



При ключі `protected` (захищене успадкування) захищені та доступні елементи базового класу стають захищеними в похідному, – вони стають доступні лише нащадкам похідного класу. Тобто, при звертанні до цих елементів базового класу за допомогою похідного, вони будуть доступні лише методам похідного класу, його друзям та його прямим нащадкам – доступ до відкритих елементів базового класу звужується в похідному класі до рівня «захищені елементи»:

```
C c;
c.x = 1; // помилка: немає доступу (A::x - приватне поле)
c.y = 2; // помилка: немає доступу (A::y - захищене поле)
c.z = 3; // помилка: немає доступу (B - захищений нащадок A)
```

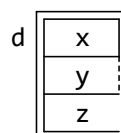
Об'єкт `c` – це «захищена обгортка» для успадкованих від класу `A` полів. Зобразимо це «напів-відкритим» прямокутником з пунктирною однією стороною, нарисованим довкола прямокутників, що зображують успадковані поля. Всі поля отримали «захищену» обгортку і тепер недоступні поза класом:



При ключі `public` (відкрите успадкування) ніякого звуження не відбувається: захищені та доступні елементи базового класу залишаються відповідно захищеними та доступними в похідному:

```
D d;
d.x = 1; // помилка: немає доступу (A::x - приватне поле)
d.y = 2; // помилка: немає доступу (A::y - захищене поле)
d.z = 3; // вірно: є доступ (A::z - доступне поле)
```

Об'єкт `d` – це «відкрита обгортка» для успадкованих від класу `A` полів. Зобразимо це «відкритим» прямокутником з відсутньою однією стороною, нарисованим довкола прямокутників, що зображують успадковані поля. Всі поля отримали «відкриту» обгортку і ніяких додаткових обмежень доступу не мають. Поля, які були доступними в базовому класі, так і залишилися доступними поза класом:



Просте відкрите успадкування

Найбільш поширеним є просте відкрите успадкування. Багато об'єктно-орієнтованих мов програмування (зокрема, java, C# та ін.) лише його і реалізують. Тому будемо розглядати нюанси простого відкритого успадкування, а потім зазначимо відмінності для випадку закритого успадкування.

Просте відкрите успадкування описується наступною конструкцією:

```
class ім'я_похідного_класу: public ім'я_базового_класу
{
    тіло_похідного_класу;
};
```

Конструктори і деструктори при успадкуванні

Конструктори

Конструктори не успадковуються – вони створюються в похідному класі (якщо не визначені програмістом явно). Конструктори опрацьовуються наступним чином:

- якщо в базовому класі немає явно визначених конструкторів або є конструктор без аргументів (або аргументи присвоюються за умовчанням), то в похідному класі конструктор можна не писати – автоматично будуть створені конструктор копіювання та конструктор без аргументів;
- якщо в базовому класі всі конструктори – з аргументами, то похідний клас має мати конструктор, в якому потрібно явно викликати конструктор базового класу;
- при створенні об'єкта похідного класу спочатку викликається конструктор базового класу, а потім – похідного.

Приклад 1.

```
class Money // Базовий клас - з конструктором за умовчанням
{
    double summa;

public:
    Money(const double &r = 0.0) // конструктор ініціалізації та без аргументів
    {
        summa = r;
    }
};

class Dollars: public Money {}; // похідний клас - без конструкторів
```

Приклад 2.

```
class Basis // Базовий клас - без конструктора за умовчанням
{
    int a, b;
```

```

public:
    Basis(const int x, const int y) // лише явний конструктор з аргументами
    {
        a = x;
        b = y;
    }
};

class Inherit: public Basis
{
    int sum;

public:
    Inherit(const int x, const int y, const int s)
        : Basis(x, y) // явний виклик конструктора
    {
        sum = s;
    }
};

```

В першому прикладі в класі `Money` явно визначено конструктор ініціалізації, який виконує роль конструктора без аргументів, бо аргумент задано за умовчанням. В похідному класі `Dollars` конструктор можна не писати.

В другому прикладі в класі `Basis` явно визначено конструктор ініціалізації, при цьому конструктор копіювання буде створено автоматично, а конструктор без аргументів – ні. В похідному класі `Inherit` конструктор базового класу явно викликається в списку ініціалізації.

Деструктори

Деструктор, як і конструктори, не успадковується, а створюється в похідному класі. Деструктори опрацьовуються наступним чином:

- при відсутності явно визначеного деструктора в похідному класі система автоматично створює деструктор за умовчанням;
- деструктор базового класу викликається в деструкторі похідного класу автоматично – незалежно від того, чи він визначений явно, чи створений автоматично;
- деструктори викликаються для знищення об'єктів в порядку, зворотному до порядку виклику конструкторів при створенні об'єктів: за принципом LIFO (last input – first output, останній зайшов – перший вийшов), – тобто, при створенні об'єкта похідного класу спочатку викликаються конструктори базових класів, а потім – похідного, а при знищенні – спочатку викликається деструктор похідного класу, а потім – базових.

Поля та методи при успадкуванні

Похідний клас успадковує структуру (всі поля) та поведінку (всі методи) базового класу. Тобто, похідний клас буде мати всі поля і всі методи базового класу (хоча, якщо вони були приватні, доступу до них мати не буде).

Поля

Якщо нові поля в похідному класі не додаються, то розмір похідного класу збігається з розміром базового.

Похідний клас може додати власні поля:

```
class Point2
{
    int x, y;

public:
    // ...
};

class Point3: public Point2
{
    int z;

public:
    // ...
};
```

Додані поля має ініціалізувати конструктор похідного класу.

Додані у похідному класі поля можуть збігатися за іменем та за типом з полями базового класу – в цьому випадку нове поле *приховує* поле базового класу, тому для доступу до зазначеного поля базового класу з методів похідного класу слід використовувати префікс – кваліфікатор базового класу (при цьому це поле має бути доступним у похідному класі):

```
class A
{
protected:
    int a, b;    // поля будуть доступні в похідних класах

public:
    // ...
};

class B: public A
{
    int a, c;    // поле a приховує однойменне поле базового класу

public:
    // ...
    void f()
    {
        a = 1;    // власне поле
        A::a = 2; // успадковане поле
    }
};
```

Методи

1. Похідний клас успадковує всі методи базового класу, крім конструкторів, деструктора та операції присвоєння – вона створюється для нового класу автоматично, якщо не визначена явно.

2. У похідному класі за потреби необхідно явно визначати свої конструктори, деструктор та перевантажені операції присвоєння – бо вони не успадковуються від базового класу. Проте їх можна викликати явно при визначенні конструкторів, деструктора чи перевантаженні операції присвоєння в похідному класі.

Розглянемо це на прикладі операції присвоєння:

```
class A
{
public:
    A& operator = (const A& r)
    {
        cout << "A::operator = ()" << endl;
        return *this;
    }
};

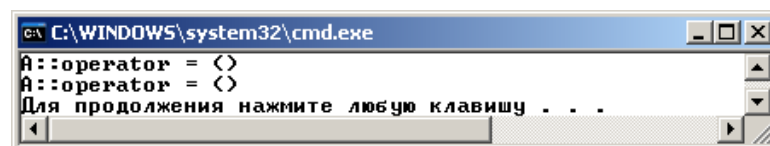
class B: public A
{};

int main()
{
    A a1, a2;
    a1 = a2;

    B b1, b2;
    b1 = b2; // "стандартна" операція присвоєння в похідному класі
             // за умовчанням викликає базову операцію присвоєння

    return 0;
}
```

В класі B не визначено явно операції присвоєння, компілятор створить свою («стандартну»), яка за умовчанням буде викликати операцію присвоєння базового класу:



Визначимо в класі B свою операцію присвоєння (новий код виділено сірим фоном):

```
class A
{
public:
    A& operator = (const A& r)
    {
        cout << "A::operator = ()" << endl;
    }
};
```



```

        return *this;
    }
};

class B: public A
{
public:
    B& operator = (const B& r)
    {
        cout << "B::operator = ()" << endl;
        return *this;
    }
};

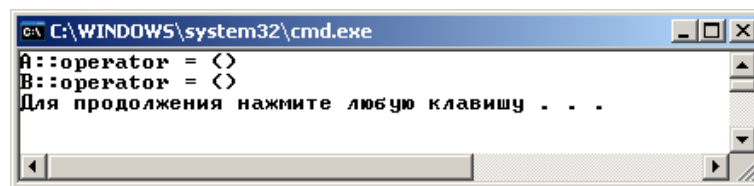
int main()
{
    A a1, a2;
    a1 = a2;

    B b1, b2;
    b1 = b2; // перевизначена в похідному класі операція присвоєння вже
             // за умовчанням не викликає базову операцію присвоєння

    return 0;
}

```

Результат:



– визначена явно операція присвоєння похідного класу ніяких неявних дій не виконує, в т.ч. – не викликає за умовчанням операцію присвоєння базового класу.

3. У похідному класі доступні операції присвоєння, визначені в базовому класі. Це правило діє не лише на операцію присвоєння, а і на всі методи, які перевизначаються в похідному класі, – тобто, методи похідного класу, сигнатура яких (ім'я та список параметрів) збігається із сигнатурою відповідного методу базового класу.

Для цього в тілі похідної операції присвоєння слід викликати базову операцію, використовуючи її функціональну форму та повне ім'я (з префіксом – іменем класу):

```
A::operator = (r); // явний виклик базової операції присвоєння
```

Приклад:

```

class A
{
public:
    A& operator = (const A& r)
    {
        cout << "A::operator = ()" << endl;
        return *this;
    }
}

```

```

};

class B: public A
{
public:
    B& operator = (const B& r)
    {
        A::operator = (r); // явний виклик базової операції присвоєння
        // тут можуть бути свої дії - присвоєння елементів похідного класу

        cout << "B::operator = ()" << endl;
        return *this;
    }
};

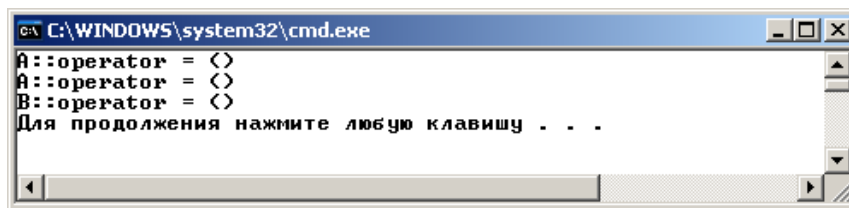
int main()
{
    A a1, a2;
    a1 = a2;

    B b1, b2;
    b1 = b2; // перевизначена в похідному класі операція присвоєння
            // явно викликає базову операцію присвоєння

    return 0;
}

```

Результат:



```

C:\WINDOWS\system32\cmd.exe
A::operator = (<)
A::operator = (<)
B::operator = (<)
Для продолжения нажмите любую клавишу . . .

```

Якщо ж у похідному класі не потрібно виконувати ніяких власних дій стосовно присвоєння власних елементів, то можна обійтися операцією присвоєння базового класу (тоді присвоюватиметься «зріз» базового класу). Для цього слід в похідному класі «відкрити» базову операцію присвоєння:

```
using A::operator =; // "відкрили" базову операцію присвоєння
```

Приклад:

```

class A
{
public:
    A& operator = (const A& r)
    {
        cout << "A::operator = ()" << endl;
        return *this;
    }
};

class B: public A
{
public:
    using A::operator =; // "відкрили" базову операцію присвоєння
};

```

```

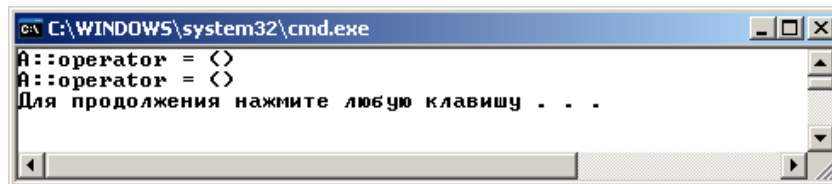
int main()
{
    A a1, a2;
    a1 = a2;

    B b1, b2;
    b1 = b2; // виклик "відкритої" базової операції присвоєння

    return 0;
}

```

Результат:



4. Механізм викликів методів у похідному класі:

Коли деякий об'єкт викликає певний метод, починається пошук адреси коду цього метода. Для цього переглядається таблиця методів відповідного класу. Якщо адреса цього метода є в цій таблиці – управління передається відповідному методу і виконується його код. Якщо ж адреси цього метода в таблиці немає – пошук продовжується в таблиці методів батьківського класу. Якщо адреса методу є в таблиці методів батьківського класу – передається управління цьому методу і виконується його код. Якщо адреси метода немає в таблиці методів батьківського класу – пошук продовжується в таблицях базових класів, вгору по ієрархії. Якщо на деякому рівні буде знайдено цей метод – йому буде передано управління. Якщо такого метода немає ніде в усій ієрархії класів – буде помилка.

Таким чином, успадкування дозволяє звертатися «вгору» по ієрархії класів.

5. У похідному класі можна визначати нові методи. В нових методах можна викликати будь-які доступні методи базового класу.

6. Успадкування вносить нові елементи в перевантаження функцій. Якщо за звичних обставин перевантажені функції мають відрізнятися набором параметрів, то при успадкуванні похідний клас може визначати методи, однойменні методам базового класу з тим самим набором параметрів, що і у відповідного метода базового класу.

7. Якщо в похідному класі ім'я методу збігається з іменем метода базового класу (параметри та тип результату можуть відрізнятися!), то цей метод похідного класу *приховує* відповідний метод базового класу. Для виклику зазначеного метода базового класу слід вказати префікс-кваліфікатор класу, при цьому цей метод має бути доступним у похідному класі:

```

class A
{
public:
    int f() const                // звичайний метод
    {
        cout << "A::f()" << endl;
        return 0;
    }
    void f(const string &s) const // перевизначений метод
    {
        cout << "A::f(string)" << endl;
    }
};

class B: public A
{
public:
    int f(int i) const          // перевизначений метод
                                // з іншим набором параметрів
    {
        cout << "B::f(int)" << endl;
        return 0;
    }
};

int main()                    // виклик перевантажених методів
{
    A a;                      // об'єкт базового класу
    B b;                      // об'єкт похідного класу

    b.A::f();                 // вірно
    b.f();                   // помилка – відсутність аргументу
    b.f("abc");              // помилка – спроба перетворити char[4] до int
    a.f(1);                  // помилка – спроба перетворити int до string

    return 0;
}

```

Причина такого приховування базових методів для об'єктів похідного класу полягає в тому, що похідний об'єкт зв'язаний з таблицею методів похідного класу, і пошук метода в цій таблиці починається з адрес методів саме похідного класу (пошук виконується «знизу вгору»).

```

class Man
{
    /* ... */
public:
    void Display()
    {
        cout << "Man" << endl;
    }
};

class Student : public Man
{
    /* ... */
public:
    void Display()
    {
        cout << "Student" << endl;
    }
}

```

```

    }
};

int main()
{
    Student *s = new Student();

    s->Display(); // виклик власного метода

    dynamic_cast<Man*>(s)->Display(); // виклик успадкованого метода

    // або так:
    s->Man::Display();

    system("pause");
}

```

8. Об'єкти похідного класу можуть використовувати як операції базового класу, так і операції похідного, наприклад:

```

class Base
{
    int x, y;

public:
    Base(): x(), y() {} // конструктор без аргументів

    Base(int a, int b=0) // конструктор ініціалізації
    {
        x = a;
        y = b;
    }

    // методи доступу (аксесори)
    int getX() const { return x; }
    int getY() const { return y; }
    void setX(const int value) { x = value; }
    void setY(const int value) { y = value; }

    Base& operator ++() // префіксний інкремент
    {
        x++;
        return *this;
    }

    Base operator ++(int) // постфіксний інкремент
    {
        Base t = *this;
        x++;
        return t;
    }

    Base& operator +=(const Base &r) // додавання з присвоєнням
    {
        x += r.x;
        y += r.y;
        return *this;
    }
};

class Derive: public Base

```

```

{
public:
    Derive& operator --()           // новий метод: префіксний декремент
    {
        int x = getX();           // реалізуємо x--
        setX(--x);                // поле x - недоступне поза класом Base
        return *this;
    }
};

int main()
{
    Derive d;
    ++d; d++; --d; d+=4;           // викликаємо як власні, так і
                                   // успадковані операції

    return 0;
}

```

Статичні елементи класу при успадкуванні

Статичні поля

Статичні поля успадковуються, як і звичайні поля. Проте всі нащадки спільно використовують єдиний екземпляр успадкованих статичних полів.

Приклад 1.

```

class A
{
public:
    static int n;
};

class B: public A {};
int A::n = 0;

int main()
{
    A::n = 1;
    B::n = 2;
    cout << A::n << endl; // 2
    system("pause");
}

```

Приклад 2.

```

class A
{
public:
    static int n;
};

class B: public A
{
public:
    static int n;
};

int A::n = 0;
int B::n = 0;

```

```

int main()
{
    A::n = 1;
    B::n = 2;

    cout << A::n << endl;    // 1
    cout << B::n << endl;    // 2
    cout << B::A::n << endl;  // 1
    system("pause");
}

```

Статичні методи

Статичні методи успадковуються як і звичайні.

При звертанні до успадкованих статичних елементів класу можна використовувати або префікс базового класу, або префікс похідного, наприклад:

```

class A
{
public:
    static void f() {}
};

class B: public A {};

int main()
{
    A::f();
    B::f();

    return 0;
}

```

Вкладені класи при успадкуванні

Ніяких обмежень при успадкуванні вкладених класів немає: зовнішній клас може успадковувати від вкладеного і навпаки, вкладений клас може успадковувати від зовнішнього:

```

class A                                // зовнішній клас
{
};

class B
{
public:
    class C: public A                  // внутрішній клас успадковує від зовнішнього
    {
};

class D: public B::C                  // зовнішній клас успадковує від внутрішнього
{
};

class E
{
    class F: public B::C              // внутрішній клас успадковує від внутрішнього
    {
};
};

```

Слід подбати про доступність базового класу в точці успадкування: вкладений клас `E::F` визначений в секції `private` класу `E`, тому клас `F` поза класом `E` не доступний.

Принцип підстановки, операція присвоєння та розщеплення

Принцип підстановки

Відкрите успадкування встановлює між класами відношення «є різновидом» (іншими словами, «є підмножиною», «*is a*»): похідний клас є різновидом (підмножиною) базового класу. Це означає, що скрізь, де може бути використаний об'єкт базового класу (при присвоюванні, при передаванні параметрів та поверненні результату функції), замість нього можна підставляти об'єкт похідного класу. Така можливість підстановки об'єктів похідного класу замість об'єктів базового класу називається *принципом підстановки*, вона автоматично забезпечується компілятором. Цей принцип працює не лише для об'єктів, а і для посилань та для вказівників: замість посилання (вказівника) на об'єкт базового класу можна підставити посилання (вказівник) на об'єкт похідного класу:

```
class A
{};

class B: public A
{};

void f(A c) // функція очікує значення об'єкта базового класу
{}

void g(A *c) // функція очікує вказівника на об'єкт базового класу
{}

void h(A &c) // функція очікує посилання на об'єкт базового класу
{}

int main()
{
    A a, *pa;
    B b, *pb;

    pb = new B(); // вказівник на динамічний об'єкт похідного класу

    // демонстрація принципу підстановки
    // підстановка об'єктів в присвоєнні:
    a = b; // підставляємо об'єкт похідного класу
           // як значення для об'єкта базового класу

    // підстановка вказівників на об'єкти в присвоєнні:
    pa = pb; // підставляємо вказівник на об'єкт похідного класу
            // як значення для вказівника на об'єкт базового класу

    // підстановка об'єктів при передаванні параметрів:
    f(b); // у функцію замість об'єкта базового класу
         // передали об'єкт похідного класу
```



```

// підстановка вказівників на об'єкти при передаванні параметрів:
g(pb);    // у функцію замість вказівника на об'єкт базового класу
          // передали вказівник на об'єкт похідного класу

// підстановка об'єктів при передаванні параметрів-посилань на об'єкти:
h(b);     // у функцію замість посилання на об'єкт базового класу
          // передали об'єкт похідного класу

return 0;
}

```

Принцип підстановки встановлює односторонній зв'язок: не можна замість об'єкта (посилання чи вказівника) похідного класу підставляти об'єкт (посилання чи вказівник) базового класу, наприклад: програміст є людиною, проте не кожна людина – програміст (тут людина – базовий клас, а програміст – похідний).

Операція присвоєння

Окрім конструкторів та деструктора, не успадковуються два види функцій: операція присвоєння та дружні функції. Операція присвоєння, як і конструктори та деструктор, для кожного класу створюється автоматично і має наступний прототип:

```
клас& клас::operator = (const клас &r);
```

Операція бінарна, лівим аргументом є поточний об'єкт. Ця стандартна операція забезпечує копіювання полів об'єкта *r* в поля поточного об'єкта. Для базового класу *Base* та похідного класу *Derive* відповідні операції присвоєння мають вигляд:

```

class Base
{
public:
    Base& operator = (const Base &b);
};

class Derive: public Base
{
public:
    Derive& operator = (const Derive &d);
};

Base& Base::operator = (const Base &b)
{
    // ...
    return *this;
}

Derive& Derive::operator = (const Derive &d)
{
    // ...
    return *this;
}

```

Наявність цих методів дозволяє виконувати наступні присвоєння:

```

int main()
{
    Base b1, b2;    // об'єкти базового класу
    Derive d1, d2;  // об'єкти похідного класу

    b1 = b2;        // операція базового класу
    d1 = d2;        // операція похідного класу
    b1 = d2;        // операція базового класу
                    // базовий = похідний
                    // - використовується підстановка:
                    //   замість об'єкта базового класу
                    //   підставляється об'єкт похідного

    return 0;
}

```

В останньому випадку працює принцип підстановки: праворуч знаку присвоєння вказано об'єкт похідного класу, який підставляється на місце аргументу базового класу в операції присвоєння базового класу. Перетворення типу при цьому вказувати не потрібно – воно відбувається автоматично.

Проте не можна виконати присвоєння

```
d1 = b2;          // похідний = базовий
```

Для того, щоб можна було виконувати вказане присвоєння, слід в похідному класі визначити ще одну операцію присвоєння, яка буде приймати аргумент базового класу:

```

class Derive: public Base
{
public:
    Derive& operator = (const Derive &d);
    Derive& operator = (const Base &b);
};

```

Визначення цієї операції має бути таким:

```

Derive& Derive::operator = (const Base &b)
{
    this->Base::operator = (b); // виклик операції базового класу
    // ...
    return *this;
}

```

В тілі такої операції виконується виклик функціональної форми операції базового класу – це подібно до операції перетворення об'єктів базового класу в похідний.

Для будь-якого класу операцію присвоєння можна перевантажувати багато разів. При цьому допускається, щоб ні аргумент, ні результат не були того класу, в якому визначається така операція присвоєння. Єдиним обмеженням є видимість необхідних класів в точці визначення операції присвоєння.

Розщеплення та зрізування

Якщо в похідному класі визначено додаткові поля, то при підстановці об'єктів відбувається *зрізування* (*розщеплення*): об'єкту базового класу будуть присвоєні лише успадковані похідним класом поля:

```
class Point2
{
    int x, y;

public:
    // ...
};

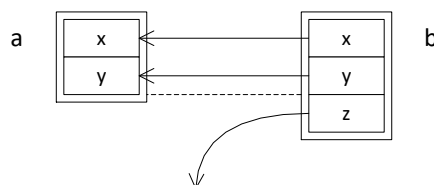
class Point3: public Point2
{
    int z;

public:
    // ...
};

int main()
{
    Point2 a;
    Point3 b;

    a = b;    // підстановка в присвоєнні
              // - розщеплення та зрізування

    return 0;
}
```



Зрізування та розщеплення може відбутися лише при передаванні параметрів за значенням та при присвоєнні об'єктів, коли похідний клас визначає власні поля.

При передаванні параметрів за посиланням (чи за вказівником), а також при присвоєнні вказівників, – зрізування та розщеплення не відбувається.

Дружні функції при успадкуванні

Дружні функції не успадковуються, оскільки не є методами базового класу, хоча і мають доступ до внутрішньої структури класу. При відкритому успадкуванні можна не повторювати дружні функції для похідного класу, оскільки принцип підстановки забезпечує підстановку аргументів похідного класу на місце параметрів базового класу.

Наприклад, можна не повторювати функцію-операцію виводу, якщо виведення об'єктів похідного класу не зміниться:

```
class Base
{
public:
    friend ostream& operator << (ostream &out, const Base &b);
};

ostream& operator << (ostream &out, const Base &b)
{
    out << b;
    return out;
}
```

Відкрите успадкування – успадкування інтерфейсу

Просте відкрите успадкування означає, що *похідний клас є різновидом (підмножиною) базового класу*: кожний об'єкт похідного класу також є об'єктом базового класу. При відкритому успадкуванні інтерфейс базового класу буде доступний програмі-клієнту, яка використовує похідний клас: доступні елементи базового класу залишатимуться доступними у похідному класі. Таким чином, відкрите успадкування – це успадкування інтерфейсу.

Згідно принципу підстановки, скрізь, де можна використовувати об'єкт базового класу, замість нього можна підставляти об'єкт похідного класу. Цей принцип працює і для посилань, і для вказівників: замість посилання (вказівника) на об'єкт базового класу можна підставити посилання (вказівник) на об'єкт похідного класу. Перетворення типу при цьому вказувати не потрібно – воно виконується автоматично. Таке приведення типу називається *приведення, що підвищує* або *приведення вгору по ієрархії (upcasting)*.

Закрите успадкування. Делегування

Закрите успадкування – це *успадкування реалізації* (на відміну від успадкування інтерфейсу, яке здійснюється при відкритому успадкуванні): похідний клас *реалізується* за допомогою базового класу.

Просте закрите успадкування описується наступною конструкцією:

```
class ім'я_похідного_класу: private ім'я_базового_класу
{
    тіло_похідного_класу;
};
```

Закрите успадкування принципово відрізняється від відкритого: принцип підстановки не виконується. Це означає, що не можна присвоїти (без явного перетворення типу) об'єкт похідного класу базовому. Закрите успадкування зазвичай використовують, коли потрібно отримати функціональність базового класу, але не потрібні ні копіювання, ні присвоєння.

Делегування – це передоручення обов’язків виконати роботу від класу-контейнеру до класу-елементу контейнера. Делегування реалізується так: Клас-елемент контейнера містить певний метод. Клас-контейнер містить однойменний метод, який викликає відповідний метод класу-елемента контейнера.

При закритому успадкуванні всі успадковані елементи базового класу стають приватними та недоступними програмі-клієнту:

```
class A
{
public:
    void f1() { /*...*/ }
    void f2() { /*...*/ }
};

class B: private A // успадковані методи недоступні клієнту
{
};
```

Програма, яка використовує клас B, не може викликати ні метод f1(), ні метод f2(). Тобто, якщо в main() створити об’єкт класу B, то з цього об’єкта не можна буде викликати методи, успадковані від A.

Щоб надати програмі таку можливість, потрібно в похідному класі заново реалізувати необхідні методи, які будуть *делегувати* (тобто, передавати) виклик відповідним методам базового класу:

```
class B: private A // успадковані методи недоступні клієнту
{
public:
    void f1() { A::f1(); } // делегування
    void f2() { A::f2(); } // делегування
};
```

Префікс при делегуванні вказувати обов’язково, бо інакше виникне рекурсія.

Успадковані методи можна відкрити в похідному класі також за допомогою конструкції `using`, яка має наступний синтаксис:

```
using ім'я_базового_класу::ім'я_елемента_базового_класу;
```

Для наведеного прикладу це буде виглядати так:

```
class B: private A // успадковані методи недоступні клієнту
{
public:
    using A::f1; // відкрили успадкований метод
    using A::f2; // відкрили успадкований метод
};
```

Таким чином, закрите успадкування обмежує функціональність, яку надає базовий клас; проте це обмеження можна зняти за допомогою *делегування* (передавання виклику) або конструкції `using`.

Закрите успадкування та композиція

Розглянемо попередній приклад з делегуванням:

```
class A
{
public:
    void f1() { /*...*/ }
    void f2() { /*...*/ }
};

class B: private A // успадковані методи недоступні клієнту
{
public:
    void f1() { A::f1(); } // делегування
    void f2() { A::f2(); } // делегування
};
```

При закритому успадкуванні ні структура (поля), ні поведінка (методи) базового класу не доступні програмі, яка використовує похідний клас. Лише за допомогою створення методів-оболонок, які *делегують* (тобто, передають) виклики відповідним методам базового класу, можна забезпечити доступ до функціональності базового класу (ще один спосіб, використання конструкції `using`, описаний в попередньому параграфі, тут не розглядається).

Перепишемо цей приклад з використанням композиції:

```
class A
{
public:
    void f1() { /*...*/ }
    void f2() { /*...*/ }
};

class B
{
    A a; // закрите поле-об'єкт класу A
public:
    void f1() { a.f1(); } // делегування
    void f2() { a.f2(); } // делегування
};
```

Зміст при такій організації зв'язків між класами **A** та **B** – той самий, що і при закритому успадкуванні класу **B** від класу **A**: клас **B** містить (успадковує) реалізацію класу **A**, але не надає програмі-клієнту його інтерфейсу.

Таким чином, закрите успадкування – це різновид композиції.

Закрите успадкування – успадкування реалізації

При закритому успадкуванні *похідний клас реалізується за допомогою базового*, тобто, – закрите успадкування – це *успадкування реалізації*.

Оскільки принцип підстановки при закритому успадкуванні не виконується, то не можна присвоїти (без явного перетворення типу) об'єкт похідного класу базовому. Тому

закрите успадкування слід використовувати в тих випадках, коли потрібно зберегти функціональність базового класу, але не потрібні ні присвоєння, ні копіювання.

Патерн Adapter (адаптер об'єктів)

Таке використання закритого успадкування – один із варіантів реалізації патерну Adapter (*адаптер об'єктів*). Цей патерн потрібний для того, щоб використовувати функціональність деякого класу, але в новому інтерфейсі: патерн Adapter перетворює інтерфейс одного класу до іншого інтерфейсу, якого очікує клієнт.



Він забезпечує сумісну роботу кількох класів (клієнт та наявна система), неможливу в звичних умовах із-за несумісності інтерфейсів.

Для нашого випадку: адаптуємо наявний клас (базовий при закритому успадкуванні чи агрегований при композиції) до нових потреб.

При вивченні попередньої теми розглядався приклад – клас-оболонка для стеку (ООП. Тема 02. Конструктори та перевантаження операцій. Клас-оболонка для стеку. – С. 66).

Наведемо його ще раз:

```
class Stack                                // універсальний стек
{
    struct Elem                            // елемент стеку
    {
        Elem *next;
        void *data;
        Elem (Elem *n, void *d)           // конструктор елемента стеку
            : data(d), next(n) {}
    };

    Elem *head;                            // вершина стеку
    Stack(const Stack &);                  // закрили копіювання
    Stack& operator = (const Stack &);    // закрили присвоєння

public:
    Stack() : head(0) {}                  // конструктор - ініціалізує
                                          // вказівник на вершину нулем

    void* Top() const                      // значення з вершини стеку
    {
        return Empty() ? 0 : head->data;
    }

    bool Empty() const                    // стек порожній, якщо
    {                                     // вершина == 0
        return head == 0;
    }
}
```

```

void Push(void *d)                // додати елемент
{
    head = new Elem(head, d);
}

void* Pop()                        // вилучити елемент
{
    if ( Empty() ) return 0;      // якщо стек - порожній,
                                  // то нічого не робити

    void *top = head->data;        // зберегли для повернення
    Elem *old = head;             // запам'ятали вказівник
    head = head->next;             // пересунули вершину
    delete old;                  // звільнили пам'ять

    return top;                   // повернули значення
                                  // елемента

unsigned int Count() const
{
    Elem *tmp = head;
    int count = 0;
    while (tmp)
    {
        count++;
        tmp = tmp->next;
    }

    return count;
}

friend ostream& operator << (ostream& out, const Stack& s);
friend istream& operator >> (istream& in, Stack& s);
};

ostream& operator << (ostream& out, const Stack& s)
{
    Stack::Elem *tmp = s.head;
    while (tmp)
    {
        out << *(int*)tmp->data << " "; // трактуємо data як int*
        tmp = tmp->next;
    }
    out << endl;

    return out;
}

istream& operator >> (istream& in, Stack& s)
{
    int value;
    cout << "value = "; in >> value;
    s.Push(new int(value));           // заносимо вказівник на int

    return in;
}

int main()
{
    {
        Stack s;

```



```

        cin >> s >> s >> s;
        cout << s << endl;
        cout << s.Count() << endl;

        while (s.Top())
            s.Pop();
    }

    return 0;
}

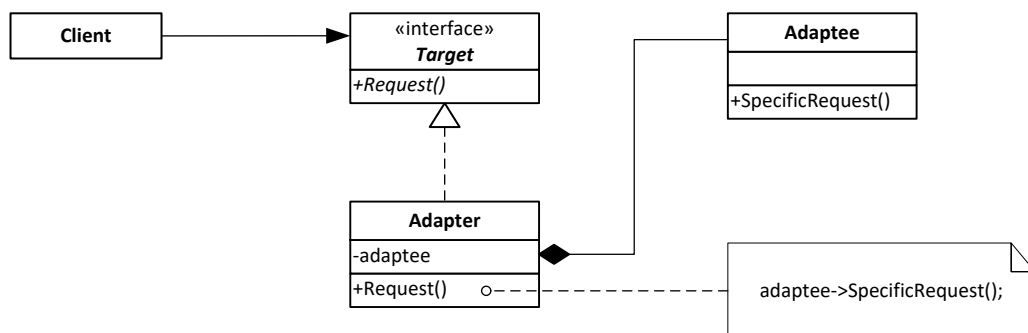
```

Така реалізація має ряд недоліків:

- 1) Не типізовані вказівники можуть бути джерелом помилок, оскільки перетворення до типу `*void` виконуються «мовчки». Наприклад, в стек, призначений для зберігання вказівників на числа, ненавмисне помилково можна записати вказівник на літерний рядок. Навіть гірше: до стеку може потрапити вказівник на локальну змінну, яка автоматично буде знищена при виході із свого блоку (помилка «підвішений вказівник», – вказівник на вже неіснуючу змінну, на звільнену область пам'яті).
- 2) Очевидний недолік такої універсальності – необхідність явного приведення типу при кожній вибірці вказівника із контейнера-стеку. Можна позбутися явного приведення типу за допомогою успадкування або композиції.

Використовуючи патерн **Adapter**, спеціалізуємо універсальний стек **Stack** – створимо клас **DoubleStack**, призначений для зберігання дійсних чисел (точніше, вказівників на них).

Патерн **Adapter** (*адаптер об'єктів*) має наступну будову:



Client – клієнт: програма, що взаємодіє з об'єктами, які задовольняють інтерфейсу **Target** (тут це – функція `main()`, яка очікує стек вказівників на дійсні числа);

Target – цільовий інтерфейс: визначає залежний від предметної області інтерфейс, яким користується і якого очікує **Client** (тему Інтерфейси будемо розглядати пізніше, в цій реалізації інтерфейсів не буде);

Adaptee – адаптований (клас, який ми адаптуємо до вимог клієнта): визначає інтерфейс, який потребує адаптації (в нашому випадку це – клас `Stack`);

Adapter – адаптер: адаптує інтерфейс **Adaptee** до інтерфейсу **Target** (в нашому випадку це – клас **DoubleStack**).

Клієнт бачить лише інтерфейс **Target**. Адаптер реалізує інтерфейс **Target**. Адаптер зв'язується із адаптованим об'єктом за допомогою композиції. Всі виклики клієнта адаптер делегує адаптованому класу.

Існує два різновиди патерну **Adapter**: *адаптер об'єктів* та *адаптер класів*.

Тут використовується адаптер об'єктів. Адаптер класів потребує *множинного успадкування*, яке розглядається в наступному розділі цієї теми.

Результати застосування патерну *адаптер об'єктів*:

- дозволяє одному адаптеру **Adapter** працювати з багатьма адаптованими об'єктами **Adaptee**, тобто з самим **Adaptee** та його підкласами (якщо вони існують). Адаптер може додати нову функціональність одразу всім адаптованим об'єктам;
- затрудняє заміщення операцій класу **Adaptee**. Для цього потрібно утворити підклас **Adaptee** та заставити **Adapter** посилатися на цей підклас, а не на сам **Adaptee**.

Спеціалізуємо стек за допомогою композиції:

```
class DoubleStack                                // стек вказівників
{                                                  // на дійсні числа
    Stack stack;                                 // поле-об'єкт класу
                                                // універсальний стек

public:
    void Push(double *d)                        // помістити в стек
    {
        stack.Push(d);                          // делегування
    }

    double* Top() const                          // отримати значення
    {
        return (double *)stack.Top();           // з вершини стеку
                                                // делегування
    }

    double* Pop()                               // вилучити елемент
    {
        return (double *)stack.Pop();           // з вершини стеку
                                                // делегування
    }

    bool Empty() const                          // чи стек – порожній?
    {
        return stack.Empty();                   // делегування
    }
};
```

Спеціалізація універсального стеку за допомогою закритого успадкування – все аналогічно, лише делегування здійснюється шляхом виклику успадкованого методу:

```
class DoubleStack: private Stack                // стек вказівників
{                                              // на дійсні числа
public:
    void Push(double *d)                      // помістити в стек
    {
```

```

        Stack::Push(d);                // делегування - виклик
    }                                  // успадкованого методу

    double* Top() const                // отримати значення
    {                                  // з вершини стеку
        return (double *)Stack::Top(); // делегування - виклик
    }                                  // успадкованого методу

    double* Pop()                      // вилучити елемент
    {                                  // з вершини стеку
        return (double *)Stack::Pop(); // делегування - виклик
    }                                  // успадкованого методу

    bool Empty() const                // чи стек - порожній?
    {
        return Stack::Empty();        // делегування - виклик
    }                                  // успадкованого методу
};

```

Такий підхід не зручний: користувачеві все ще потрібно працювати з вказівниками (тепер – на дійсні числа):

```

DoubleStack t;

t.Push( new double(11) );             // добавили елементи
t.Push( new double(22) );             // до стеку
t.Push( new double(33) );

while ( ! t.Empty() )
{
    cout << *(t.Top()) << endl;       // без перетворення
    double *p = t.Pop();              // без перетворення
    delete p;
}

```

Перепишемо спеціалізацію стеку так, щоб приховати від користувача всі дії з вказівниками.

Спеціалізація стеку за допомогою композиції:

```

class StackDouble                    // стек дійсних чисел
{
    Stack stack;                    // поле-об'єкт класу
                                    // універсальний стек
public:
    void Push(const double &d)      // помістити в стек
    {
        double *p = new double(d); // вказівник на дійсне
        stack.Push(p);              // помістили в стек
    }

    double Top() const              // отримати значення
    {                                // з вершини стеку
        double *p = (double *)stack.Top();
        return *p;
    }

    double Pop()                    // вилучити елемент
    {                                // з вершини стеку

```

```

        double *p = (double *)stack.Pop(); // отримали вказівник
        double t = *p;                     // зберегли значення
        delete p;                          // звільнили пам'ять
        return t;                          // повернули значення
    }

    bool Empty() const                     // чи стек – порожній?
    {
        return stack.Empty();             // делегування
    }
};

```

Спеціалізація універсального стеку за допомогою закритого успадкування – знову все аналогічно, лише делегування здійснюється шляхом виклику успадкованого методу:

```

class StackDouble: private Stack           // стек дійсних чисел
{
public:
    void Push(const double &d)             // помістити в стек
    {
        double *p = new double(d);        // вказівник на дійсне
        Stack::Push(p);                   // помістили в стек
    }

    double Top() const                     // отримати значення
    {                                       // з вершини стеку
        double *p = (double *)Stack::Top();
        return *p;
    }

    double Pop()                          // вилучити елемент
    {                                       // з вершини стеку
        double *p = (double *)Stack::Pop(); // отримали вказівник
        double t = *p;                   // зберегли значення
        delete p;                        // звільнили пам'ять
        return t;                        // повернули значення
    }

    bool Empty() const                     // чи стек – порожній?
    {
        return Stack::Empty();           // делегування
    }
};

```

Ми сховали всередину методів не лише перетворення, а і вказівники. Тепер метод Push() сам організовує вказівник, який буде поміщений в стек, а метод Pop() звільняє пам'ять і повертає її системі. Робота з таким стеком для користувача стала простішою:

```

StackDouble t;

t.Push( 111 ); // добавили елементи
t.Push( 222 ); // до стеку
t.Push( 333 );

while ( !t.Empty() )
{
    double p = t.Pop(); // видаляємо елемент і звільняємо
    cout << p << endl; // пам'ять
}

```

Користувач, який використовує такий стек, навіть не здогадується, що працює з вказівниками! – це і є хороший стиль програмування.

Відмінності структур та об'єднань від класів

Структури – теж класи, всі елементи структур за умовчанням доступні (в класах – приватні); ключ успадкування за умовчанням – `public` (для класів – `private`), тобто для структур за умовчанням реалізується відкрите успадкування (а для класів – закрите).

Об'єднання представляють по суті структуру, в якій всі елементи зберігаються в одному і тому ж місці. Об'єднання можуть містити конструктори і деструктори, а також методи і дружні функції. Подібно структурам, елементи об'єднання за умовчанням доступні (`public`).

Обмеження на об'єднання в C++:

1. Об'єднання не може успадковувати будь-які інші класи.
2. Об'єднання не може використовуватися в якості базового класу.
3. Об'єднання не може мати віртуальні методи.
4. Об'єднання не може мати статичних полів.
5. Об'єднання не може мати в якості елемента будь-який об'єкт, що перевантажує операцію присвоєння `=`.
6. Ніякий об'єкт не може бути елементом об'єднання, якщо цей об'єкт має конструктор або деструктор.

Множинне успадкування

Механізм множинного успадкування з концептуальної точки зору є цілком природнім: якщо кожний із базових класів містить лише частину необхідних властивостей, то чому би не об'єднати ці властивості у їх спільному класі-нащадку? Адже і в реальному житті кожна дитина має двох батьків:



Множинне успадкування, реалізоване в C++, не позбавлене недоліків і часто є джерелом різних проблем. Недаремно в більш нових об'єктно-орієнтованих мовах `java` та `C#` множинне успадкування класів заборонене, дозволяється лише множинна реалізація інтерфейсів (інтерфейси будуть вивчатися в темі «Віртуальні функції. Поліморфізм»).

Множинне успадкування відрізняється від простого (одиначного) наявністю кількох базових класів, наприклад:

```
class A {};  
  
class B {};  
  
class C: public A, public B  
{};
```

Базові класи перелічуються через кому, їх кількість не обмежена. Ключ успадкування для кожного базового класу може бути різним: від одного класу можна успадковувати відкрито, від іншого – закрито, від третього – захищено.

При множинному успадкуванні виконується все те ж саме, що і при одиначному: похідний клас успадковує структуру (всі елементи даних – поля) та поведінку (всі методи) всіх своїх базових класів.

При цьому:

- 1) З конструкторами при множинному успадкуванні слід поступати так ж само, як і при одиначному: якщо в базових класах визначені конструктори, то і в похідному класі слід визначити конструктор. При цьому конструктори-ініціалізатори базових класів слід явно викликати в конструкторі ініціалізації похідного класу.
- 2) При відкритому множинному успадкуванні виконується принцип підстановки: замість об'єкта одного із базових класів можна підставити об'єкт похідного класу.

Неоднозначність

Основна проблема, яка виникає при множинному успадкуванні, – неоднозначність, яка проявляється у двох видах: (1) неоднозначність даних та (2) неоднозначність методів.

Розглянемо приклад неоднозначності даних:

```
class B1  
{  
    int x;  
protected:  
    int s;          // похідні класи можуть користуватися цим полем  
public:  
    B1()            // конструктор  
    {  
        x = 1;  
        s = 2;  
    }  
  
    void Print()    // вивід  
    {  
        cout << "B1::x = " << x << endl;  
        cout << "B1::s = " << s << endl;  
    }  
};
```

```

class B2
{
protected:
    string s;    // похідні класи можуть користуватися цим полем

public:
    B2()          // конструктор
    {
        s = "3";
    }

    void Print()  // вивід
    {
        cout << "B2::s = " << s << endl;
    }
};

class D: public B1, public B2
{
public:
    void Print()  // вивід - помилка: невідомо, якого саме поля s
    {
        cout << "D::s = " << s << endl;
    }
};

int main()
{
    D d;
    d.Print();

    return 0;
}

```

У цьому прикладі обидва базові класи: і клас **B1**, і клас **B2** містять захищене поле **s**. Клас **D** може мати доступ до кожного з цих полів – він успадковує всі поля, в тому числі обидва поля **s**. При трансляції методу **Print()** класу **D** виникне конфлікт імен: компілятор не зможе визначити, яке поле використовується.

Аналогічно з неоднозначністю методів: припустимо, що в базових класах **B1** та **B2** визначено метод **f()**:

```

class B1
{
public:
    void f()
    {
        cout << "B1::f()" << endl;
    }
};

class B2
{
public:
    void f()
    {
        cout << "B2::f()" << endl;
    }
};

```

```

class D: public B1, public B2
{};

int main()
{
    D d;
    d.f(); // помилка: невідомо, який саме метод f() викликаємо

    return 0;
}

```

Клас D успадкує обидві версії цього методу і при спробі викликати `f()` виникає помилка неоднозначності. Ця проблема достатньо легко вирішується за допомогою префіксів, які уточнюють доступ до області дії. Наприклад, для функції виводу `Print()` класу D слід явно вказати, яке поле ми бажаємо вивести: `B1::s` чи `B2::s`. Виклик функції теж можна записати з префіксом: `B1::f()` чи `B2::f()`.

Ситуація ще більше ускладнюється при «ромбовидному» успадкуванні:

Наприклад, для класів

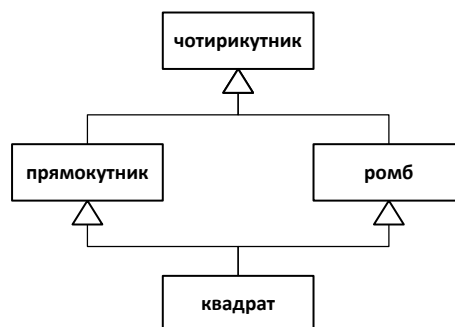
```

class A { /*...*/ };
class B1: public A { /*...*/ };
class B2: public A { /*...*/ };
class D: public B1, public B2 { /*...*/ };

```

– клас D отримує по 2 екземпляри усіх полів та методів спільного предка – класу A. В цьому легко переконатися за допомогою операції `sizeof()`.

Приклад «ромбовидного» успадкування:



Гірше, коли метод `f()` – віртуальний (віртуальні методи вивчаються в наступній темі «Віртуальні функції. Поліморфізм»):

```

class B1
{
public:
    virtual void f()
    {
        cout << "B1::f()" << endl;
    }
}

```



```

};

class B2
{
public:
    virtual void f()
    {
        cout << "B2.f()" << endl;
    }
};

class D: public B1, public B2
{};

int main()
{
    D d;
    d.f(); // помилка: невідомо, який саме метод f() викликаємо

    return 0;
}

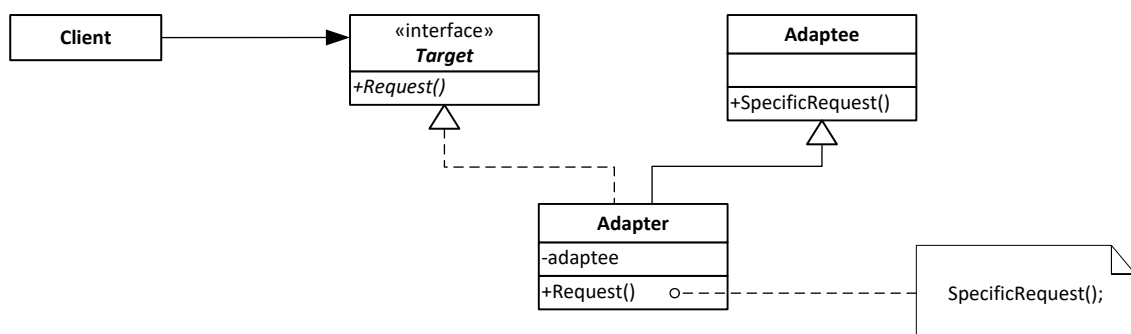
```

Якщо в похідному класі не визначено власного методу з таким ж самим прототипом, то виклик `d.f()`; буде неоднозначним, тобто виклик віртуального методу теж має бути записаний з префіксом: `d.B1::f()`; чи `d.B2::f()`; . Але при цьому виклик перестане бути віртуальним!

Бачимо, що множинне успадкування створює багато проблем, якщо його використовувати не знаючи механізму реалізації. Проте «вірне» використання буває надзвичайно корисним, наприклад – реалізація патерну **Adapter** (*адаптер класів*).

Патерн Adapter (адаптер класів)

Адаптер класу використовує множинне успадкування для адаптації одного інтерфейсу до іншого:



Ролі учасників – ті самі, що і у випадку *адаптера об'єктів* (див. стор. 35-36).

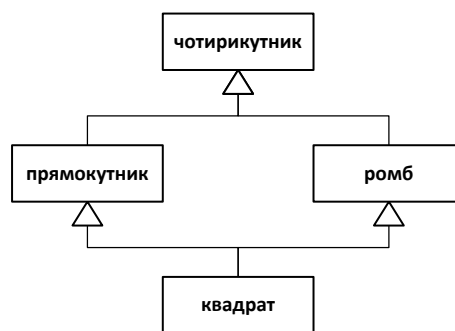
Клієнти викликають операції екземпляру класу **Adapter**. В свою чергу **Adapter** викликає операції адаптованого об'єкту чи класу **Adaptee**, який і виконує запит.

Результати застосування патерну *адаптер класів*:

- адаптує **Adaptee** до **Target**, передоручаючи дії конкретному класу **Adaptee**. Тому цей патерн не буде працювати, якщо ми захочемо одночасно адаптувати клас та його підкласи;
- дозволяє адаптеру **Adapter** замінити деякі операції адаптованого класу **Adaptee**, оскільки **Adapter** є підкласом **Adaptee**;
- вводить лише один новий об'єкт. Щоб добратися до адаптованого класу, не потрібно ніякого додаткового звертання за вказівником.

Віртуальне успадкування

Для усунення неоднозначності даних при ромбовидному успадкуванні в C++ реалізовано спеціальний механізм віртуального успадкування. Для випадку «ромба»



цей механізм записується так:

```

class A { /*...*/ };
class B1: virtual public A { /*...*/ };
class B2: virtual public A { /*...*/ };
class D: public B1, public B2 { /*...*/ };
  
```

– класи **B1** та **B2** віртуально успадковують від класу **A**. Клас **A** є для них *віртуальним базовим класом*. Віртуальне успадкування приводить до того, що клас **D** буде містити єдину копію полів класу **A**. Проте це не усуває неоднозначності полів класів **B1** та **B2**, а також неоднозначності їх методів.

При звичайному одиночному успадкуванні конструктор похідного класу завжди викликає конструктор базового класу для ініціалізації полів об'єкта, успадкованих від базового класу. При звичайному (не віртуальному) множинному успадкуванні виконується це ж правило. А ось при ромбовидному віртуальному успадкуванні виникає питання: який із класів (**B1** чи **B2**) відповідає за ініціалізацію єдиної копії полів спільного базового класу? – невже неоднозначність...? Тут неоднозначності немає.

Ця ситуація роз'яснюється правилом: віртуальний базовий клас ініціалізується *останнім похідним класом* в ієрархії. Це правило забезпечується компілятором.

Напишемо приклад ініціалізації віртуального базового класу:

```

class A
  
```

```

{
    int a;

public:
    A(const int a): a(a) {}           // конструктор ініціалізації
};

class B1: virtual public A
{
    int b;                           // неоднозначність з B2

public:
    B1(const int a, const int b): A(a) // ініціалізація предка
    {
        this->b = b;
    }
};

class B2: virtual public A
{
    int b;                           // неоднозначність з B1

public:
    B2(const int a, const int b): A(a) // ініціалізація предка
    {
        this->b = b;
    }
};

class D: public B1, public B2
{
    double y;

public:
    D(const int b1, const int b2, double y)
      : B1(0, b1), B2(0, b2)
    {
        this->y = y;                 // помилка трансляції
    }
};

```

Спроба відкомпілювати таку ієрархію класів приведе до помилки

error C2512: 'A::A' : no appropriate default constructor available

– немає підходящого конструктора за умовчанням в класі A.

Причина полягає в тому, що ми не написали виклику конструктора віртуального базового класу A в класі D – останньому похідному класі в ієрархії. Тому компілятор вважає, що в D слід «мовчки» викликати конструктор за умовчанням класу A. Але в класі A його немає – звідси і помилка! Таким чином, саме клас D відповідає за ініціалізацію класу A.

Правильна будова конструктора класу D має бути такою:

```

class D: public B1, public B2
{
    double y;

public:
    D(const int a, const int b1, const int b2, double y)

```

```

        : A(a), B1(0, b1), B2(0, b2) // ініціалізація віртуального
    {                               // базового класу
        this->y = y;
    }
};

```

Звичайно, власне поле `y` теж можна ініціалізувати в списку ініціалізації.

Зауважимо, що в конструкторах `B1` та `B2` перший аргумент (значення для поля `a` віртуального базового класу) дорівнює нулю. Це – фіктивний аргумент, оскільки в `D` явно викликається конструктор `A(a)`. Компілятор вірно все зробить, і поле `a` буде ініціалізоване першим аргументом конструктора `D`, а не нулем.

Принцип домінування

В деяких випадках (як і в попередньому прикладі), які виглядають неоднозначними, насправді неоднозначності немає:

```
class Up
{
public:
    virtual ~Up() {}
    virtual void Print()
    {
        cout << "Up" << endl;
    }
};

class Left: virtual public Up
{
public:
    virtual void Print()           // перевизначили метод Print
    {
        cout << "Left" << endl;
    }
};

class Right: virtual public Up    // метод Print - успадкований
{};

class Down: public Left, public Right
{};
```

Клас **Down** успадкував метод **Print()** безпосередньо від класу **Left**, а також метод **Print()** від класу **Up** «транзитом» через клас **Right**. Виглядає, начебто тут є неоднозначність:

```
int main()
{
    Down d;
    d.Print();

    return 0;
}
```

Проте згідно правила домінування, в цьому випадку буде викликатися метод **Print()** класу **Left**, – компілятор вибирає «найбільш похідний» клас, який і буде домінувати. Це правило працює лише при віртуальному успадкуванні.

Фінальний клас

Віртуальне успадкування дозволяє заборонити успадкування від цього класу. Це – потрібний та корисний інструмент, в мові `java` навіть введено ключове слово `final`. Якщо в `java` клас оголошено фінальним (з ключовим словом `final`), то від нього не можна успадковувати.

В `C++` немає спеціальних ключових слів, які забороняють успадкування, тому створення «фінального» класу ґрунтується на управлінні доступом до конструкторів та деструктору. Зокрема, якщо деструктор базового класу оголосити закритим, то не можна буде створити об'єкт ні базового, ні похідного класів.

Як вказано в [8, 26], використовуючи віртуальне успадкування, можна «заборонити» успадкування (а фактично – створення об'єктів похідного класу). Цю цікаву особливість виявив Ендрю Кенінг.

Приклад:

```
class Lock                                // клас, що закриває успадкування
{
    Lock() {}                             // закритий конструктор
    Lock(const Lock&) {}                  // закритий конструктор

    friend class Final;                   // нащадок - дружній клас

public:
    // ...                                інші елементи
};

class Final: public virtual Lock // "фінальний" клас
{
public:
    Final()
    {
        cout << "Final" << endl;
    }
};

class Derived: public Final              // формально не заборонено
{};

int main()
{
    Final f;                             // все працює

    Derived d;                            // помилка компіляції
    Derived *pd = new Derived();          // помилка компіляції

    return 0;
}
```

В наведеному прикладі головне – це ієрархія із двох класів:

- клас `Lock`, який використовується для «внутрішніх» потреб. Цей клас – віртуальна база, в якого конструктори – приватні. Він закриває успадкування;

- клас **Final**, віртуальний нащадок класу **Lock**, – фінальний клас, від якого «не можна» успадковувати.

Хоча успадкування від класу **Final** формально не заборонене, спроби створити об'єкт класу **Derived**, похідного від **Final**, приводять до помилок компіляції. Для об'єктів класу **Derived** виводиться повідомлення про відсутність доступу до приватного конструктора класу **Lock**: error C2248: 'Lock::Lock' : cannot access private member declared in class 'Lock'.

Розміри класів при множинному успадкуванні

Віртуальне успадкування додає в класи-нащадки віртуального базового класу додатково 4 байти.

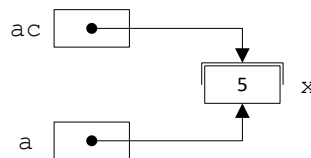
Явні перетворення типів в мові C++

Перетворення `const_cast<>`

Операція `const_cast`, як випливає з її імені, дозволяє зняти або додати константність:

```
typedef int A;  
A x = 5;  
  
// маємо  
const A *ac = &x;  
// або  
//A const *ac = &x;  
  
// *ac = 2; - помилка: не можна змінити константу  
  
// зняли константність:  
A *a = const_cast<A *>(ac);  
  
*a = 2; // все правильно
```

В наведеному прикладі визначається тип `A` – синонім типу `int`. Потім створюється змінна `x` типу `A`, яка отримує значення 5. Наступна команда створює вказівник `ac` – тип якого «вказівник на константу типу `A`». Спроба змінити значення комірки, на яку налаштований вказівник `ac`, приведе до помилки «не можна змінювати значення константи», – якщо таку спробу робити за допомогою адреси, записаної в `ac`. Далі оголошується вказівник `a`, який отримує значення вказівника `ac` при зняттю признаку константності. Остання команда демонструє, що тепер можна змінювати значення комірки, на яку налаштований вказівник `a`, – якщо це робити за допомогою адреси, записаної в `a`:



Для приведення констант застосовується лише оператор `const_cast`. Застосування будь-якого іншого оператора приведення типів у цьому випадку приведе до помилки при компіляції. Аналогічно, помилка при компіляції відбудеться у випадку використання оператора `const_cast` в команді, яка здійснює будь-які інші перетворення типів, крім створення або видалення константності.

Див. також «Тема 2. Конструктори та перевантаження операцій. Перевантаження операцій. Приклади та рекомендації щодо перевантаження операцій. Перевантаження операції приведення типу. Явні перетворення типу в C++. Операція `const_cast`».

Перетворення `static_cast<>`

Див. «Тема 2. Конструктори та перевантаження операцій. Перевантаження операцій. Приклади та рекомендації щодо перевантаження операцій. Перевантаження операції приведення типу. Явні перетворення типу в C++. Операція `static_cast`».

Перетворення `reinterpret_cast<>`

Див. «Тема 2. Конструктори та перевантаження операцій. Перевантаження операцій. Приклади та рекомендації щодо перевантаження операцій. Перевантаження операції приведення типу. Явні перетворення типу в C++. Операція `reinterpret_cast`».

Перетворення типів споріднених класів ієрархії `dynamic_cast<>`

Операція `dynamic_cast<>`

Операція використовується для перетворення вказівників та посилань на об'єкти споріднених класів ієрархії (зазвичай – вказівника на об'єкт базового класу у вказівник на об'єкт похідного класу), при цьому під час виконання перетворення виконується перевірка допустимості перетворення.

Формат операції:

```
dynamic_cast <тип *> (вираз)  
dynamic_cast <тип &> (вираз)
```

вираз – має бути вказівником або посиланням на об'єкт;

тип – базовий або похідний клас (або належати до тої ж самої ієрархії, що і клас *виразу*).

Після перевірки допустимості перетворення у випадку успішного виконання операція формує результат заданого типу, в іншому випадку для вказівника результат буде дорівнювати нулю, а для посилання генерується виняткова ситуація `bad_cast` (виняткові ситуації розглядаються при вивченні Теми 5. Опрацювання виняткових ситуацій). Якщо цільовий тип і тип виразу не відносяться до однієї ієрархії, то перетворення не допускається.

Перетворення із базового класу у похідний називають перетворенням *донизу* (downcast), – перетворення, яке *понижує* рівень ієрархії, оскільки при графічному поданні ієрархії прийнято зображувати похідні класи нижче (зокрема, і на UML-діаграмах класів). Перетворення із похідного класу у базовий називається перетворенням *догори* (upcast), – перетворення, яке *підвищує* рівень ієрархії, а перетворення між похідними класами одного базового класу, або між базовими класами одного похідного класу – *перехресним* (crosscast) перетворенням.

Перетворення, що підвищує рівень ієрархії

Це перетворення рівносильне простому присвоєнню, при якому виконується принцип підстановки – за умови відкритого успадкування:

```
class A  
{  
};  
  
class B: public A  
{  
};  
  
B *b = new B();  
A *a = dynamic_cast <A *> (b); // еквівалентно A *a = b;
```

Перетворення, що понижує рівень ієрархії

Похідні класи можуть мати елементи, яких немає у базовому. Для їх виклику через вказівник на об'єкт базового класу слід переконатися, що цей вказівник насправді вказує на об'єкт похідного класу. Така перевірка виконується в момент перетворення типу за допомогою RTTI (run-time type information) – інформації про тип під час виконання. Для того, щоб така перевірка могла бути виконана, аргумент операції `dynamic_cast<>` має бути вказівником чи посиланням на поліморфний об'єкт – тобто, об'єкт, який має хоча би один віртуальний метод (віртуальні методи та поліморфні об'єкти розглядаються при вивченні Тем 4. Віртуальні функції. Поліморфізм), справжній тип поліморфного об'єкту може відрізнятися від оголошеного типу.

Для поліморфного об'єкту реалізація операції `dynamic_cast<>` достатньо ефективна, бо посилання на інформацію про справжній тип об'єкта заноситься в таблицю віртуальних методів, і доступ до неї легко здійснити.

Результат застосування операції `dynamic_cast<>` до вказівника завжди слід перевіряти явно:

```
#include <iostream>
#include <typeinfo>
#include <Windows.h>           // забезпечення відображення кирилиці

using namespace std;

class B
{
public:
    virtual void f1() {}       // щоб зробити клас поліморфним
};

class D: public B
{
public:
    void f2() { cout << "f2" << endl; }
};

void demo(B *p)               // може бути поліморфним об'єктом
{
    D *d = dynamic_cast<D*>(p);
    if (d)                    // перевірка коректності dynamic_cast
        d->f2();
    else
        cerr << "Передали не об'єкт класу D" << endl;
}

int main()
{
    SetConsoleCP(1251);       // забезпечення відображення кирилиці
    SetConsoleOutputCP(1251); // забезпечення відображення кирилиці

    B *b = new B();
    demo(b);                  // виводить "Передали не об'єкт класу D"
```

```

        D *d = new D();
        demo(d);                                // виводить "f2" (все вірно)

        return 0;
    }

```

Перетворення посилань

Для аргумента-посилання зміст операції перетворення відрізняється від перетворення вказівника. Оскільки посилання завжди вказує на конкретний об'єкт, то операція `dynamic_cast<>` має виконувати перетворення саме до цього типу об'єкта. Коректність перетворення перевіряється автоматично; у випадку, коли фактичний та цільовий типи об'єкта – не збігаються, генерується виняток `bad_cast` (винятки розглядаються пізніше, при вивченні Теми 5. Опрацювання виняткових ситуацій):

```

#include <iostream>
#include <typeinfo>
#include <Windows.h>                                // забезпечення відображення кирилиці

using namespace std;

class B
{
public:
    virtual void f1() {}                            // щоб зробити клас поліморфним
};

class D: public B
{
public:
    void f2() { cout << "f2" << endl; }
};

void demo(B &p)                                    // може бути поліморфним об'єктом
{
    try
    {
        D &d = dynamic_cast<D&>(p);
        d.f2();
    }
    catch (bad_cast)
    {
        cerr << "Передали не об'єкт класу D" << endl;
    }
}

int main()
{
    SetConsoleCP(1251);                            // забезпечення відображення кирилиці
    SetConsoleOutputCP(1251);                      // забезпечення відображення кирилиці

    B *b = new B();
    demo(*b);                                        // виводить "Передали не об'єкт класу D"

    D *d = new D();
    demo(*d);                                        // виводить "f2" (все вірно)
}

```

```

    return 0;
}

```

Перехресне перетворення

Операція `dynamic_cast<>` дозволяє виконати безпечне перетворення між похідними класами одного базового класу:

```

#include <iostream>
#include <typeinfo>
#include <Windows.h>           // забезпечення відображення кирилиці
using namespace std;

class B
{
public:
    virtual void f1() {}       // щоб зробити клас поліморфним
};

class C: public B
{
public:
    void f2() { cout << "C::f2" << endl; }
};

class D: public B
{
public:
    void f2() { cout << "D::f2" << endl; }
};

void demo1(B *p) // може бути поліморфним об'єктом
{
    D *d = dynamic_cast<D*>(p);
    if (d)
        d->f2();
    else
        cerr << "Передали не об'єкт класу D" << endl;
}

void demo2(D *p) // оголошений тип параметра – той самий, що в аргумента
{               // справжній тип аргумента = class C

    dynamic_cast<C*>(p)->f2();

    D *d = dynamic_cast<D*>(p);
    if (d)
        d->f2();
    else
        cerr << "Передали не об'єкт класу D" << endl;
}

int main()
{
    SetConsoleCP(1251);       // забезпечення відображення кирилиці
    SetConsoleOutputCP(1251); // забезпечення відображення кирилиці

    B *b = new C();
    demo1((D*)b);              // виводить "Передали не об'єкт класу D"
    demo2((D*)b);              // виводить "C::f2" (все вірно)
                                // а потім "D::f2" (все вірно)
}

```

```

    return 0;
}

```

Вказівник для передавання у функції `demo1()` та `demo2()` сформований для демонстрації перехресного перетворення типів.

Можна здійснити перетворення між базовими класами одного похідного класу:

```

#include <iostream>
#include <typeinfo>
#include <Windows.h>           // забезпечення відображення кирилиці

using namespace std;

class B
{
public:
    virtual void f1() {}       // щоб зробити клас поліморфним
};

class C
{
public:
    virtual void f2()         // щоб зробити клас поліморфним
    {
        cout << "C::f2" << endl;
    }
};

class D: public B, public C {};

void demo(B *p)               // може бути поліморфним об'єктом
{
    C *c = dynamic_cast<C*>(p);
    if (c)
        c->f2();
    else
        cerr << "Передали не об'єкт класу C" << endl;
}

int main()
{
    SetConsoleCP(1251);       // забезпечення відображення кирилиці
    SetConsoleOutputCP(1251); // забезпечення відображення кирилиці

    B *d = new D();
    demo(d);                  // виводить "C::f2" (все вірно)

    return 0;
}

```

– якщо у функцію `demo()` передається насправді не вказівник на об'єкт класу `B`, а вказівник на об'єкт класу `D`, то його можна перетворити до його другого базового класу `C`.

Операція `static_cast<>`

Операція `static_cast<>` може виконувати перетворення із похідного класу до базового і навпаки:

```
class B
{};

class D: public B
{};

// ...

D *pd = new D();
B *pb = static_cast<B*>(pd); // похідний -> базовий

D d;
B &rb = static_cast<B&>(d); // похідний -> базовий

B b;
D &rd = static_cast<D&>(b); // базовий -> похідний
```

Таке перетворення виконується при компіляції, причому об'єкти можуть і не бути поліморфними. Програміст сам має відслідковувати допустимість таких перетворень.

В загальному випадку, для перетворення вказівників чи посилань на об'єкти споріднених класів ієрархії, краще використовувати операцію `dynamic_cast<>`.

Лабораторний практикум

Оформлення звіту про виконання лабораторних робіт

Вимоги до оформлення звіту про виконання лабораторних робіт №№ 3.1–3.6

Звіт про виконання лабораторних робіт №№ 3.1–3.6 має містити наступні елементи:

- 1) заголовок;
- 2) мету роботи;
- 3) умову завдання;

Умова завдання має бути вставлена у звіт як фрагмент зображення (скрін) сторінки посібника.

- 4) UML-діаграму класів;
- 5) структурну схему програми;

Структурна схема програми зображує взаємозв'язки програми та всіх її програмних одиниць: схему вкладеності та охоплення підпрограм, програми та модулів; а також схему звертання одних програмних одиниць до інших.

- 6) текст програми;

Текст програми має бути правильно відформатований: відступами і порожніми рядками слід відображати логічну структуру програми; програма має містити необхідні коментарі – про призначення підпрограм, змінних та параметрів – якщо їх імена не значущі, та про призначення окремих змістовних фрагментів програми. Текст програми слід подавати моноширинним шрифтом (Courier New розміром 10 пт. або Consolas розміром 9,5 пт.) з одинарним міжрядковим інтервалом;

- 7) посилання на git-репозиторій з проектом (див. інструкції з Лабораторної роботи № 2.2 з предмету «Алгоритмізація та програмування»);
- 8) хоча б для одної функції, яка повертає результат (як результат функції чи як параметр-посилання) – результати unit-тесту: текст програми unit-тесту та скрін результатів її виконання (див. інструкції з Лабораторної роботи № 5.6 з предмету «Алгоритмізація та програмування»);
- 9) висновки.

Зразок оформлення звіту про виконання лабораторних робіт №№ 3.1–3.6

ЗВІТ

про виконання лабораторної роботи № < номер >

« назва теми лабораторної роботи »

з дисципліни

«Об'єктно-орієнтоване програмування»

студента(ки) групи КН-27

< Прізвище Ім'я По_батькові >

Мета роботи:

...

Умова завдання:

...

UML-діаграма класів:

...

Структурна схема програми:

...

Текст програми:

...

Посилання на git-репозиторій з проектом:

...

Результати unit-тесту:

...

Висновки:

...

Лабораторна робота № 3.1. Відкрите успадкування

Мета роботи

Освоїти використання простого відкритого успадкування.

Питання, які необхідно вивчити та пояснити на захисті

- 1) Поняття та призначення успадкування.
- 2) Поняття базового / похідного класу, клас-предок / клас-нащадок.
- 3) Загальний синтаксис директиви успадкування.
- 4) Види успадкування.
- 5) Ключі успадкування та директиви доступу.
- 6) Просте успадкування.
- 7) Відкрите успадкування.

Зразок виконання завдання

Подається лише умова завдання та текст програми.

Варіант 0.

Умова завдання

У всіх завданнях реалізувати:

- операції вводу / виводу,
- методи отримання і
- методи встановлення значень полів, а також необхідні
- конструктори (якщо це не вказано в завданні явно);
- перетворення до літерного рядку реалізувати у вигляді операції приведення типу.

Конструктори і методи обов'язково мають перевіряти параметри на допустимість; у разі неправильних даних – виводити повідомлення про помилку і завершувати роботу.

У всіх завданнях реалізувати функцію, що одержує і повертає об'єкти базового класу. Продемонструвати принцип підстановки.

Для демонстрації роботи з об'єктами нового типу у всіх завданнях потрібно написати головну функцію. У програмі мають бути реалізовані різні способи створення об'єктів і масивів об'єктів. Програма має демонструвати використання всіх функцій та методів.

Створити клас **Pair** (пара цілих чисел); визначити метод множення на число і операцію додавання пар

$$(a, b) + (c, d) = (a + c, b + d).$$

Визначити похідний клас **Money** з полями: гривні і копійки. Перевизначити операцію додавання і визначити методи віднімання і ділення грошових сум.

Текст програми

```
// Лабораторна робота № 3.1
// Source.cpp

#include "Money.h"

using namespace std;

int main()
{
    Pair a, b, c;
    Money x(1, 2), y, z(x);

    cout << x << endl;

    return 0;
}

// Pair.h

#pragma once
#include <iostream>
#include <string>

using namespace std;

class Pair
{
    int a, b;
public:
    Pair(const int x = 0, const int y = 0);
    Pair(const Pair& r);
    ~Pair(void);

    int getA() const { return a; }
    int getB() const { return b; }

    void setA(const int value) { a = value; }
    void setB(const int value) { b = value; }

    friend Pair operator + (const Pair& l, const Pair& r);
    friend Pair operator * (const Pair& l, const int k);
    friend Pair operator * (const int k, const Pair& r);

    operator string() const;
    friend ostream& operator << (ostream& out, const Pair& r);
    friend istream& operator >> (istream& in, Pair& r);
};

// Pair.cpp
```

```

#include "Pair.h"
#include <sstream>

Pair::Pair(const int x, const int y)
    : a(x), b(y)
{}

Pair::Pair(const Pair& r)
{
    a = r.a;
    b = r.b;
}

Pair::~Pair(void)
{}

Pair operator + (const Pair& l, const Pair& r)
{
    Pair t;
    t.a = l.a + r.a;
    t.b = l.b + r.b;
    return t;
}

Pair operator * (const Pair& l, const int k)
{
    Pair t;
    t.a = l.a * k;
    t.b = l.b * k;
    return t;
}

Pair operator * (const int k, const Pair& r)
{
    Pair t;
    t.a = r.a * k;
    t.b = r.b * k;
    return t;
}

Pair::operator string() const
{
    stringstream ss;
    ss << "(" << getA() << ", " << getB() << " )";
    return ss.str();
}

ostream& operator << (ostream& out, const Pair& r)
{
    return out << (string)r;
}

istream& operator >> (istream& in, Pair& r)
{
    int a, b;
    cout << "a = "; in >> a;
    cout << "b = "; in >> b;
    r.setA(a); r.setB(b);
    return in;
}

// Money.h

```

```

#include "Pair.h"

class Money :
    public Pair
{
public:
    Money(const int x = 0, const int y = 0);
    Money(const Money& r);
    Money(double z);
    ~Money(void);

    friend Money operator + (const Money& l, const Money& r);
    friend Money operator * (const Money& l, const int k);
    friend Money operator * (const int k, const Money& r);

    friend Money operator - (const Money& l, const Money& r);
    friend double operator / (const Money& l, const Money& r);

    operator string() const;
    friend ostream& operator << (ostream& out, const Money& r);
    friend istream& operator >> (istream& in, Money& r);
};

// Money.cpp

#include "Money.h"
#include <sstream>

Money::Money(const int x, const int y)
{
    int a = x;
    int b = y;

    while (b > 99)
    {
        b -= 100;
        a++;
    }

    while (b < -99)
    {
        b += 100;
        a--;
    }

    setA(a);
    setB(b);
}

Money::Money(double z)
{
    int a = (int)z;
    z -= a;
    z *= 100;
    int b = (int)z;

    while (b > 99)
    {
        b -= 100;
        a++;
    }
}

```

```

        while (b < -99)
        {
            b += 100;
            a--;
        }
        setA(a);
        setB(b);
    }

Money::Money(const Money& r)
{
    int a = r.getA();
    int b = r.getB();

    while (b > 99)
    {
        b -= 100;
        a++;
    }

    while (b < -99)
    {
        b += 100;
        a--;
    }
    setA(a);
    setB(b);
}

Money::~Money(void)
{}

Money operator + (const Money& l, const Money& r)
{
    Money t;

    int a = l.getA() + r.getA();
    int b = l.getB() + r.getB();

    if (b > 99)
    {
        b -= 100;
        a++;
    }
    t.setA(a);
    t.setB(b);

    return t;
}

Money operator * (const Money& l, const int k)
{
    Money t;

    int a = l.getA() * k;
    int b = l.getB() * k;

    while (b > 99)
    {
        b -= 100;
        a++;
    }
}

```

```

        t.setA(a);
        t.setB(b);

        return t;
    }

Money operator * (const int k, const Money& r)
{
    Money t;

    int a = r.getA() * k;
    int b = r.getB() * k;

    while (b > 99)
    {
        b -= 100;
        a++;
    }
    t.setA(a);
    t.setB(b);

    return t;
}

Money operator - (const Money& l, const Money& r)
{
    Money t;

    int a = l.getA() - r.getA();
    int b = l.getB() - r.getB();

    while (b > 99)
    {
        b -= 100;
        a++;
    }

    while (b < -99)
    {
        b += 100;
        a--;
    }
    t.setA(a);
    t.setB(b);

    return t;
}

double operator / (const Money& l, const Money& r)
{
    double z1 = l.getA() + 0.01 * l.getB();
    double z2 = r.getA() + 0.01 * r.getB();

    return z1/z2;
}

string format(int b)
{
    stringstream ss;
    if (b < 10)
        ss << "0";
    ss << b;
}

```

```

        return ss.str();
    }

Money::operator string() const
{
    stringstream ss;
    ss << getA() << "," << format(getB()) << " hrn.";
    return ss.str();
}

ostream& operator << (ostream& out, const Money& r)
{
    return out << (string)r;
}

istream& operator >> (istream& in, Money& r)
{
    int a, b;
    cout << "hrn = "; in >> a;
    cout << "kop = "; in >> b;
    r.setA(a); r.setB(b);
    return in;
}

```

Варіанти завдань

У всіх завданнях реалізувати:

- операції вводу / виводу,
- методи отримання і
- методи встановлення значень полів, а також необхідні
- конструктори (якщо це не вказано в завданні явно);
- перетворення до літерного рядку реалізувати у вигляді операції приведення типу.

Конструктори і методи обов'язково мають перевіряти параметри на допустимість; у разі неправильних даних – виводити повідомлення про помилку і завершувати роботу.

У всіх завданнях потрібно реалізувати по два-три класи. Один клас є базовим, всі решту – похідні.

Визначення класів та реалізації методів слід розмістити в окремих модулях.

У всіх завданнях реалізувати функцію, що одержує і повертає об'єкти базового класу. Продемонструвати принцип підстановки.

Для демонстрації роботи з об'єктами нового типу у всіх завданнях потрібно написати головну функцію. У програмі мають бути реалізовані різні способи створення об'єктів і масивів об'єктів. Програма має демонструвати використання всіх функцій та методів.

Варіант 1.

Створити базовий клас **Car** (машина), що характеризується торговою маркою

(літерний рядок), кількістю циліндрів, потужністю. Визначити методи пере-присвоєння та зміни потужності.

Створити похідний клас **Lorry** (вантажівка), що характеризується також вантажопідйомністю кузова. Визначити функції пере-присвоєння марки та зміни вантажопідйомності.

Варіант 2.

Створити клас **Pair** (пара чисел); визначити методи зміни полів і порівняння пар: пара $p1$ більше пари $p2$, якщо

$(p1.first > p2.first)$ або $(p1.first = p2.first)$ і $(p1.second > p2.second)$.

Визначити похідний клас **Fraction**, пара чисел трактується як ціла частина числа і дробова частина числа. Визначити повний набір методів порівняння.

Варіант 3.

Створити клас **Liquid** (рідина), що має поля «назва» і «густина». Визначити методи перепризначення і зміни густини.

Створити похідний клас **Alcohol** (спирт), що має поле «міцність». Визначити методи пере-присвоєння та зміни міцності.

Варіант 4.

Створити клас **Pair** (пара чисел); визначити методи зміни полів та обчислення добутку чисел.

Визначити похідний клас **Rectangle** (прямокутник), пара чисел трактується як сторони. Визначити методи обчислення периметру та площі прямокутника.

Варіант 5.

Створити клас **Man** (людина) з полями: ім'я, вік, стать і вага. Визначити методи пере-присвоєння імені, зміни віку і зміни ваги.

Створити похідний клас **Student**, що має поле «рік навчання». Визначити методи пере-присвоєння та збільшення року навчання.

Варіант 6.

Створити клас **Triad** (трійка чисел); визначити методи зміни полів і обчислення суми чисел.

Визначити похідний клас **Triangle** (трикутник), трійка чисел трактується як сторони. Визначити методи обчислення кутів і площі трикутника.

Варіант 7.

Створити клас `Triangle` (трикутник) з полями-сторонами. Визначити методи зміни сторін, обчислення кутів, обчислення периметру.

Створити похідний клас `Equilateral` (рівносторонній трикутник), що має поле «площа». Визначити метод обчислення площі.

Варіант 8.

Створити клас `Triangle` (трикутник) з полями-сторонами. Визначити методи зміни сторін, обчислення кутів, обчислення периметру.

Створити похідний клас `RightAngled` (прямокутний трикутник), що має поле «площа». Визначити метод обчислення площі.

Варіант 9.

Створити клас `Pair` (пара чисел); визначити методи зміни полів і обчислення добутку чисел.

Визначити похідний клас `RightAngled` (прямокутний трикутник), пара чисел трактується як катети. Визначити методи обчислення гіпотенузи і площі трикутника.

Варіант 10.

Створити клас `Triad` (трійка чисел); визначити метод порівняння тріад:
тріада `t1` більше тріади `t2`, якщо

`(t1.first > t2.first) або (t1.first = t2.first) і (t1.second > t2.second)`
або `(t1.first = t2.first) і (t1.second = t2.second) і (t1.third > t2.third)`.

Визначити похідний клас `Date`, трійка чисел трактується як рік, місяць і день. Визначити повний набір методів порівняння дат.

Варіант 11.

Створити клас `Triad` (трійка чисел); визначити метод порівняння тріад:
тріада `t1` більше тріади `t2`, якщо

`(t1.first > t2.first) або (t1.first = t2.first) і (t1.second > t2.second)`
або `(t1.first = t2.first) і (t1.second = t2.second) і (t1.third > t2.third)`.

Визначити похідний клас `Time`, трійка чисел трактується як година, хвилина і секунда. Визначити повний набір методів порівняння моментів часу.

Варіант 12.

Реалізувати клас-оболонку `Number` для числового типу `float`. Реалізувати методи

додавання і ділення.

Створити похідний клас **Real**, в якому реалізувати метод піднесення до довільного степеня, та метод для обчислення логарифму числа.

Варіант 13.

Створити клас **Triad** (трійка чисел); визначити методи збільшення полів на 1.

Визначити похідний клас **Date**, трійка чисел трактується як рік, місяць і день. Перевизначити методи збільшення полів на 1 і визначити метод збільшення дати на n днів.

Варіант 14.

Реалізувати клас-оболонку **Number** для числового типу **double**. Реалізувати методи множення і віднімання.

Створити похідний клас **Real**, в якому реалізувати метод, що обчислює корінь довільного степеня, і метод для обчислення числа π в заданій степені.

Варіант 15.

Створити клас **Triad** (трійка чисел); визначити методи збільшення полів на 1.

Визначити клас-нащадок **Time**, трійка чисел трактується як година, хвилина, секунда. Перевизначити методи збільшення полів на 1 і визначити методи збільшення часу на n секунд і хвилин.

Варіант 16.

Створити базовий клас **Pair** (пара цілих чисел) з операціями перевірки на рівність і множення полів. Реалізувати операцію віднімання пар за формулою

$$(a, b) - (c, d) = (a - c, b - d).$$

Створити похідний клас **Rational**; визначити нові операції:

додавання

$$(a, b) + (c, d) = (ad + bc, bd)$$

і ділення

$$(a, b) / (c, d) = (ad, bc);$$

перевизначити операцію віднімання

$$(a, b) - (c, d) = (ad - bc, bd).$$

Варіант 17.

Створити клас **Pair** (пара чисел); визначити метод множення полів та операцію додавання пар

$$(a, b) + (c, d) = (a + c, b + d).$$

Визначити похідний клас **Complex**, пара чисел трактується як дійсна і мніма частини числа. Визначити методи множення

$$(a, b) \times (c, d) = (ac - bd, ad + bc)$$

і віднімання

$$(a, b) - (c, d) = (a - c, b - d).$$

Варіант 18.*

Створити клас **Pair** (пара цілих чисел); визначити методи зміни полів і операцію додавання пар

$$(a, b) + (c, d) = (a + c, b + d).$$

Визначити похідний клас **Long**, пара чисел трактується як старша частина числа і молодша частина числа. Перевизначити операцію додавання і визначити методи множення і віднімання.

Варіант 19.

Створити базовий клас **Triad** (трійка чисел) з операціями: додавання числа, множення на число, перевірки на рівність.

Створити похідний клас **vector3D**, трійка чисел трактується як координати. Мають бути реалізовані: операція додавання векторів, скалярний добуток векторів.

Варіант 20.

Створити клас **Pair** (пара цілих чисел); визначити метод множення на число і операцію додавання пар

$$(a, b) + (c, d) = (a + c, b + d).$$

Визначити похідний клас **Fraction**, пара чисел трактується як ціла частина числа і дробова частина числа. Перевизначити операцію додавання і визначити методи віднімання і ділення величин типу **Fraction**.

Варіант 21.

Створити клас **Car** (машина), що характеризується торговою маркою (літерний рядок), кількістю циліндрів, потужністю. Визначити методи пере-присвоєння та зміни потужності.

Створити похідний **Bus** (автобус), що характеризується також кількістю пасажирських місць. Визначити функції пере-присвоєння марки та зміни кількості пасажирських місць.

Варіант 22.

Створити клас **Pair** (пара чисел); визначити методи зміни полів і порівняння пар: пара **p1** більше пари **p2**, якщо

$(p1.first > p2.first)$ або $(p1.first = p2.first)$ і $(p1.second > p2.second)$.

Визначити похідний **Rational** (дріб), пара чисел трактується як чисельник і знаменник. Визначити повний набір методів порівняння.

Варіант 23.

Створити клас **Liquid** (рідина), що має поля «назва» і «густина». Визначити методи перепризначення і зміни густини.

Створити похідний **Solution** (розчин), що має поле «відносна кількість речовини». Визначити методи пере-присвоєння та зміни відносної кількості речовини.

Варіант 24.

Створити клас **Pair** (пара чисел); визначити методи зміни полів та обчислення добутку чисел.

Визначити похідний **Ellipse** (еліпс), пара чисел трактується як пів-осі. Визначити методи обчислення периметру та площі еліпсу.

Варіант 25.

Створити клас **Man** (людина) з полями: ім'я, вік, стать і вага. Визначити методи пере-присвоєння імені, зміни віку і зміни ваги.

Створити похідний **Student**, що має поле «курс». Визначити методи пере-присвоєння та збільшення курсу.

Варіант 26.

Створити базовий клас **Car** (машина), що характеризується торговою маркою (літерний рядок), кількістю циліндрів, потужністю. Визначити методи пере-присвоєння та зміни потужності.

Створити похідний клас **Lorry** (вантажівка), що характеризується також вантажопідйомністю кузова. Визначити функції пере-присвоєння марки та зміни вантажопідйомності.

Варіант 27.

Створити клас **Pair** (пара чисел); визначити методи зміни полів і порівняння пар: пара **p1** більше пари **p2**, якщо

$(p1.first > p2.first)$ або $(p1.first = p2.first)$ і $(p1.second > p2.second)$.

Визначити похідний клас **Fraction**, пара чисел трактується як ціла частина числа і дробова частина числа. Визначити повний набір методів порівняння.

Варіант 28.

Створити клас **Liquid** (рідина), що має поля «назва» і «густина». Визначити методи перепризначення і зміни густини.

Створити похідний клас **Alcohol** (спирт), що має поле «міцність». Визначити методи пере-присвоєння та зміни міцності.

Варіант 29.

Створити клас **Pair** (пара чисел); визначити методи зміни полів та обчислення добутку чисел.

Визначити похідний клас **Rectangle** (прямокутник), пара чисел трактується як сторони. Визначити методи обчислення периметру та площі прямокутника.

Варіант 30.

Створити клас **Man** (людина) з полями: ім'я, вік, стать і вага. Визначити методи пере-присвоєння імені, зміни віку і зміни ваги.

Створити похідний клас **Student**, що має поле «рік навчання». Визначити методи пере-присвоєння та збільшення року навчання.

Варіант 31.

Створити клас **Triad** (трійка чисел); визначити методи зміни полів і обчислення суми чисел.

Визначити похідний клас **Triangle** (трикутник), трійка чисел трактується як сторони. Визначити методи обчислення кутів і площі трикутника.

Варіант 32.

Створити клас **Triangle** (трикутник) з полями-сторонами. Визначити методи зміни сторін, обчислення кутів, обчислення периметру.

Створити похідний клас **Equilateral** (рівносторонній трикутник), що має поле «площа». Визначити метод обчислення площі.

Варіант 33.

Створити клас **Triangle** (трикутник) з полями-сторонами. Визначити методи зміни

сторін, обчислення кутів, обчислення периметру.

Створити похідний клас `RightAngled` (прямокутний трикутник), що має поле «площа». Визначити метод обчислення площі.

Варіант 34.

Створити клас `Pair` (пара чисел); визначити методи зміни полів і обчислення добутку чисел.

Визначити похідний клас `RightAngled` (прямокутний трикутник), пара чисел трактується як катети. Визначити методи обчислення гіпотенузи і площі трикутника.

Варіант 35.

Створити клас `Triad` (трійка чисел); визначити метод порівняння тріад: тріада `t1` більше тріади `t2`, якщо

$(t1.first > t2.first)$ або $(t1.first = t2.first)$ і $(t1.second > t2.second)$
або $(t1.first = t2.first)$ і $(t1.second = t2.second)$ і $(t1.third > t2.third)$.

Визначити похідний клас `Date`, трійка чисел трактується як рік, місяць і день. Визначити повний набір методів порівняння дат.

Варіант 36.

Створити клас `Triad` (трійка чисел); визначити метод порівняння тріад: тріада `t1` більше тріади `t2`, якщо

$(t1.first > t2.first)$ або $(t1.first = t2.first)$ і $(t1.second > t2.second)$
або $(t1.first = t2.first)$ і $(t1.second = t2.second)$ і $(t1.third > t2.third)$.

Визначити похідний клас `Time`, трійка чисел трактується як година, хвилина і секунда. Визначити повний набір методів порівняння моментів часу.

Варіант 37.

Реалізувати клас-оболонку `Number` для числового типу `float`. Реалізувати методи додавання і ділення.

Створити похідний клас `Real`, в якому реалізувати метод піднесення до довільного степеня, та метод для обчислення логарифму числа.

Варіант 38.

Створити клас `Triad` (трійка чисел); визначити методи збільшення полів на 1.

Визначити похідний клас `Date`, трійка чисел трактується як рік, місяць і день. Перевизначити методи збільшення полів на 1 і визначити метод збільшення дати на n днів.

Варіант 39.

Реалізувати клас-оболонку **Number** для числового типу **double**. Реалізувати методи множення і віднімання.

Створити похідний клас **Real**, в якому реалізувати метод, що обчислює корінь довільного степеню, і метод для обчислення числа π в заданій степені.

Варіант 40.

Створити клас **Triad** (трійка чисел); визначити методи збільшення полів на 1.

Визначити клас-нащадок **Time**, трійка чисел трактується як година, хвилина, секунда. Перевизначити методи збільшення полів на 1 і визначити методи збільшення часу на n секунд і хвилин.

Лабораторна робота № 3.2. Просте успадкування

Мета роботи

Освоїти використання простого успадкування.

Питання, які необхідно вивчити та пояснити на захисті

- 1) Поняття та призначення успадкування.
- 2) Поняття базового / похідного класу, клас-предок / клас-нащадок.
- 3) Загальний синтаксис директиви успадкування.
- 4) Види успадкування.
- 5) Ключі успадкування та директиви доступу.
- 6) Просте успадкування.
- 7) Відкрите успадкування.

Варіанти завдань

У всіх завданнях реалізувати:

- операції вводу / виводу,
- методи отримання і
- методи встановлення значень полів, а також необхідні
- конструктори (якщо це не вказано в завданні явно);
- перетворення до літерного рядку реалізувати у вигляді операції приведення типу.

Конструктори і методи обов'язково мають перевіряти параметри на допустимість; у разі неправильних даних – виводити повідомлення про помилку і завершувати роботу.

У всіх завданнях реалізувати функцію, що одержує і повертає об'єкти базового класу. Продемонструвати принцип підстановки.

Для демонстрації роботи з об'єктами нового типу у всіх завданнях потрібно написати головну функцію. У програмі мають бути реалізовані різні способи створення об'єктів і масивів об'єктів. Програма має демонструвати використання всіх функцій та методів.

Варіант 1.*

Створити клас **Карта**, що має ранг і масть. Карту можна перевернути і відкрити. Створити клас – **Колода карт**, що містить карти. Створити два похідні класи від колоди карт, в одному карти можуть діставатися тільки підряд, в іншому – вибиратися довільно.

Варіант 2.

Створити клас **Студент**, що має ім'я (вказівник на рядок), курс та ідентифікаційний номер. Визначити конструктори, деструктор і функцію виведення. Створити **public**-похідний клас – **студент-дипломник**, що має тему дипломної роботи. Визначити конструктори за умовчанням та з різним числом параметрів, деструктор, функцію виведення. Визначити функції зміни назви дипломної роботи та ідентифікаційного номера.

Варіант 3.*

Створити клас **Кімната** з даними: ширина, довжина, висота. Визначити конструктор і метод доступу. Створити клас **Однокімнатної квартири**, яка містить кімнату і кухню, номер та поверх. Визначити конструктори, методи доступу. Визначити **public**-похідний клас **однокімнатних квартир різних міст** (додаткове поле – назва міста). Визначити конструктори, деструктор і функцію виведення.

Варіант 4.

Створити ієрархію класів **спортивна гра** і **футбол**. Перевизначити виведення у потік і введення з потоку, конструктор копіювання, операцію присвоєння через відповідні функції базового класу.

Варіант 5.

Створити клас **автомобіль**, який має марку (вказівник на рядок), колір, об'єм двигуна, потужність. Визначити конструктори, деструктор і функцію виведення. Створити **public**-похідний клас – **вантажівка**, що має вантажопідйомність кузова. Визначити конструктори за умовчанням і з різним числом параметрів, деструктор, функцію виведення. Визначити функції перепризначення марки і вантажопідйомності.

Варіант 6.*

Створити клас **двигун** із можливістю визначення потужності. Визначити конструктори і метод доступу до даних. Створити клас **машин**, що містить клас **двигун**. Додатково є марка (вказівник на рядок), ціна. Визначити конструктори і деструктор. Визначити **public**-похідний клас **вантажівка**, що має додатково вантажопідйомність. Визначити конструктори, деструктори і функцію виведення.

Варіант 7.

Створіть клас **точка**, що має координати. Визначити класи **еліпсів** і **кіл**. Визначити ієрархію типів. Визначити функції виведення, конструктори, деструктори, обчислення площі.

Варіант 8.

Створити класи транспортні засоби: автомобіль, вантажівка, пароплав і літак. Створити з них ієрархію. Визначити функції виведення, конструктори і деструктори.

Варіант 9.

Створити клас **рідина**, що має назву (вказівник на рядок), густину. Визначити конструктори, деструктор і функцію виведення даних. Створити **public**-похідний клас – **спиртні напої**, які мають міцність. Визначити конструктори за умовчанням і з різним числом параметрів, деструктори, функцію виведення. Визначити функції перепризначення густини і міцності.

Варіант 10.

Використовуючи ієрархію та успадкування, створити класи **вікна**, **вікна з заголовком** і **вікна з кнопкою**.

Варіант 11.

Створити ієрархію класів **людина**, **студент** і **студент-дипломник**. Перевизначити виведення у потік і введення з потоку, конструктор копіювання, оператор присвоювання через відповідні функції базового класу.

Варіант 12.

Створити клас **людина**, що має ім'я (вказівник на рядок), вік, вагу. Визначити конструктори, деструктор і функцію виведення. Створити **public**-похідний клас – **школяр**, який має рік навчання. Визначити конструктори за умовчанням і з різним числом параметрів, деструктори, функцію виведення. Визначити функції перепризначення віку і класу.

Варіант 13.

Створити клас **вікно**, що має координати верхнього лівого і нижнього правого кута, колір фону (вказівник на рядок). Визначити конструктори, деструктор і функцію виведення. Створити **public**-похідний клас – **вікно**, яке має рядок з меню. Визначити конструктори за умовчанням і з різним числом параметрів, деструктор, функцію виведення. Визначити функції перепризначення кольору фону і рядка меню.

Варіант 14.

Створити ієрархію класів **людина** і **службовець**. Перевизначити виведення у потік і введення з потоку, конструктор копіювання, операцію присвоєння через відповідні функції

базового класу.

Варіант 15.

Створити клас **точка**, що має координати. Визначити конструктори, деструктор і функцію виведення. Створити **public**-похідний клас – **кольорова точка**, що має колір. Визначити конструктори за умовчанням і з різним числом параметрів, деструктор, функцію виведення. Визначити функції перепризначення кольору і координат точки, виведення точки на екран.

Варіант 16.

Створити ієрархію класів **людина** і **студент**. Перевизначити виведення у потік і введення з потоку, конструктор копіювання, операцію присвоєння через відповідні функції базового класу.

Варіант 17.

Створити клас **людина**, що має ім'я (вказівник на рядок), вік, вагу. Визначити конструктори, деструктор і функцію виведення. Створити **public**-похідний клас – **повнолітній**, що має номер паспорта. Визначити конструктори за умовчанням і з різним числом параметрів, деструктори, функцію виведення. Визначити функції перепризначення віку і номера паспорта.

Варіант 18.

Створити ієрархію класів **вікно** і **вікно з заголовком**. Перевизначити виведення у потік і введення з потоку, конструктор копіювання, операцію присвоєння через відповідні функції базового класу.

Варіант 19.*

Створити ієрархію класів **вектор** та **безпечний вектор** з перевіркою виходу за межі. **Безпечний вектор** визначає змінні – нижню і верхню межу. Перевизначити виведення у потік і введення з потоку, конструктор копіювання, операцію присвоєння через відповідні функції базового класу.

Варіант 20.*

Створити клас **жорсткий диск**, що має обсяг (Мбайт). Визначити конструктори і метод доступу. Створити клас **комп'ютер**, що містить клас **жорсткий диск**. Додатково є марка (вказівник на рядок), ціна. Визначити конструктори і деструктор. Визначити **public**-

похідний клас комп'ютерів з монітором, що має додатково розмір діагоналі монітора. Визначити конструктори, деструктор і функцію виведення.

Варіант 21.*

Створити клас процесор, що має потужність (МГц). Визначити конструктори і метод доступу. Створити клас комп'ютер, що містить клас процесор. Додатково задається марка (вказівник на рядок), ціна. Визначити конструктори і деструктор. Визначити public-похідний клас комп'ютерів з монітором, що має додатково розмір діагоналі монітора. Визначити конструктори, деструктор і функцію виведення.

Варіант 22.*

Створити клас кнопка, що має розмір. Визначити конструктори і метод доступу. Створити клас вікно, який містить клас кнопка. Додатково задаються координати вікна. Визначити конструктори і деструктор. Визначити public-похідний клас вікно з кнопкою, яке має меню (вказівник на рядок). Визначити конструктори, деструктор і функцію виведення.

Варіант 23.***

Використовуючи ієрархію і композицію класів, створити бінарне дерево. Визначити методи обходу дерева. Вузол може бути розміщений у дерево тільки як кінцевий. Якщо дерево є порожнім, то створюється новий екземпляр класу вузол дерева і вузол розміщується у дерево. Якщо дерево не є порожнім, то програма порівнює значення, що включається в дерево, зі значенням у кореневому вузлі і якщо менше, то поміщає в ліві під-дерева, а якщо більше, то – в праві. Якщо значення рівні, то виводиться повідомлення, і вершина не включається в дерево.

Варіант 24.***

Створити базовий клас – черга з двома кінцями. Елементи можуть вилучатися і додаватися з будь-якої сторони. Створити похідні класи – стек і звичайна черга.

Варіант 25.***

Задано два класи – список List і масив Array. Побудувати класи стеку і черги на базі композиції та успадкування цих класів.

Варіант 26.*

Створити клас Карта, що має ранг і масть. Карту можна перевернути і відкрити.

Створити клас – Колода карт, що містить карти. Створити два похідні класи від колоди карт, в одному карти можуть діставатися тільки підряд, в іншому – вибиратися довільно.

Варіант 27.

Створити клас Студент, що має ім'я (вказівник на рядок), курс та ідентифікаційний номер. Визначити конструктори, деструктор і функцію виведення. Створити public-похідний клас – студент-дипломник, що має тему дипломної роботи. Визначити конструктори за умовчанням та з різним числом параметрів, деструктор, функцію виведення. Визначити функції зміни назви дипломної роботи та ідентифікаційного номера.

Варіант 28.*

Створити клас Кімната з даними: ширина, довжина, висота. Визначити конструктор і метод доступу. Створити клас Однокімнатної квартири, яка містить кімнату і кухню, номер та поверх. Визначити конструктори, методи доступу. Визначити public-похідний клас однокімнатних квартир різних міст (додаткове поле – назва міста). Визначити конструктори, деструктор і функцію виведення.

Варіант 29.

Створити ієрархію класів спортивна гра і футбол. Перевизначити виведення у потік і введення з потоку, конструктор копіювання, операцію присвоєння через відповідні функції базового класу.

Варіант 30.

Створити клас автомобіль, який має марку (вказівник на рядок), колір, об'єм двигуна, потужність. Визначити конструктори, деструктор і функцію виведення. Створити public-похідний клас – вантажівка, що має вантажопідйомність кузова. Визначити конструктори за умовчанням і з різним числом параметрів, деструктор, функцію виведення. Визначити функції перепризначення марки і вантажопідйомності.

Варіант 31.*

Створити клас двигун із можливістю визначення потужності. Визначити конструктори і метод доступу до даних. Створити клас машин, що містить клас двигун. Додатково є марка (вказівник на рядок), ціна. Визначити конструктори і деструктор. Визначити public-похідний клас вантажівка, що має додатково вантажопідйомність. Визначити конструктори, деструктори і функцію виведення.

Варіант 32.

Створіть клас `точка`, що має координати. Визначити класи `еліпсів` і `кіл`. Визначити ієрархію типів. Визначити функції виведення, конструктори, деструктори, обчислення площі.

Варіант 33.

Створити класи транспортні засоби: `автомобіль`, `вантажівка`, `пароплав` і `літак`. Створити з них ієрархію. Визначити функції виведення, конструктори і деструктори.

Варіант 34.

Створити клас `рідина`, що має назву (вказівник на рядок), густину. Визначити конструктори, деструктор і функцію виведення даних. Створити `public`-похідний клас – `спиртні напої`, які мають міцність. Визначити конструктори за умовчанням і з різним числом параметрів, деструктори, функцію виведення. Визначити функції перепризначення густини і міцності.

Варіант 35.

Використовуючи ієрархію та успадкування, створити класи `вікна`, `вікна з заголовком` і `вікна з кнопкою`.

Варіант 36.

Створити ієрархію класів `людина`, `студент` і `студент-дипломник`. Перевизначити виведення у потік і введення з потоку, конструктор копіювання, оператор присвоювання через відповідні функції базового класу.

Варіант 37.

Створити клас `людина`, що має ім'я (вказівник на рядок), вік, вагу. Визначити конструктори, деструктор і функцію виведення. Створити `public`-похідний клас – `школяр`, який має рік навчання. Визначити конструктори за умовчанням і з різним числом параметрів, деструктори, функцію виведення. Визначити функції перепризначення віку і класу.

Варіант 38.

Створити клас `вікно`, що має координати верхнього лівого і нижнього правого кута, колір фону (вказівник на рядок). Визначити конструктори, деструктор і функцію виведення. Створити `public`-похідний клас – `вікно, яке має рядок з меню`. Визначити конструктори за умовчанням і з різним числом параметрів, деструктор, функцію виведення. Визначити функції перепризначення кольору фону і рядка меню.

Варіант 39.

Створити ієрархію класів **людина** і **службовець**. Перевизначити виведення у потік і введення з потоку, конструктор копіювання, операцію присвоєння через відповідні функції базового класу.

Варіант 40.

Створити клас **точка**, що має координати. Визначити конструктори, деструктор і функцію виведення. Створити **public**-похідний клас – **кольорова точка**, що має колір. Визначити конструктори за умовчанням і з різним числом параметрів, деструктор, функцію виведення. Визначити функції перепризначення кольору і координат точки, виведення точки на екран.

Лабораторна робота № 3.3. Успадкування замість композиції

Мета роботи

Освоїти використання успадкування.

Питання, які необхідно вивчити та пояснити на захисті

- 1) Поняття та призначення успадкування.
- 2) Поняття базового / похідного класу, клас-предок / клас-нащадок.
- 3) Загальний синтаксис директиви успадкування.
- 4) Види успадкування.
- 5) Ключі успадкування та директиви доступу.
- 6) Просте успадкування.
- 7) Відкрите успадкування – успадкування інтерфейсу.
- 8) Закрите успадкування – успадкування реалізації.
- 9) Зв'язок між успадкуванням та композицією.
- 10) Обчислення кількості створених об'єктів.

Зразок виконання завдання

Подається лише умова завдання та текст програми.

Завдання Е

Умова завдання

Реалізувати завдання свого варіанту Лабораторної роботи № 1.5 «Композиція класів та об'єктів» з конструкторами та перевантаженням операцій як класи-нащадки від класів із завдання Лабораторної роботи № 1.5 (агрегований клас слід використовувати як базовий, а клас-контейнер стане похідним класом).

Оскільки слід реалізувати конструктори та операції, то зручніше за основу брати завдання свого варіанту Лабораторної роботи № 2.5 «Конструктори та перевантаження операцій для класів з композицією».

При цьому допоміжний клас слід реалізувати як відкритий клас-нащадок від базового класу Object. Клас Object реалізує лічильник кількості створених об'єктів.

Лабораторна робота № 2.5:

В кожній лабораторній роботі цієї теми потрібно реалізувати в тому або іншому вигляді визначення нового класу. У всіх завданнях необхідно реалізувати:

- конструктор ініціалізації (один або декілька),
- конструктор без аргументів і
- конструктор копіювання.

Вказані в завданні операції реалізуються за допомогою перевантаження підходящих операцій. У всіх завданнях обов'язково мають бути реалізовані відповідні операції:

- присвоєння,
- введення з клавіатури,
- виводу на екран,
- приведення типу – перетворення у літерний рядок.

Також треба реалізувати операції

- інкременту в обох формах (префіксній та постфіксній) і
- декременту в обох формах (префіксній та постфіксній), – для числових полів (наприклад: так, як вказано у варіантах завдань Лабораторної роботи № 2.3).

Перевантаження операцій виконується таким чином: підходящі операції реалізуються як методи класу, а інші – як зовнішні дружні функції.

Для демонстрації роботи з об'єктами нового типу у всіх завданнях потрібно написати головну функцію. У програмі обов'язково мають бути продемонстровані різні способи створення об'єктів і масивів об'єктів. Програма має демонструвати використання всіх функцій і методів. Вона має виводити на екран розмір класу в режимі `#pragma pack(1)` і без нього.

Визначення класів та реалізації методів слід розмістити в окремих модулях.

Лабораторна робота № 1.5:

У всіх завданнях потрібно реалізувати по два-три класи. Один клас є «контейнером», всі решту – описують поля, які містяться в «контейнері». Класи, що описують поля класу-«контейнера», мають бути визначені як незалежні.

Визначення класів та реалізації методів слід розмістити в окремих модулях.

Завдання наступне:

Створити клас **Man** (людина) з полями: ім'я, вік. Визначити методи пере-присвоєння імені, зміни віку.

Створити похідний клас **Student** (студент), що має поле «спеціальність». Визначити методи пере-присвоєння та зміни спеціальності.

Текст програми

```
////////////////////////////////////
// Source.cpp
//          головний файл проекту - функція main

#include "Student.h"

using namespace std;

int main()
{
    Man m1("Vasia", 18);
    cout << m1 << endl;
    cout << "count = " << Man::Count() << endl;
    {
        Student s4("Kuzia", 19, "KN");
        cout << s4 << endl;
        cout << "count = " << Student::Count() << endl;
    }
    cout << "count = " << Object::Count() << endl;

    return 0;
}

////////////////////////////////////
// Object.h
//          заголовний файл - визначення класу

#pragma once

class Object
{
    static unsigned int count;

public:
    static unsigned int Count()
    {
        return count;
    }

    Object()
    {
        count++;
    }

    ~Object()
    {
        count--;
    }
};

////////////////////////////////////
// Object.cpp
//          файл реалізації - реалізація методів класу

#include "Object.h"

unsigned int Object::count = 0;
```

```

////////////////////////////////////
// Man.h
// заголовний файл - визначення класу

#pragma once
#include <string>
#include <iostream>
#include "Object.h"

using namespace std;

class Man :public Object
{
private:
    string name;
    int age;

public:
    string getName() const { return name; }
    int getAge() const { return age; }

    void setName(string name) { this->name = name; }
    void setAge(int age) { this->age = age; }

    void Init(string name, int age);
    void Display() const;
    void Read();

    Man();
    Man(const string name);
    Man(const int age);
    Man(const string name, const int age);
    Man(const Man& m);

    Man& operator = (const Man& m);

    friend ostream& operator << (ostream& out, const Man& m);
    friend istream& operator >> (istream& in, Man& m);

    operator string () const;

    Man& operator ++ ();
    Man& operator -- ();
    Man operator ++ (int);
    Man operator -- (int);

    ~Man(void);
};

////////////////////////////////////
// Man.cpp
// файл реалізації - реалізація методів класу

#include "Man.h"
#include <sstream>

void Man::Init(string name, int age)
{
    setName(name);
    setAge(age);
}

```

```

void Man::Display() const
{
    cout << "name = " << name << endl;
    cout << "age = " << age << endl;
}

void Man::Read()
{
    string name;
    int age;
    cout << endl;
    cout << "name = ? "; cin >> name;
    cout << "age = ? "; cin >> age;
    Init(name, age);
}

Man::Man()
    : name(""), age(0)
{}

Man::Man(const string name)
    : name(name), age(0)
{}

Man::Man(const int age)
    : name(""), age(age)
{}

Man::Man(const string name, const int age)
    : name(name), age(age)
{}

Man::Man(const Man& m)
    : name(m.name), age(m.age)
{}

Man& Man::operator = (const Man& m)
{
    this->name = m.name;
    this->age = m.age;

    return *this;
}

ostream& operator << (ostream& out, const Man& m)
{
    out << string(m);
    return out;
}

istream& operator >> (istream& in, Man& m)
{
    string name;
    int age;
    cout << endl;
    cout << "name = ? "; in >> name;
    cout << "age = ? "; in >> age;
    m.setName(name);
    m.setAge(age);

    return in;
}

```

```

Man::operator string () const
{
    stringstream ss;
    ss << endl;
    ss << "name = " << name << endl;
    ss << "age = " << age << endl;

    return ss.str();
}

Man& Man::operator ++ ()
{
    ++age;
    return *this;
}

Man& Man::operator -- ()
{
    --age;
    return *this;
}

Man Man::operator ++ (int)
{
    Man t(*this);
    age++;
    return t;
}

Man Man::operator -- (int)
{
    Man t(*this);
    age--;
    return t;
}

Man::~Man(void)
{}

////////////////////////////////////
// Student.h
//          заголовний файл – визначення класу

#pragma once
#include "Man.h"

class Student :public Man
{
private:
    string spec;

public:
    string getSpec() const { return spec; }
    void setSpec(string spec) { this->spec = spec; }

    void Init(string name, int age, string spec);
    void Display() const;
    void Read();

    Student(const string name = "", const int age = 0, const string spec = "");
    Student(const Student& s);

```

```

Student& operator = (const Student& s);

friend ostream& operator << (ostream& out, const Student& s);
friend istream& operator >> (istream& in, Student& s);

operator string () const;

~Student(void);
};

////////////////////////////////////
// Student.cpp
//          файл реалізації – реалізація методів класу

#include "Man.h"
#include "Student.h"
#include <sstream>

void Student::Init(string name, int age, string spec)
{
    Man::Init(name, age);           // виклик базового метода
    setSpec(spec);
}

void Student::Display() const
{
    cout << endl;
    cout << "man = " << endl;
    Man::Display();                // виклик базового метода
    cout << "spec = " << spec << endl;
}

void Student::Read()
{
    string name;
    int age;
    cout << endl;
    cout << "name = ? "; cin >> name;
    cout << "age = ? "; cin >> age;

    string spec;
    cout << endl;
    cout << "spec = ? "; cin >> spec;
    Init(name, age, spec);
}

Student::Student(const string name, const int age, const string spec)
    : Man(name, age), spec(spec)    // виклик базового конструктора
{}

Student::Student(const Student& s)
    : Man(s)                        // виклик базового конструктора
{
    spec = s.spec;
}

Student& Student::operator = (const Student& s)
{
    Man::operator = (s);            // виклик базового присвоєння
    spec = s.spec;
    return *this;
}

```

```

ostream& operator << (ostream& out, const Student& s)
{
    out << string(s);
    return out;
}

istream& operator >> (istream& in, Student& s)
{
    string name;
    int age;
    cout << endl;
    cout << "name = ? "; in >> name;
    cout << "age = ? "; in >> age;
    s.setName(name);
    s.setAge(age);

    string spec;
    cout << endl;
    cout << "spec = ? "; in >> spec;
    s.setSpec(spec);

    return in;
}

Student::operator string () const
{
    stringstream ss;
    ss << "spec = " << spec << endl;

    return Man::operator string() + ss.str(); // виклик базового метода
}

Student::~Student(void)
{}

```


Варіанти завдань

Завдання А

Реалізувати класи із завдання свого варіанту Лабораторної роботи № 2.3 «Конструктори та перевантаження операцій для класів» з конструкторами, але без перевантаження операцій, – в якості базових класів. Реалізувати для базових класів операції вводу-виводу як дружні функції. Лабораторна робота № 2.3 виконується на основі Лабораторної роботи № 1.3 «Об’єкти – параметри методів (дії над кількома об’єктами)».

Реалізувати класи-нащадки, які розширюють контент базових класів перевантаженими операціями: базові класи містять конструктори та операції вводу-виводу, а похідні – всі інші операції. В нащадках методи базових класів можна використовувати для реалізації перевантажених операцій.

Класи-нащадки реалізувати в двох варіантах: як відкриті і як закриті класи-нащадки.

При відкритому успадкуванні не визначати функції вводу-виводу для класів-нащадків.

Лабораторна робота № 2.3:

В кожній лабораторній роботі цієї теми потрібно реалізувати в тому або іншому вигляді визначення нового класу. У всіх завданнях необхідно реалізувати:

- конструктор ініціалізації (один або декілька),
- конструктор без аргументів і
- конструктор копіювання.

Вказані в завданні операції реалізуються за допомогою перевантаження підходящих операцій. У всіх завданнях обов’язково мають бути реалізовані відповідні операції:

- присвоєння,
- введення з клавіатури,
- виводу на екран,
- приведення типу – перетворення у літерний рядок.

Також треба реалізувати операції

- інкременту в обох формах (префіксній та постфіксній) і
- декременту в обох формах (префіксній та постфіксній), – зміст цих операцій визначити самостійно.

Перевантаження операцій виконується таким чином: підходящі операції реалізуються як методи класу, а інші – як зовнішні дружні функції.

Для демонстрації роботи з об'єктами нового типу у всіх завданнях потрібно написати головну функцію. У програмі обов'язково мають бути продемонстровані різні способи створення об'єктів і масивів об'єктів. Програма має демонструвати використання всіх функцій і методів. Вона має виводити на екран розмір класу в режимі `#pragma pack(1)` і без нього.

Визначення класу та реалізацію його методів слід розмістити в окремих модулях.

Завдання наступне:

Виконати завдання свого варіанту Лабораторної роботи № 1.3. «Об'єкти – параметри методів (дії над кількома об'єктами)» як незалежні класи з конструкторами і перевантаженням операцій.

Лабораторна робота № 1.3:

У всіх завданнях, крім вказаних в завданні операцій, обов'язково мають бути реалізовані наступні методи:

- метод ініціалізації `Init()`;
- метод введення з клавіатури `Read()`;
- метод виведення на екран `Display()`;
- метод перетворення до літерного рядку `toString()`.

Всі завдання мають бути реалізовані як клас із закритими полями, де операції реалізуються як методи класу.

Визначення класу та реалізацію його методів слід розмістити в окремих модулях.

Для демонстрації роботи з об'єктами нового типу у всіх завданнях потрібно написати головну функцію. У програмі обов'язково мають бути продемонстровані різні способи створення об'єктів і масивів об'єктів – різними конструкторами. Програма має демонструвати використання всіх функцій і методів.

Варіанти завдань наступні:

Варіант 1.

Комплексне число представляється парою дійсних чисел (x, y) , де поля

- x – дійсна частина,
- y – мніма частина.

Реалізувати клас `Complex` для роботи з комплексними числами. Обов'язково мають бути реалізовані методи:

- додавання `add()` $(x_1, y_1) + (x_2, y_2) = (x_1 + x_2, y_1 + y_2)$;
- множення `mul()` $(x_1, y_1) \times (x_2, y_2) = (x_1 \cdot x_2 - y_1 \cdot y_2, x_1 \cdot y_2 + x_2 \cdot y_1)$;

- порівняння `equ()` $(x_1, y_1) = (x_2, y_2)$, якщо $(x_1 = x_2)$ і $(y_1 = y_2)$.

Варіант 2.

Створити клас `Vector3D`, що задається трійкою координат. Поля

- x
- y
- z

Обов'язково мають бути реалізовані:

- додавання векторів,
- віднімання векторів,
- скалярний добуток векторів.

Варіант 3.

Створити клас `Money` для роботи з грошовими сумами. Число має бути представлене двома полями:

- типу `long` для гривень і
- типу `byte` – для копійок.

Дробова частина (копійки) при виводі на екран має бути відокремлена від цілої частини комою. Реалізувати методи:

- додавання сум,
- ділення сум,
- ділення суми на дробове число.

Варіант 4.

Створити клас `Point` для роботи з точками на площині. Координати точки – декартові.

Поля:

- x
- y

Обов'язково мають бути реалізовані:

- переміщення точки по осі X ,
- переміщення по осі Y ,
- визначення відстані між двома точками.

Варіант 5.*

Раціональний (нескоротний) дріб представляється парою цілих чисел (a, b) , де поля:

- a – чисельник,
- b – знаменник.

Створити клас **Rational** для роботи з раціональними дробами. Обов'язково мають бути реалізовані наступні методи:

Унарна операція (аргументом є поточний об'єкт):

- обчислення значення `value()`, a / b ;
- ```
double Rational::value(){
 return 1.*a/b;
}

Rational z;
...
double x = z.value();
```

бінарні операції (перший аргумент – поточний об'єкт, другий аргумент – об'єкт-параметр):

- додавання `add()`,  $(a_1, b_1) + (a_2, b_2) = (a_1 \cdot b_2 + a_2 \cdot b_1, b_1 \cdot b_2)$ ;
- віднімання `sub()`,  $(a_1, b_1) - (a_2, b_2) = (a_1 \cdot b_2 - a_2 \cdot b_1, b_1 \cdot b_2)$ ;
- множення `mul()`,  $(a_1, b_1) \times (a_2, b_2) = (a_1 \cdot a_2, b_1 \cdot b_2)$ .

\* Має бути реалізована приватна функція скорочення дробу `Reduce()`, яка обов'язково викликається при виконанні арифметичних операцій.

## Пояснення Rational

Реалізація додавання за допомогою методу класу:

```
class Rational {
private:
 int a, b;
public:
 /* ... */
 Rational add(Rational& r);
};

Rational Rational::add(Rational& r) {
 Rational tmp;

 tmp.a = a * r.b + b * r.a;
 tmp.b = b * r.b;

 return tmp;
}
```

Використання додавання як методу класу:

```
Rational z1, z2, z3;
/* ... */
z3 = z1.add(z2);
```

Реалізація додавання за допомогою дружньої функції:

```

class Rational {
private:
 int a, b;
public:
 /* ... */
 friend Rational add(Rational& l, Rational& r);
};

Rational add(Rational& l, Rational& r) {
 Rational tmp;

 tmp.a = l.a * r.b + l.b * r.a;
 tmp.b = l.b * r.b;

 return tmp;
}

```

Використання додавання як дружньої функції:

```

Rational z1, z2, z3;
/* ... */
z3 = add(z1, z2);

```

## Варіант 6.

Реалізувати клас **FuzzyNumber** для роботи з нечіткими числами, які представляються трійками чисел  $(x - l, x, x + r)$ . Поля:

- $x$
- $l$
- $r$

Для чисел  $A = (x_A - l_A, x_A, x_A + r_A)$  та  $B = (x_B - l_B, x_B, x_B + r_B)$  арифметичні операції виконуються за наступними формулами:

- додавання

$$A + B = (x_A + x_B - l_A - l_B, x_A + x_B, x_A + x_B + r_A + r_B);$$

- множення

$$A \times B = (x_A \times x_B - x_B \times l_A - x_A \times l_B - l_A \times l_B, x_A \times x_B, x_A \times x_B + x_B \times r_A + x_A \times r_B + r_A \times r_B).$$

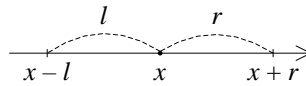
## Пояснення FuzzyNumber

Нечіткі числа подаються трійками  $(x - l, x, x + r)$ , де

- $x$  — координата центру,
- $x - l$  — координата лівої границі,
- $x + r$  — координата правої границі.

Відповідно:

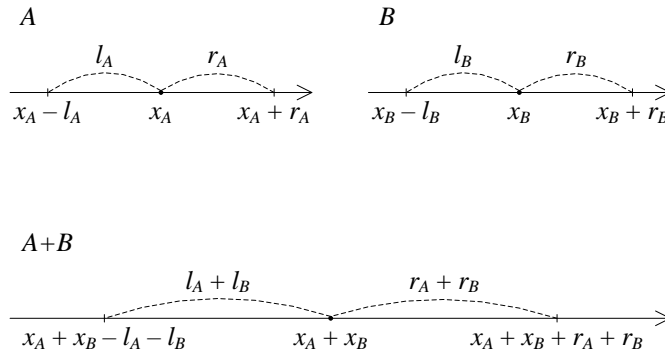
- $l$  — відстань від лівої границі до центру,
- $r$  — відстань від правої границі до центру:



Клас **FuzzyNumber** містить три поля:  $\{x, l, r\}$ .

Додавання двох нечітких чисел  $A = (x_A - l_A, x_A, x_A + r_A)$  та  $B = (x_B - l_B, x_B, x_B + r_B)$  описується формулою

$$A + B = (x_A + x_B - l_A - l_B, x_A + x_B, x_A + x_B + r_A + r_B)$$



Тобто, для суми двох нечітких чисел  $A+B$ :

- $x_A + x_B$  — координата центру,
- $x_A + x_B - l_A - l_B$  — координата лівої границі,
- $x_A + x_B + r_A + r_B$  — координата правої границі.

Відповідно,

- $l_A + l_B$  — відстань від лівої границі до центру,
- $r_A + r_B$  — відстань від правої границі до центру.

Таким чином, якщо об'єкт-число  $A$  містить поля  $\{x_A, l_A, r_A\}$ , а об'єкт-число  $B$  — поля  $\{x_B, l_B, r_B\}$ , то об'єкт-сума містить поля  $\{x_A + x_B, l_A + l_B, r_A + r_B\}$ .

## Варіант 7.

Номінали гривень можуть приймати значення 1, 2, 5, 10, 20, 50, 100, 200, 500. Копійки представити як 0.01 (1 копія), 0.02 (2 копійки), 0.05 (5 копійок), 0.1 (10 копійок), 0.25 (25 копійок), 0.5 (50 копійок).

Створити клас **Money** для роботи з грошовими сумами. Сума має бути представлена полями-номіналами, значеннями яких має бути кількість купюр відповідного номіналу. Поля:

- кількість банкнот по 500 грн.
- кількість банкнот по 200 грн.
- кількість банкнот по 100 грн.
- кількість банкнот по 50 грн.

- кількість банкнот по 20 грн.
- кількість банкнот по 10 грн.
- кількість банкнот по 5 грн.
- кількість банкнот по 2 грн.
- кількість банкнот по 1 грн.
- кількість монет по 50 коп.
- кількість монет по 25 коп.
- кількість монет по 10 коп.
- кількість монет по 5 коп.
- кількість монет по 2 коп.
- кількість монет по 1 коп.

Реалізувати:

- додавання сум,
- віднімання сум,
- множення суми на дробове число.

Дробова частина (копійки) при виводі на екран має бути відокремлена від цілої частини комою.

### Варіант 8.

Створити клас `Fraction` для роботи з дробовими числами. Число має бути представлене двома полями:

- ціла частина – довге ціле із знаком,
- дробова частина – без-знакове коротке ціле.

Реалізувати методи – арифметичні операції:

- додавання,
- множення.

### Варіант 9.

Створити клас `BitString` для роботи з 64-бітовими рядками. Бітовий рядок має бути представлений двома полями типу `long`. Мають бути визначені методи, які реалізують всі традиційні операції для роботи з бітами:

- `not`,
- `and`,
- `or`.

### Варіант 10.\*

Створити клас **LongLong** для роботи з 64-розрядними цілими числами. Число має бути представлене двома полями:

- типу **long** – старша частина,
- типу **long** – молодша частина.

Мають бути реалізовані методи, які представляють:

- арифметичні операції, присутні в мові програмування (без присвоєння):
  - \* додавання,
  - \* множення;
- операції порівняння:
  - менше, не менше, більше.

### Варіант 11.

Створити клас **Vector2D**, що задається парою координат. Поля

- $x$
- $y$

Обов'язково мають бути реалізовані:

- скалярний добуток векторів,
- множення на скаляр,
- обчислення довжини вектора,
- порівняння довжин векторів.

### Варіант 12.

Комплексне число представляється парою дійсних чисел  $(x, y)$ , де поля

- $x$  – дійсна частина,
- $y$  – мніма частина.

Реалізувати клас **Complex** для роботи з комплексними числами. Обов'язково мають бути реалізовані методи:

- віднімання **sub()**  $(x_1, y_1) - (x_2, y_2) = (x_1 - x_2, y_1 - y_2);$
- ділення **div()**  $(x_1, y_1) / (x_2, y_2) = (x_1 \cdot x_2 + y_1 \cdot y_2, x_2 \cdot y_1 - x_1 \cdot y_2) / (x_2^2 + y_2^2);$
- комплексно спряжене число **conj()**  $\text{conj}(x, y) = (x, -y).$

### Варіант 13.

Створити клас **Vector3D**, що задається трійкою координат. Поля



- $x$
- $y$
- $z$

Обов'язково мають бути реалізовані:

- множення на скаляр,
- порівняння векторів,
- обчислення довжини вектора,
- порівняння довжин векторів.

#### Варіант 14.

Створити клас **Money** для роботи з грошовими сумами. Число має бути представлене двома полями:

- типу **long** для гривень і
- типу **byte** – для копійок.

Дробова частина (копійки) при виводі на екран має бути відокремлена від цілої частини комою. Реалізувати методи:

- віднімання сум,
- множення на дробове число,
- операції порівняння сум.

#### Варіант 15.

Створити клас **Point** для роботи з точками на площині. Координати точки – декартові.

Поля:

- $x$
- $y$

Обов'язково мають бути реалізовані:

- перетворення у полярні координати,
- визначення відстані до початку координат,
- порівняння на рівність та нерівність.

#### Варіант 16.\*

Раціональний (нескоротний) дріб представляється парою цілих чисел  $(a, b)$ , де поля:

- $a$  – чисельник,
- $b$  – знаменник.

Створити клас **Rational** для роботи з раціональними дробами. Обов'язково мають

бути реалізовані наступні методи:

Унарна операція (аргументом є поточний об'єкт):

- обчислення значення `value()`,  $a / b$ ;

```
double Rational::value(){
 return 1.*a/b;
}

Rational z;
...
double x = z.value();
```

бінарні операції (перший аргумент – поточний об'єкт, другий аргумент – об'єкт-параметр):

- ділення `div()`,  $(a_1, b_1) / (a_2, b_2) = (a_1 \cdot b_2, a_2 \cdot b_1)$ ;
- порівняння «чи рівне» `equal()`;
- порівняння «чи більше» `great()`;
- порівняння «чи менше» `less()`.

\* Має бути реалізована приватна функція скорочення дробу `Reduce()`, яка обов'язково викликається при виконанні арифметичних операцій.

## Пояснення Rational

Реалізація додавання за допомогою методу класу:

```
class Rational {
private:
 int a, b;
public:
 /* ... */
 Rational add(Rational& r);
};

Rational Rational::add(Rational& r) {
 Rational tmp;

 tmp.a = a * r.b + b * r.a;
 tmp.b = b * r.b;

 return tmp;
}
```

Використання додавання як методу класу:

```
Rational z1, z2, z3;
/* ... */
z3 = z1.add(z2);
```

Реалізація додавання за допомогою дружньої функції:

```
class Rational {
private:
 int a, b;
```

```

public:
 /* ... */
 friend Rational add(Rational& l, Rational& r);
};

Rational add(Rational& l, Rational& r) {
 Rational tmp;

 tmp.a = l.a * r.b + l.b * r.a;
 tmp.b = l.b * r.b;

 return tmp;
}

```

Використання додавання як дружньої функції:

```

Rational z1, z2, z3;
/* ... */
z3 = add(z1, z2);

```

## Варіант 17.

Реалізувати клас **FuzzyNumber** для роботи з нечіткими числами, які представляються трійками чисел  $(x - l, x, x + r)$ . Поля:

- $x$
- $l$
- $r$

Для чисел  $A = (x_A - l_A, x_A, x_A + r_A)$  та  $B = (x_B - l_B, x_B, x_B + r_B)$  арифметичні операції виконуються за наступними формулами:

- віднімання  

$$A - B = (x_A - x_B - l_A - l_B, x_A - x_B, x_A - x_B + r_A + r_B);$$
- зворотне число  

$$1 / A = (1/(x_A + r_A), 1 / x_A, 1/(x_A - l_A)), x_A > 0;$$
- ділення  

$$A / B = ((x_A - l_A)/(x_B + r_B), x_A / x_B, (x_A + r_A)/(x_B - l_B)), x_B > 0.$$

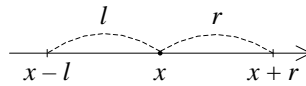
## Пояснення FuzzyNumber

Нечіткі числа подаються трійками  $(x - l, x, x + r)$ , де

- $x$  — координата центру,
- $x - l$  — координата лівої границі,
- $x + r$  — координата правої границі.

Відповідно:

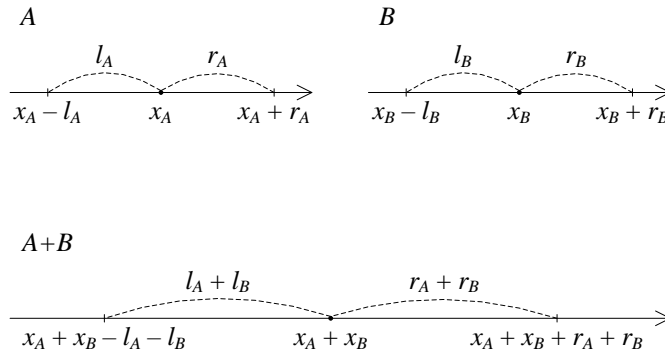
- $l$  — відстань від лівої границі до центру,
- $r$  — відстань від правої границі до центру:



Клас **FuzzyNumber** містить три поля:  $\{x, l, r\}$ .

Додавання двох нечітких чисел  $A = (x_A - l_A, x_A, x_A + r_A)$  та  $B = (x_B - l_B, x_B, x_B + r_B)$  описується формулою

$$A + B = (x_A + x_B - l_A - l_B, x_A + x_B, x_A + x_B + r_A + r_B)$$



Тобто, для суми двох нечітких чисел  $A+B$ :

- $x_A + x_B$  — координата центру,
- $x_A + x_B - l_A - l_B$  — координата лівої границі,
- $x_A + x_B + r_A + r_B$  — координата правої границі.

Відповідно,

- $l_A + l_B$  — відстань від лівої границі до центру,
- $r_A + r_B$  — відстань від правої границі до центру.

Таким чином, якщо об'єкт-число  $A$  містить поля  $\{x_A, l_A, r_A\}$ , а об'єкт-число  $B$  — поля  $\{x_B, l_B, r_B\}$ , то об'єкт-сума містить поля  $\{x_A + x_B, l_A + l_B, r_A + r_B\}$ .

## Варіант 18.

Номінали гривень можуть приймати значення 1, 2, 5, 10, 20, 50, 100, 200, 500. Копійки представити як 0.01 (1 копійка), 0.02 (2 копійки), 0.05 (5 копійок), 0.1 (10 копійок), 0.25 (25 копійок), 0.5 (50 копійок).

Створити клас **Money** для роботи з грошовими сумами. Сума має бути представлена полями-номіналами, значеннями яких має бути кількість купюр відповідного номіналу. Поля:

- кількість банкнот по 500 грн.
- кількість банкнот по 200 грн.
- кількість банкнот по 100 грн.
- кількість банкнот по 50 грн.

- кількість банкнот по 20 грн.
- кількість банкнот по 10 грн.
- кількість банкнот по 5 грн.
- кількість банкнот по 2 грн.
- кількість банкнот по 1 грн.
- кількість монет по 50 коп.
- кількість монет по 25 коп.
- кількість монет по 10 коп.
- кількість монет по 5 коп.
- кількість монет по 2 коп.
- кількість монет по 1 коп.

Реалізувати:

- ділення сум,
- ділення суми на дробове число,
- операції порівняння сум.

Дробова частина (копійки) при виводі на екран має бути відокремлена від цілої частини комою.

### Варіант 19.

Створити клас `Fraction` для роботи з дробовими числами. Число має бути представлене двома полями:

- ціла частина – довге ціле із знаком,
- дробова частина – без-знакове коротке ціле.

Реалізувати методи – арифметичні операції:

- віднімання,
- операції порівняння.

### Варіант 20.\*

Створити клас `BitString` для роботи з 64-бітовими рядками. Бітовий рядок має бути представлений двома полями типу `long`. Мають бути визначені методи, які реалізують всі традиційні операції для роботи з бітами:

- `xor`,
- \* зсув ліворуч `shiftLeft` на задану кількість бітів,
- \* зсув праворуч `shiftRight` на задану кількість бітів.

### Варіант 21.\*

Створити клас **LongLong** для роботи з 64-розрядними цілими числами. Число має бути представлене двома полями:

- типу **long** – старша частина,
- типу **long** – молодша частина.

Мають бути реалізовані методи, які представляють:

- арифметичні операції (без присвоєння):
  - \* віднімання,
  - \* ділення;
- операції порівняння:
  - не більше, дорівнює, не дорівнює.

### Варіант 22.

Створити клас **VectorN**, що задається групою  $N$  дійсних чисел – координат вектора. Поля

- $N$  – розмірність вектора,
- $a$  – масив дійсних чисел, який реалізує вектор.

Обов'язково мають бути реалізовані:

- додавання векторів,
- віднімання векторів,
- скалярний добуток векторів.

### Варіант 23.

Створити клас **VectorN**, що задається групою  $N$  дійсних чисел – координат вектора. Поля

- $N$  – розмірність вектора,
- $a$  – масив дійсних чисел, який реалізує вектор.

Обов'язково мають бути реалізовані:

- множення на скаляр,
- порівняння векторів,
- обчислення довжини вектора,
- порівняння довжин векторів.

### Варіант 24.

Створити клас **Matrix** – реалізує матрицю цілих елементів, який містить закриті поля:

- $m$  – двовимірний масив,

- $R$  – кількість рядків,
- $C$  – кількість стовпців.

Визначити методи для:

- повернення значення елемента, який має індекси  $(i, j)$ ;
- виведення матриці;
- додавання матриць;
- віднімання матриць;
- множення матриць;
- множення матриці на число.

### Варіант 25.

Розробити клас `CharLine` – реалізує рядок  $N$  символів. У закритій частині визначити поля:

- $N$  – довжина рядка (кількість символів);
- $s$  – масив, який вміщує  $N$  символів.

Визначити методи:

- введення-виведення рядка,
- виведення символу у вказаній позиції,
- перевірки входження заданого символу у рядок.
- конкатенації,
- порівняння рядків,
- перевірки входження під-рядка у рядок.

### Варіант 26.

Комплексне число представляється парою дійсних чисел  $(x, y)$ , де поля

- $x$  – дійсна частина,
- $y$  – мніма частина.

Реалізувати клас `Complex` для роботи з комплексними числами. Обов'язково мають бути реалізовані методи:

- додавання `add()`  $(x_1, y_1) + (x_2, y_2) = (x_1 + x_2, y_1 + y_2)$ ;
- множення `mul()`  $(x_1, y_1) \times (x_2, y_2) = (x_1 \cdot x_2 - y_1 \cdot y_2, x_1 \cdot y_2 + x_2 \cdot y_1)$ ;
- порівняння `equ()`  $(x_1, y_1) = (x_2, y_2)$ , якщо  $(x_1 = x_2)$  і  $(y_1 = y_2)$ .

### Варіант 27.

Створити клас `Vector3D`, що задається трійкою координат. Поля

- $x$
- $y$
- $z$

Обов'язково мають бути реалізовані:

- додавання векторів,
- віднімання векторів,
- скалярний добуток векторів.

### Варіант 28.

Створити клас `Money` для роботи з грошовими сумами. Число має бути представлене двома полями:

- типу `long` для гривень і
- типу `byte` – для копійок.

Дробова частина (копійки) при виводі на екран має бути відокремлена від цілої частини комою. Реалізувати методи:

- додавання сум,
- ділення сум,
- ділення суми на дробове число.

### Варіант 29.

Створити клас `Point` для роботи з точками на площині. Координати точки – декартові.

Поля:

- $x$
- $y$

Обов'язково мають бути реалізовані:

- переміщення точки по осі  $X$ ,
- переміщення по осі  $Y$ ,
- визначення відстані між двома точками.

### Варіант 30.\*

Раціональний (нескоротний) дріб представляється парою цілих чисел  $(a, b)$ , де поля:

- $a$  – чисельник,
- $b$  – знаменник.

Створити клас `Rational` для роботи з раціональними дробами. Обов'язково мають бути реалізовані наступні методи:



Унарна операція (аргументом є поточний об'єкт):

- обчислення значення `value()`,  $a / b$ ;  

```
double Rational::value(){
 return 1.*a/b;
}

Rational z;
...
double x = z.value();
```

Бінарні операції (перший аргумент – поточний об'єкт, другий аргумент – об'єкт-параметр):

- додавання `add()`,  $(a_1, b_1) + (a_2, b_2) = (a_1 \cdot b_2 + a_2 \cdot b_1, b_1 \cdot b_2)$ ;
- віднімання `sub()`,  $(a_1, b_1) - (a_2, b_2) = (a_1 \cdot b_2 - a_2 \cdot b_1, b_1 \cdot b_2)$ ;
- множення `mul()`,  $(a_1, b_1) \times (a_2, b_2) = (a_1 \cdot a_2, b_1 \cdot b_2)$ .

\* Має бути реалізована приватна функція скорочення дробу `Reduce()`, яка обов'язково викликається при виконанні арифметичних операцій.

## Пояснення Rational

Реалізація додавання за допомогою методу класу:

```
class Rational {
private:
 int a, b;
public:
 /* ... */
 Rational add(Rational& r);
};

Rational Rational::add(Rational& r) {
 Rational tmp;

 tmp.a = a * r.b + b * r.a;
 tmp.b = b * r.b;

 return tmp;
}
```

Використання додавання як методу класу:

```
Rational z1, z2, z3;
/* ... */
z3 = z1.add(z2);
```

Реалізація додавання за допомогою дружньої функції:

```
class Rational {
private:
 int a, b;
public:
 /* ... */
 friend Rational add(Rational& l, Rational& r);
};
```

```

Rational add(Rational& l, Rational& r) {
 Rational tmp;

 tmp.a = l.a * r.b + l.b * r.a;
 tmp.b = l.b * r.b;

 return tmp;
}

```

Використання додавання як дружньої функції:

```

Rational z1, z2, z3;
/* ... */
z3 = add(z1, z2);

```

### Варіант 31.

Реалізувати клас **FuzzyNumber** для роботи з нечіткими числами, які представляються трійками чисел  $(x - l, x, x + r)$ . Поля:

- $x$
- $l$
- $r$

Для чисел  $A = (x_A - l_A, x_A, x_A + r_A)$  та  $B = (x_B - l_B, x_B, x_B + r_B)$  арифметичні операції виконуються за наступними формулами:

- додавання  

$$A + B = (x_A + x_B - l_A - l_B, x_A + x_B, x_A + x_B + r_A + r_B);$$
- множення  

$$A \times B = (x_A \times x_B - x_B \times l_A - x_A \times l_B - l_A \times l_B, x_A \times x_B, x_A \times x_B + x_B \times r_A + x_A \times r_B + r_A \times r_B).$$

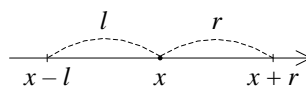
### Пояснення FuzzyNumber

Нечіткі числа подаються трійками  $(x - l, x, x + r)$ , де

- $x$  – координата центру,
- $x - l$  – координата лівої границі,
- $x + r$  – координата правої границі.

Відповідно:

- $l$  – відстань від лівої границі до центру,
- $r$  – відстань від правої границі до центру:

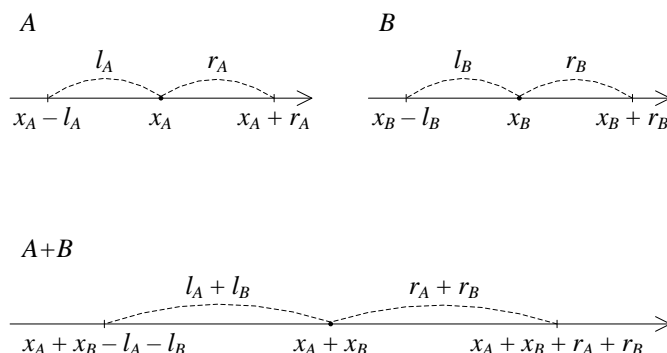


Клас **FuzzyNumber** містить три поля:  $\{x, l, r\}$ .

Додавання двох нечітких чисел  $A = (x_A - l_A, x_A, x_A + r_A)$  та  $B = (x_B - l_B, x_B, x_B + r_B)$

описується формулою

$$A + B = (x_A + x_B - l_A - l_B, x_A + x_B, x_A + x_B + r_A + r_B)$$



Тобто, для суми двох нечітких чисел  $A+B$ :

$x_A + x_B$  — координата центру,

$x_A + x_B - l_A - l_B$  — координата лівої границі,

$x_A + x_B + r_A + r_B$  — координата правої границі.

Відповідно,

$l_A + l_B$  — відстань від лівої границі до центру,

$r_A + r_B$  — відстань від правої границі до центру.

Таким чином, якщо об'єкт-число  $A$  містить поля  $\{x_A, l_A, r_A\}$ , а об'єкт-число  $B$  — поля  $\{x_B, l_B, r_B\}$ , то об'єкт-сума містить поля  $\{x_A + x_B, l_A + l_B, r_A + r_B\}$ .

## Варіант 32.

Номінали гривень можуть приймати значення 1, 2, 5, 10, 20, 50, 100, 200, 500. Копійки представити як 0.01 (1 копійка), 0.02 (2 копійки), 0.05 (5 копійок), 0.1 (10 копійок), 0.25 (25 копійок), 0.5 (50 копійок).

Створити клас **Money** для роботи з грошовими сумами. Сума має бути представлена полями-номіналами, значеннями яких має бути кількість купюр відповідного номіналу. Поля:

- кількість банкнот по 500 грн.
- кількість банкнот по 200 грн.
- кількість банкнот по 100 грн.
- кількість банкнот по 50 грн.
- кількість банкнот по 20 грн.
- кількість банкнот по 10 грн.
- кількість банкнот по 5 грн.

- кількість банкнот по 2 грн.
- кількість банкнот по 1 грн.
- кількість монет по 50 коп.
- кількість монет по 25 коп.
- кількість монет по 10 коп.
- кількість монет по 5 коп.
- кількість монет по 2 коп.
- кількість монет по 1 коп.

Реалізувати:

- додавання сум,
- віднімання сум,
- множення суми на дробове число.

Дробова частина (копійки) при виводі на екран має бути відокремлена від цілої частини комою.

### **Варіант 33.**

Створити клас `Fraction` для роботи з дробовими числами. Число має бути представлене двома полями:

- ціла частина – довге ціле із знаком,
- дробова частина – без-знакове коротке ціле.

Реалізувати методи – арифметичні операції:

- додавання,
- множення.

### **Варіант 34.**

Створити клас `BitString` для роботи з 64-бітовими рядками. Бітовий рядок має бути представлений двома полями типу `long`. Мають бути визначені методи, які реалізують всі традиційні операції для роботи з бітами:

- `not`,
- `and`,
- `or`.

### **Варіант 35.\***

Створити клас `LongLong` для роботи з 64-розрядними цілими числами. Число має бути представлене двома полями:

- типу `long` – старша частина,
- типу `long` – молодша частина.

Мають бути реалізовані методи, які представляють:

- арифметичні операції, присутні в мові програмування (без присвоєння):
  - \* додавання,
  - \* множення;
- операції порівняння:
  - менше, не менше, більше.

### Варіант 36.

Створити клас `Vector2D`, що задається парою координат. Поля

- $x$
- $y$

Обов'язково мають бути реалізовані:

- скалярний добуток векторів,
- множення на скаляр,
- обчислення довжини вектора,
- порівняння довжин векторів.

### Варіант 37.

Комплексне число представляються парою дійсних чисел  $(x, y)$ , де поля

- $x$  – дійсна частина,
- $y$  – мніма частина.

Реалізувати клас `Complex` для роботи з комплексними числами. Обов'язково мають бути реалізовані методи:

- віднімання `sub()`  $(x_1, y_1) - (x_2, y_2) = (x_1 - x_2, y_1 - y_2)$ ;
- ділення `div()`  $(x_1, y_1) / (x_2, y_2) = (x_1 \cdot x_2 + y_1 \cdot y_2, x_2 \cdot y_1 - x_1 \cdot y_2) / (x_2^2 + y_2^2)$ ;
- комплексно спряжене число `conj()`  $\text{conj}(x, y) = (x, -y)$ .

### Варіант 38.

Створити клас `Vector3D`, що задається трійкою координат. Поля

- $x$
- $y$
- $z$

Обов'язково мають бути реалізовані:

- множення на скаляр,
- порівняння векторів,
- обчислення довжини вектора,
- порівняння довжин векторів.

### Варіант 39.

Створити клас `Money` для роботи з грошовими сумами. Число має бути представлене двома полями:

- типу `long` для гривень і
- типу `byte` – для копійок.

Дробова частина (копійки) при виводі на екран має бути відокремлена від цілої частини комою. Реалізувати методи:

- віднімання сум,
- множення на дробове число,
- операції порівняння сум.

### Варіант 40.

Створити клас `Point` для роботи з точками на площині. Координати точки – декартові.

Поля:

- $x$
- $y$

Обов'язково мають бути реалізовані:

- перетворення у полярні координати,
- визначення відстані до початку координат,
- порівняння на рівність та нерівність.

## Завдання В

Реалізувати класи із завдання свого варіанту Лабораторної роботи № 1.3 «Об'єкти – параметри методів (дії над кількома об'єктами)» як класи-нащадки; використовувати відкрите успадкування (оскільки слід реалізувати конструктори та операції, то зручніше за основу брати завдання свого варіанту Лабораторної роботи № 2.3 «Конструктори та перевантаження операцій для класів»).

В якості базового класу реалізувати відповідний клас: пару чисел (**Pair**) або трійку чисел (**Triad**) з конструкторами та повним набором операцій порівняння.

Клас **Pair** (пара чисел): пара **p1** більше пари **p2**, якщо

`(p1.first > p2.first)`

або

`(p1.first = p2.first) і (p1.second > p2.second).`

Клас **Triad** (трійка чисел): тріада **t1** більше тріади **t2**, якщо

`(t1.first > t2.first)`

або

`(t1.first = t2.first) і (t1.second > t2.second)`

або

`(t1.first = t2.first) і (t1.second = t2.second) і (t1.third > t2.third).`

Реалізувати для базових класів операції вводу-виводу як дружні функції.

Не визначати функції вводу-виводу для класів-нащадків.

### Лабораторна робота № 1.3:

*Класом-парою* називається клас з двома приватними полями, які мають імена **first** та **second**. Потрібно реалізувати такий клас. Обов'язково мають бути присутніми:

- методи доступу (константні методи зчитування та методи запису) значення кожного поля;
- метод ініціалізації `Init()`; метод має контролювати значення аргументів на коректність;
- метод введення з клавіатури `Read()`;
- метод виведення на екран `Display()`.

Реалізувати зовнішню функцію з ім'ям `makeКлас()`, де *Клас* – ім'я класу, об'єкт

якого вона створює. Функція має отримувати як аргументи значення для полів класу і повертати об'єкт необхідного класу. При передачі помилкових параметрів слід виводити повідомлення і закінчувати роботу.

Визначення класу та реалізацію його методів слід розмістити в окремих модулях.

Варіанти завдань наступні:

### Варіант 1.

Комплексне число представляється парою дійсних чисел  $(x, y)$ , де поля

- $x$  – дійсна частина,
- $y$  – мніма частина.

Реалізувати клас **Complex** для роботи з комплексними числами. Обов'язково мають бути реалізовані методи:

- додавання **add()**  $(x_1, y_1) + (x_2, y_2) = (x_1 + x_2, y_1 + y_2)$ ;
- множення **mul()**  $(x_1, y_1) \times (x_2, y_2) = (x_1 \cdot x_2 - y_1 \cdot y_2, x_1 \cdot y_2 + x_2 \cdot y_1)$ ;
- порівняння **equ()**  $(x_1, y_1) = (x_2, y_2)$ , якщо  $(x_1 = x_2)$  і  $(y_1 = y_2)$ .

### Варіант 2.

Створити клас **Vector3D**, що задається трійкою координат. Поля

- $x$
- $y$
- $z$

Обов'язково мають бути реалізовані:

- додавання векторів,
- віднімання векторів,
- скалярний добуток векторів.

### Варіант 3.

Створити клас **Money** для роботи з грошовими сумами. Число має бути представлене двома полями:

- типу **long** для гривень і
- типу **byte** – для копійок.

Дробова частина (копійки) при виводі на екран має бути відокремлена від цілої частини комою. Реалізувати методи:

- додавання сум,
- ділення сум,



- ділення суми на дробове число.

#### Варіант 4.

Створити клас **Point** для роботи з точками на площині. Координати точки – декартові.

Поля:

- $x$
- $y$

Обов'язково мають бути реалізовані:

- переміщення точки по осі  $X$ ,
- переміщення по осі  $Y$ ,
- визначення відстані між двома точками.

#### Варіант 5.\*

Раціональний (нескоротний) дріб представляється парою цілих чисел  $(a, b)$ , де поля:

- $a$  – чисельник,
- $b$  – знаменник.

Створити клас **Rational** для роботи з раціональними дробами. Обов'язково мають бути реалізовані наступні методи:

Унарна операція (аргументом є поточний об'єкт):

- обчислення значення `value()`,  $a / b$ ;  

```
double Rational::value(){
 return 1.*a/b;
}

Rational z;
...
double x = z.value();
```

бінарні операції (перший аргумент – поточний об'єкт, другий аргумент – об'єкт-параметр):

- додавання `add()`,  $(a_1, b_1) + (a_2, b_2) = (a_1 \cdot b_2 + a_2 \cdot b_1, b_1 \cdot b_2)$ ;
- віднімання `sub()`,  $(a_1, b_1) - (a_2, b_2) = (a_1 \cdot b_2 - a_2 \cdot b_1, b_1 \cdot b_2)$ ;
- множення `mul()`,  $(a_1, b_1) \times (a_2, b_2) = (a_1 \cdot a_2, b_1 \cdot b_2)$ .

\* Має бути реалізована приватна функція скорочення дробу `Reduce()`, яка обов'язково викликається при виконанні арифметичних операцій.

#### Пояснення Rational

Реалізація додавання за допомогою методу класу:

```

class Rational {
private:
 int a, b;
public:
 /* ... */
 Rational add(Rational& r);
};

Rational Rational::add(Rational& r) {
 Rational tmp;

 tmp.a = a * r.b + b * r.a;
 tmp.b = b * r.b;

 return tmp;
}

```

Використання додавання як методу класу:

```

Rational z1, z2, z3;
/* ... */
z3 = z1.add(z2);

```

Реалізація додавання за допомогою дружньої функції:

```

class Rational {
private:
 int a, b;
public:
 /* ... */
 friend Rational add(Rational& l, Rational& r);
};

Rational add(Rational& l, Rational& r) {
 Rational tmp;

 tmp.a = l.a * r.b + l.b * r.a;
 tmp.b = l.b * r.b;

 return tmp;
}

```

Використання додавання як дружньої функції:

```

Rational z1, z2, z3;
/* ... */
z3 = add(z1, z2);

```

## Варіант 6.

Реалізувати клас **FuzzyNumber** для роботи з нечіткими числами, які представляються трійками чисел  $(x - l, x, x + r)$ . Поля:

- $x$
- $l$
- $r$

Для чисел  $A = (x_A - l_A, x_A, x_A + r_A)$  та  $B = (x_B - l_B, x_B, x_B + r_B)$  арифметичні операції виконуються за наступними формулами:

- додавання

$$A + B = (x_A + x_B - l_A - l_B, x_A + x_B, x_A + x_B + r_A + r_B);$$

- множення

$$A \times B = (x_A \times x_B - x_B \times l_A - x_A \times l_B - l_A \times l_B, x_A \times x_B, x_A \times x_B + x_B \times r_A + x_A \times r_B + r_A \times r_B).$$

## Пояснення FuzzyNumber

Нечіткі числа подаються трійками  $(x - l, x, x + r)$ , де

$x$  – координата центру,

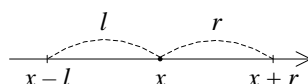
$x - l$  – координата лівої границі,

$x + r$  – координата правої границі.

Відповідно:

$l$  – відстань від лівої границі до центру,

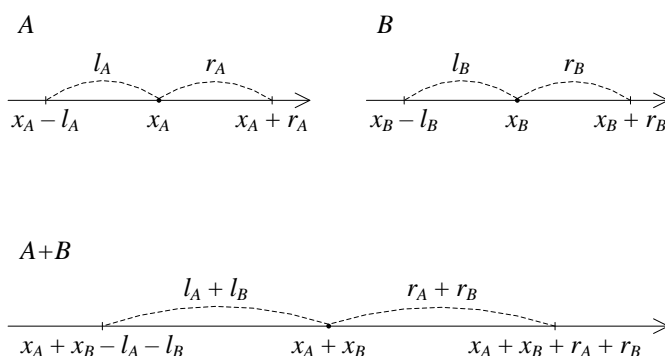
$r$  – відстань від правої границі до центру:



Клас FuzzyNumber містить три поля:  $\{x, l, r\}$ .

Додавання двох нечітких чисел  $A = (x_A - l_A, x_A, x_A + r_A)$  та  $B = (x_B - l_B, x_B, x_B + r_B)$  описується формулою

$$A + B = (x_A + x_B - l_A - l_B, x_A + x_B, x_A + x_B + r_A + r_B)$$



Тобто, для суми двох нечітких чисел  $A+B$ :

$x_A + x_B$  – координата центру,

$x_A + x_B - l_A - l_B$  – координата лівої границі,

$x_A + x_B + r_A + r_B$  – координата правої границі.

Відповідно,

$l_A + l_B$  – відстань від лівої границі до центру,

$r_A + r_B$  – відстань від правої границі до центру.

Таким чином, якщо об'єкт-число  $A$  містить поля  $\{x_A, l_A, r_A\}$ , а об'єкт-число  $B$  – поля  $\{x_B, l_B, r_B\}$ , то об'єкт-сума містить поля  $\{x_A + x_B, l_A + l_B, r_A + r_B\}$ .

### Варіант 7.

Номінали гривень можуть приймати значення 1, 2, 5, 10, 20, 50, 100, 200, 500. Копійки представити як 0.01 (1 копійка), 0.02 (2 копійки), 0.05 (5 копійок), 0.1 (10 копійок), 0.25 (25 копійок), 0.5 (50 копійок).

Створити клас `Money` для роботи з грошовими сумами. Сума має бути представлена полями-номіналами, значеннями яких має бути кількість купюр відповідного номіналу.

Поля:

- кількість банкнот по 500 грн.
- кількість банкнот по 200 грн.
- кількість банкнот по 100 грн.
- кількість банкнот по 50 грн.
- кількість банкнот по 20 грн.
- кількість банкнот по 10 грн.
- кількість банкнот по 5 грн.
- кількість банкнот по 2 грн.
- кількість банкнот по 1 грн.
- кількість монет по 50 коп.
- кількість монет по 25 коп.
- кількість монет по 10 коп.
- кількість монет по 5 коп.
- кількість монет по 2 коп.
- кількість монет по 1 коп.

Реалізувати:

- додавання сум,
- віднімання сум,
- множення суми на дробове число.

Дробова частина (копійки) при виводі на екран має бути відокремлена від цілої частини комою.

### Варіант 8.

Створити клас `Fraction` для роботи з дробовими числами. Число має бути

представлене двома полями:

- ціла частина – довге ціле із знаком,
- дробова частина – без-знакове коротке ціле.

Реалізувати методи – арифметичні операції:

- додавання,
- множення.

### Варіант 9.

Створити клас `BitString` для роботи з 64-бітовими рядками. Бітовий рядок має бути представлений двома полями типу `long`. Мають бути визначені методи, які реалізують всі традиційні операції для роботи з бітами:

- `not`,
- `and`,
- `or`.

### Варіант 10.\*

Створити клас `LongLong` для роботи з 64-розрядними цілими числами. Число має бути представлене двома полями:

- типу `long` – старша частина,
- типу `long` – молодша частина.

Мають бути реалізовані методи, які представляють:

- арифметичні операції, присутні в мові програмування (без присвоєння):
  - `*` додавання,
  - `*` множення;
- операції порівняння:
  - `<` менше, `<=` не менше, `>` більше.

### Варіант 11.

Створити клас `Vector2D`, що задається парою координат. Поля

- `x`
- `y`

Обов'язково мають бути реалізовані:

- скалярний добуток векторів,
- множення на скаляр,
- обчислення довжини вектора,

- порівняння довжин векторів.

## Варіант 12.

Комплексне число представляється парою дійсних чисел  $(x, y)$ , де поля

- $x$  – дійсна частина,
- $y$  – мніма частина.

Реалізувати клас **Complex** для роботи з комплексними числами. Обов'язково мають бути реалізовані методи:

- віднімання **sub()**  $(x_1, y_1) - (x_2, y_2) = (x_1 - x_2, y_1 - y_2)$ ;
- ділення **div()**  $(x_1, y_1) / (x_2, y_2) = (x_1 \cdot x_2 + y_1 \cdot y_2, x_2 \cdot y_1 - x_1 \cdot y_2) / (x_2^2 + y_2^2)$ ;
- комплексно спряжене число **conj()**  $\text{conj}(x, y) = (x, -y)$ .

## Варіант 13.

Створити клас **Vector3D**, що задається трійкою координат. Поля

- $x$
- $y$
- $z$

Обов'язково мають бути реалізовані:

- множення на скаляр,
- порівняння векторів,
- обчислення довжини вектора,
- порівняння довжин векторів.

## Варіант 14.

Створити клас **Money** для роботи з грошовими сумами. Число має бути представлене двома полями:

- типу **long** для гривень і
- типу **byte** – для копійок.

Дробова частина (копійки) при виводі на екран має бути відокремлена від цілої частини комою. Реалізувати методи:

- віднімання сум,
- множення на дробове число,
- операції порівняння сум.

## Варіант 15.

Створити клас **Point** для роботи з точками на площині. Координати точки – декартові.

Поля:

- $x$
- $y$

Обов'язково мають бути реалізовані:

- перетворення у полярні координати,
- визначення відстані до початку координат,
- порівняння на рівність та нерівність.

## Варіант 16.\*

Раціональний (нескоротний) дріб представляється парою цілих чисел  $(a, b)$ , де поля:

- $a$  – чисельник,
- $b$  – знаменник.

Створити клас **Rational** для роботи з раціональними дробами. Обов'язково мають бути реалізовані наступні методи:

Унарна операція (аргументом є поточний об'єкт):

- обчислення значення `value()`,  $a / b$ ;

```
double Rational::value(){
 return 1.*a/b;
}
```

```
Rational z;
```

```
...
```

```
double x = z.value();
```

бінарні операції (перший аргумент – поточний об'єкт, другий аргумент – об'єкт-параметр):

- ділення `div()`,  $(a_1, b_1) / (a_2, b_2) = (a_1 \cdot b_2, a_2 \cdot b_1)$ ;
- порівняння «чи рівне» `equal()`;
- порівняння «чи більше» `great()`;
- порівняння «чи менше» `less()`.

\* Має бути реалізована приватна функція скорочення дробу `Reduce()`, яка обов'язково викликається при виконанні арифметичних операцій.

## Пояснення Rational

Реалізація додавання за допомогою методу класу:

```

class Rational {
private:
 int a, b;
public:
 /* ... */
 Rational add(Rational& r);
};

Rational Rational::add(Rational& r) {
 Rational tmp;

 tmp.a = a * r.b + b * r.a;
 tmp.b = b * r.b;

 return tmp;
}

```

Використання додавання як методу класу:

```

Rational z1, z2, z3;
/* ... */
z3 = z1.add(z2);

```

Реалізація додавання за допомогою дружньої функції:

```

class Rational {
private:
 int a, b;
public:
 /* ... */
 friend Rational add(Rational& l, Rational& r);
};

Rational add(Rational& l, Rational& r) {
 Rational tmp;

 tmp.a = l.a * r.b + l.b * r.a;
 tmp.b = l.b * r.b;

 return tmp;
}

```

Використання додавання як дружньої функції:

```

Rational z1, z2, z3;
/* ... */
z3 = add(z1, z2);

```

## Варіант 17.

Реалізувати клас **FuzzyNumber** для роботи з нечіткими числами, які представляються трійками чисел  $(x - l, x, x + r)$ . Поля:

- $x$
- $l$
- $r$



Для чисел  $A = (x_A - l_A, x_A, x_A + r_A)$  та  $B = (x_B - l_B, x_B, x_B + r_B)$  арифметичні операції виконуються за наступними формулами:

- віднімання

$$A - B = (x_A - x_B - l_A - l_B, x_A - x_B, x_A - x_B + r_A + r_B);$$

- зворотне число

$$1 / A = (1/(x_A + r_A), 1 / x_A, 1/(x_A - l_A)), x_A > 0;$$

- ділення

$$A / B = ((x_A - l_A)/(x_B + r_B), x_A / x_B, (x_A + r_A)/(x_B - l_B)), x_B > 0.$$

### Пояснення FuzzyNumber

Нечіткі числа подаються трійками  $(x - l, x, x + r)$ , де

$x$  – координата центру,

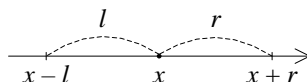
$x - l$  – координата лівої границі,

$x + r$  – координата правої границі.

Відповідно:

$l$  – відстань від лівої границі до центру,

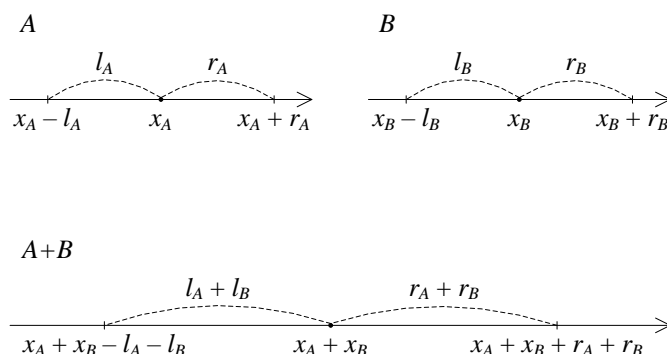
$r$  – відстань від правої границі до центру:



Клас FuzzyNumber містить три поля:  $\{x, l, r\}$ .

Додавання двох нечітких чисел  $A = (x_A - l_A, x_A, x_A + r_A)$  та  $B = (x_B - l_B, x_B, x_B + r_B)$  описується формулою

$$A + B = (x_A + x_B - l_A - l_B, x_A + x_B, x_A + x_B + r_A + r_B)$$



Тобто, для суми двох нечітких чисел  $A+B$ :

$x_A + x_B$  – координата центру,

$x_A + x_B - l_A - l_B$  – координата лівої границі,

$x_A + x_B + r_A + r_B$  – координата правої границі.

Відповідно,

$l_A + l_B$  – відстань від лівої границі до центру,

$r_A + r_B$  – відстань від правої границі до центру.

Таким чином, якщо об'єкт-число  $A$  містить поля  $\{x_A, l_A, r_A\}$ , а об'єкт-число  $B$  – поля  $\{x_B, l_B, r_B\}$ , то об'єкт-сума містить поля  $\{x_A + x_B, l_A + l_B, r_A + r_B\}$ .

### Варіант 18.

Номінали гривень можуть приймати значення 1, 2, 5, 10, 20, 50, 100, 200, 500. Копійки представити як 0.01 (1 копійка), 0.02 (2 копійки), 0.05 (5 копійок), 0.1 (10 копійок), 0.25 (25 копійок), 0.5 (50 копійок).

Створити клас **Money** для роботи з грошовими сумами. Сума має бути представлена полями-номіналами, значеннями яких має бути кількість купюр відповідного номіналу. Поля:

- кількість банкнот по 500 грн.
- кількість банкнот по 200 грн.
- кількість банкнот по 100 грн.
- кількість банкнот по 50 грн.
- кількість банкнот по 20 грн.
- кількість банкнот по 10 грн.
- кількість банкнот по 5 грн.
- кількість банкнот по 2 грн.
- кількість банкнот по 1 грн.
- кількість монет по 50 коп.
- кількість монет по 25 коп.
- кількість монет по 10 коп.
- кількість монет по 5 коп.
- кількість монет по 2 коп.
- кількість монет по 1 коп.

Реалізувати:

- ділення сум,
- ділення суми на дробове число,
- операції порівняння сум.

Дробова частина (копійки) при виводі на екран має бути відокремлена від цілої частини комою.

### Варіант 19.

Створити клас `Fraction` для роботи з дробовими числами. Число має бути представлене двома полями:

- ціла частина – довге ціле із знаком,
- дробова частина – без-знакове коротке ціле.

Реалізувати методи – арифметичні операції:

- віднімання,
- операції порівняння.

### Варіант 20.\*

Створити клас `BitString` для роботи з 64-бітовими рядками. Бітовий рядок має бути представлений двома полями типу `long`. Мають бути визначені методи, які реалізують всі традиційні операції для роботи з бітами:

- `xor`,
- \* зсув ліворуч `shiftLeft` на задану кількість бітів,
- \* зсув праворуч `shiftRight` на задану кількість бітів.

### Варіант 21.\*

Створити клас `LongLong` для роботи з 64-розрядними цілими числами. Число має бути представлене двома полями:

- типу `long` – старша частина,
- типу `long` – молодша частина.

Мають бути реалізовані методи, які представляють:

- арифметичні операції (без присвоєння):
  - \* віднімання,
  - \* ділення;
- операції порівняння:
  - не більше, дорівнює, не дорівнює.

### Варіант 22.

Створити клас `VectorN`, що задається групою  $N$  дійсних чисел – координат вектора. Поля

- $N$  – розмірність вектора,
- $a$  – масив дійсних чисел, який реалізує вектор.

Обов'язково мають бути реалізовані:

- додавання векторів,

- віднімання векторів,
- скалярний добуток векторів.

### Варіант 23.

Створити клас **VectorN**, що задається групою  $N$  дійсних чисел – координат вектора. Поля

- $N$  – розмірність вектора,
- $a$  – масив дійсних чисел, який реалізує вектор.

Обов'язково мають бути реалізовані:

- множення на скаляр,
- порівняння векторів,
- обчислення довжини вектора,
- порівняння довжин векторів.

### Варіант 24.

Створити клас **Matrix** – реалізує матрицю цілих елементів, який містить закриті поля:

- $m$  – двовимірний масив,
- $R$  – кількість рядків,
- $C$  – кількість стовпців.

Визначити методи для:

- повернення значення елемента, який має індекси  $(i, j)$ ;
- виведення матриці;
- додавання матриць;
- віднімання матриць;
- множення матриць;
- множення матриці на число.

### Варіант 25.

Розробити клас **CharLine** – реалізує рядок  $N$  символів. У закритій частині визначити поля:

- $N$  – довжина рядка (кількість символів);
- $s$  – масив, який вміщує  $N$  символів.

Визначити методи:

- введення-виведення рядка,
- виведення символу у вказаній позиції,
- перевірки входження заданого символу у рядок.

- конкатенації,
- порівняння рядків,
- перевірки входження під-рядка у рядок.

## Варіант 26.

Комплексне число представляється парою дійсних чисел  $(x, y)$ , де поля

- $x$  – дійсна частина,
- $y$  – мніма частина.

Реалізувати клас `Complex` для роботи з комплексними числами. Обов'язково мають бути реалізовані методи:

- додавання `add()`  $(x_1, y_1) + (x_2, y_2) = (x_1 + x_2, y_1 + y_2)$ ;
- множення `mul()`  $(x_1, y_1) \times (x_2, y_2) = (x_1 \cdot x_2 - y_1 \cdot y_2, x_1 \cdot y_2 + x_2 \cdot y_1)$ ;
- порівняння `equ()`  $(x_1, y_1) = (x_2, y_2)$ , якщо  $(x_1 = x_2)$  і  $(y_1 = y_2)$ .

## Варіант 27.

Створити клас `Vector3D`, що задається трійкою координат. Поля

- $x$
- $y$
- $z$

Обов'язково мають бути реалізовані:

- додавання векторів,
- віднімання векторів,
- скалярний добуток векторів.

## Варіант 28.

Створити клас `Money` для роботи з грошовими сумами. Число має бути представлене двома полями:

- типу `long` для гривень і
- типу `byte` – для копійок.

Дробова частина (копійки) при виводі на екран має бути відокремлена від цілої частини комою. Реалізувати методи:

- додавання сум,
- ділення сум,
- ділення суми на дробове число.

## Варіант 29.

Створити клас **Point** для роботи з точками на площині. Координати точки – декартові.

Поля:

- $x$
- $y$

Обов'язково мають бути реалізовані:

- переміщення точки по осі  $X$ ,
- переміщення по осі  $Y$ ,
- визначення відстані між двома точками.

## Варіант 30.\*

Раціональний (нескоротний) дріб представляється парою цілих чисел  $(a, b)$ , де поля:

- $a$  – чисельник,
- $b$  – знаменник.

Створити клас **Rational** для роботи з раціональними дробами. Обов'язково мають бути реалізовані наступні методи:

Унарна операція (аргументом є поточний об'єкт):

- обчислення значення `value()`,  $a / b$ ;  

```
double Rational::value(){
 return 1.*a/b;
}

Rational z;
...
double x = z.value();
```

Бінарні операції (перший аргумент – поточний об'єкт, другий аргумент – об'єкт-параметр):

- додавання `add()`,  $(a_1, b_1) + (a_2, b_2) = (a_1 \cdot b_2 + a_2 \cdot b_1, b_1 \cdot b_2)$ ;
- віднімання `sub()`,  $(a_1, b_1) - (a_2, b_2) = (a_1 \cdot b_2 - a_2 \cdot b_1, b_1 \cdot b_2)$ ;
- множення `mul()`,  $(a_1, b_1) \times (a_2, b_2) = (a_1 \cdot a_2, b_1 \cdot b_2)$ .

\* Має бути реалізована приватна функція скорочення дробу `Reduce()`, яка обов'язково викликається при виконанні арифметичних операцій.

## Пояснення Rational

Реалізація додавання за допомогою методу класу:

```
class Rational {
private:
```

```

 int a, b;
public:
 /* ... */
 Rational add(Rational& r);
};

Rational Rational::add(Rational& r) {
 Rational tmp;

 tmp.a = a * r.b + b * r.a;
 tmp.b = b * r.b;

 return tmp;
}

```

Використання додавання як методу класу:

```

Rational z1, z2, z3;
/* ... */
z3 = z1.add(z2);

```

Реалізація додавання за допомогою дружньої функції:

```

class Rational {
private:
 int a, b;
public:
 /* ... */
 friend Rational add(Rational& l, Rational& r);
};

Rational add(Rational& l, Rational& r) {
 Rational tmp;

 tmp.a = l.a * r.b + l.b * r.a;
 tmp.b = l.b * r.b;

 return tmp;
}

```

Використання додавання як дружньої функції:

```

Rational z1, z2, z3;
/* ... */
z3 = add(z1, z2);

```

## Варіант 31.

Реалізувати клас **FuzzyNumber** для роботи з нечіткими числами, які представляються трійками чисел  $(x - l, x, x + r)$ . Поля:

- $x$
- $l$
- $r$

Для чисел  $A = (x_A - l_A, x_A, x_A + r_A)$  та  $B = (x_B - l_B, x_B, x_B + r_B)$  арифметичні операції виконуються за наступними формулами:

- додавання

$$A + B = (x_A + x_B - l_A - l_B, x_A + x_B, x_A + x_B + r_A + r_B);$$

- множення

$$A \times B = (x_A \times x_B - x_B \times l_A - x_A \times l_B - l_A \times l_B, x_A \times x_B, x_A \times x_B + x_B \times r_A + x_A \times r_B + r_A \times r_B).$$

### Пояснення FuzzyNumber

Нечіткі числа подаються трійками  $(x - l, x, x + r)$ , де

$x$  – координата центру,

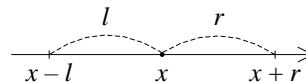
$x - l$  – координата лівої границі,

$x + r$  – координата правої границі.

Відповідно:

$l$  – відстань від лівої границі до центру,

$r$  – відстань від правої границі до центру:

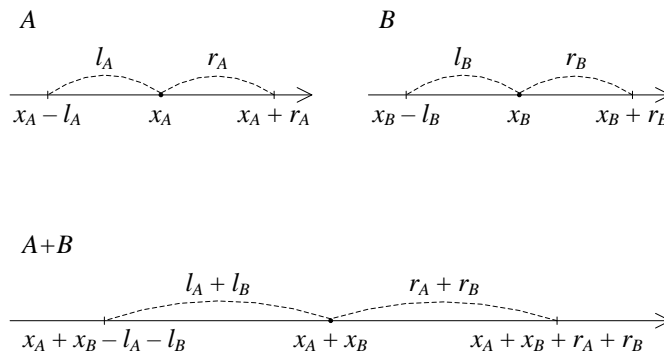


Клас FuzzyNumber містить три поля:  $\{x, l, r\}$ .

Додавання двох нечітких чисел  $A = (x_A - l_A, x_A, x_A + r_A)$  та  $B = (x_B - l_B, x_B, x_B + r_B)$

описується формулою

$$A + B = (x_A + x_B - l_A - l_B, x_A + x_B, x_A + x_B + r_A + r_B)$$



Тобто, для суми двох нечітких чисел  $A+B$ :

$x_A + x_B$  – координата центру,

$x_A + x_B - l_A - l_B$  – координата лівої границі,

$x_A + x_B + r_A + r_B$  – координата правої границі.

Відповідно,

$l_A + l_B$  – відстань від лівої границі до центру,

$r_A + r_B$  – відстань від правої границі до центру.



Таким чином, якщо об'єкт-число  $A$  містить поля  $\{x_A, l_A, r_A\}$ , а об'єкт-число  $B$  – поля  $\{x_B, l_B, r_B\}$ , то об'єкт-сума містить поля  $\{x_A + x_B, l_A + l_B, r_A + r_B\}$ .

### Варіант 32.

Номінали гривень можуть приймати значення 1, 2, 5, 10, 20, 50, 100, 200, 500. Копійки представити як 0.01 (1 копійка), 0.02 (2 копійки), 0.05 (5 копійок), 0.1 (10 копійок), 0.25 (25 копійок), 0.5 (50 копійок).

Створити клас **Money** для роботи з грошовими сумами. Сума має бути представлена полями-номіналами, значеннями яких має бути кількість купюр відповідного номіналу.

Поля:

- кількість банкнот по 500 грн.
- кількість банкнот по 200 грн.
- кількість банкнот по 100 грн.
- кількість банкнот по 50 грн.
- кількість банкнот по 20 грн.
- кількість банкнот по 10 грн.
- кількість банкнот по 5 грн.
- кількість банкнот по 2 грн.
- кількість банкнот по 1 грн.
- кількість монет по 50 коп.
- кількість монет по 25 коп.
- кількість монет по 10 коп.
- кількість монет по 5 коп.
- кількість монет по 2 коп.
- кількість монет по 1 коп.

Реалізувати:

- додавання сум,
- віднімання сум,
- множення суми на дробове число.

Дробова частина (копійки) при виводі на екран має бути відокремлена від цілої частини комою.

### Варіант 33.

Створити клас **Fraction** для роботи з дробовими числами. Число має бути представлене двома полями:

- ціла частина – довге ціле із знаком,
- дробова частина – без-знакове коротке ціле.

Реалізувати методи – арифметичні операції:

- додавання,
- множення.

#### Варіант 34.

Створити клас `BitString` для роботи з 64-бітовими рядками. Бітовий рядок має бути представлений двома полями типу `long`. Мають бути визначені методи, які реалізують всі традиційні операції для роботи з бітами:

- `not`,
- `and`,
- `or`.

#### Варіант 35.\*

Створити клас `LongLong` для роботи з 64-розрядними цілими числами. Число має бути представлене двома полями:

- типу `long` – старша частина,
- типу `long` – молодша частина.

Мають бути реалізовані методи, які представляють:

- арифметичні операції, присутні в мові програмування (без присвоєння):
  - \* додавання,
  - \* множення;
- операції порівняння:
  - менше, не менше, більше.

#### Варіант 36.

Створити клас `Vector2D`, що задається парою координат. Поля

- `x`
- `y`

Обов'язково мають бути реалізовані:

- скалярний добуток векторів,
- множення на скаляр,
- обчислення довжини вектора,
- порівняння довжин векторів.

### Варіант 37.

Комплексне число представляється парою дійсних чисел  $(x, y)$ , де поля

- $x$  – дійсна частина,
- $y$  – мніма частина.

Реалізувати клас **Complex** для роботи з комплексними числами. Обов'язково мають бути реалізовані методи:

- віднімання **sub()**  $(x_1, y_1) - (x_2, y_2) = (x_1 - x_2, y_1 - y_2)$ ;
- ділення **div()**  $(x_1, y_1) / (x_2, y_2) = (x_1 \cdot x_2 + y_1 \cdot y_2, x_2 \cdot y_1 - x_1 \cdot y_2) / (x_2^2 + y_2^2)$ ;
- комплексно спряжене число **conj()**  $\text{conj}(x, y) = (x, -y)$ .

### Варіант 38.

Створити клас **Vector3D**, що задається трійкою координат. Поля

- $x$
- $y$
- $z$

Обов'язково мають бути реалізовані:

- множення на скаляр,
- порівняння векторів,
- обчислення довжини вектора,
- порівняння довжин векторів.

### Варіант 39.

Створити клас **Money** для роботи з грошовими сумами. Число має бути представлене двома полями:

- типу **long** для гривень і
- типу **byte** – для копійок.

Дробова частина (копійки) при виводі на екран має бути відокремлена від цілої частини комою. Реалізувати методи:

- віднімання сум,
- множення на дробове число,
- операції порівняння сум.

### Варіант 40.

Створити клас **Point** для роботи з точками на площині. Координати точки – декартові.

Поля:

- $x$
- $y$

Обов'язково мають бути реалізовані:

- перетворення у полярні координати,
- визначення відстані до початку координат,
- порівняння на рівність та нерівність.

## Завдання С

Реалізувати класи із завдання свого варіанту Лабораторної роботи № 1.3 «Об'єкти – параметри методів (дії над кількома об'єктами)» як класи-нащадки (відкрите успадкування) з перевантаженням операцій та конструкторами (оскільки слід реалізувати конструктори та операції, то зручніше за основу брати завдання свого варіанту Лабораторної роботи № 2.3 «Конструктори та перевантаження операцій для класів»).

В якості базового класу реалізувати клас **Object** з лічильником кількості створених об'єктів.

### Лабораторна робота № 2.3:

В кожній лабораторній роботі цієї теми потрібно реалізувати в тому або іншому вигляді визначення нового класу. У всіх завданнях необхідно реалізувати:

- конструктор ініціалізації (один або декілька),
- конструктор без аргументів і
- конструктор копіювання.

Вказані в завданні операції реалізуються за допомогою перевантаження підходящих операцій. У всіх завданнях обов'язково мають бути реалізовані відповідні операції:

- присвоєння,
- введення з клавіатури,
- виводу на екран,
- приведення типу – перетворення у літерний рядок.

Також треба реалізувати операції

- інкременту в обох формах (префіксній та постфіксній) і
- декременту в обох формах (префіксній та постфіксній), – зміст цих операцій визначити самостійно.

Перевантаження операцій виконується таким чином: підходящі операції реалізуються як методи класу, а інші – як зовнішні дружні функції.

Для демонстрації роботи з об'єктами нового типу у всіх завданнях потрібно написати головну функцію. У програмі обов'язково мають бути продемонстровані різні способи створення об'єктів і масивів об'єктів. Програма має демонструвати використання всіх функцій і методів. Вона має виводити на екран розмір класу в режимі `#pragma pack(1)` і без нього.

Визначення класу та реалізацію його методів слід розмістити в окремих модулях.

**Завдання наступне:**

Виконати завдання свого варіанту Лабораторної роботи № 1.3. «Об'єкти – параметри методів (дії над кількома об'єктами)» як незалежні класи з конструкторами і перевантаженням операцій.

### Лабораторна робота № 1.3:

У всіх завданнях, крім вказаних в завданні операцій, обов'язково мають бути реалізовані наступні методи:

- метод ініціалізації `Init( )`;
- метод введення з клавіатури `Read( )`;
- метод виведення на екран `Display( )`;
- метод перетворення до літерного рядку `toString( )`.

Всі завдання мають бути реалізовані як клас із закритими полями, де операції реалізуються як методи класу.

Визначення класу та реалізацію його методів слід розмістити в окремих модулях.

Для демонстрації роботи з об'єктами нового типу у всіх завданнях потрібно написати головну функцію. У програмі обов'язково мають бути продемонстровані різні способи створення об'єктів і масивів об'єктів – різними конструкторами. Програма має демонструвати використання всіх функцій і методів.

Варіанти завдань наступні:

#### Варіант 1.

Комплексне число представляється парою дійсних чисел  $(x, y)$ , де поля

- $x$  – дійсна частина,
- $y$  – мніма частина.

Реалізувати клас `Complex` для роботи з комплексними числами. Обов'язково мають бути реалізовані методи:

- додавання `add()`  $(x_1, y_1) + (x_2, y_2) = (x_1 + x_2, y_1 + y_2)$ ;
- множення `mul()`  $(x_1, y_1) \times (x_2, y_2) = (x_1 \cdot x_2 - y_1 \cdot y_2, x_1 \cdot y_2 + x_2 \cdot y_1)$ ;
- порівняння `equ()`  $(x_1, y_1) = (x_2, y_2)$ , якщо  $(x_1 = x_2)$  і  $(y_1 = y_2)$ .

#### Варіант 2.

Створити клас `Vector3D`, що задається трійкою координат. Поля

- $x$
- $y$
- $z$

Обов'язково мають бути реалізовані:

- додавання векторів,
- віднімання векторів,
- скалярний добуток векторів.

### Варіант 3.

Створити клас **Money** для роботи з грошовими сумами. Число має бути представлене двома полями:

- типу **long** для гривень і
- типу **byte** – для копійок.

Дробова частина (копійки) при виводі на екран має бути відокремлена від цілої частини комою. Реалізувати методи:

- додавання сум,
- ділення сум,
- ділення суми на дробове число.

### Варіант 4.

Створити клас **Point** для роботи з точками на площині. Координати точки – декартові.

Поля:

- $x$
- $y$

Обов'язково мають бути реалізовані:

- переміщення точки по осі  $X$ ,
- переміщення по осі  $Y$ ,
- визначення відстані між двома точками.

### Варіант 5.\*

Раціональний (нескоротний) дріб представляється парою цілих чисел  $(a, b)$ , де поля:

- $a$  – чисельник,
- $b$  – знаменник.

Створити клас **Rational** для роботи з раціональними дробами. Обов'язково мають бути реалізовані наступні методи:

Унарна операція (аргументом є поточний об'єкт):

- обчислення значення `value()`,  $a / b$ ;  
`double Rational::value(){`

```

 return 1.*a/b;
 }

 Rational z;
 ...
 double x = z.value();

```

бінарні операції (перший аргумент – поточний об’єкт, другий аргумент – об’єкт-параметр):

- додавання  $\text{add}()$ ,  $(a_1, b_1) + (a_2, b_2) = (a_1 \cdot b_2 + a_2 \cdot b_1, b_1 \cdot b_2)$ ;
- віднімання  $\text{sub}()$ ,  $(a_1, b_1) - (a_2, b_2) = (a_1 \cdot b_2 - a_2 \cdot b_1, b_1 \cdot b_2)$ ;
- множення  $\text{mul}()$ ,  $(a_1, b_1) \times (a_2, b_2) = (a_1 \cdot a_2, b_1 \cdot b_2)$ .

\* Має бути реалізована приватна функція скорочення дробу `Reduce()`, яка обов’язково викликається при виконанні арифметичних операцій.

## Пояснення Rational

Реалізація додавання за допомогою методу класу:

```

class Rational {
private:
 int a, b;
public:
 /* ... */
 Rational add(Rational& r);
};

Rational Rational::add(Rational& r) {
 Rational tmp;

 tmp.a = a * r.b + b * r.a;
 tmp.b = b * r.b;

 return tmp;
}

```

Використання додавання як методу класу:

```

Rational z1, z2, z3;
/* ... */
z3 = z1.add(z2);

```

Реалізація додавання за допомогою дружньої функції:

```

class Rational {
private:
 int a, b;
public:
 /* ... */
 friend Rational add(Rational& l, Rational& r);
};

Rational add(Rational& l, Rational& r) {
 Rational tmp;

 tmp.a = l.a * r.b + l.b * r.a;

```



```

 tmp.b = l.b * r.b;

 return tmp;
}

```

Використання додавання як дружньої функції:

```

Rational z1, z2, z3;
/* ... */
z3 = add(z1, z2);

```

## Варіант 6.

Реалізувати клас **FuzzyNumber** для роботи з нечіткими числами, які представляються трійками чисел  $(x - l, x, x + r)$ . Поля:

- $x$
- $l$
- $r$

Для чисел  $A = (x_A - l_A, x_A, x_A + r_A)$  та  $B = (x_B - l_B, x_B, x_B + r_B)$  арифметичні операції виконуються за наступними формулами:

- додавання

$$A + B = (x_A + x_B - l_A - l_B, x_A + x_B, x_A + x_B + r_A + r_B);$$

- множення

$$A \times B = (x_A \times x_B - x_B \times l_A - x_A \times l_B - l_A \times l_B, x_A \times x_B, x_A \times x_B + x_B \times r_A + x_A \times r_B + r_A \times r_B).$$

## Пояснення FuzzyNumber

Нечіткі числа подаються трійками  $(x - l, x, x + r)$ , де

$x$  – координата центру,

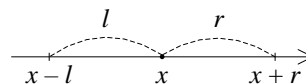
$x - l$  – координата лівої границі,

$x + r$  – координата правої границі.

Відповідно:

$l$  – відстань від лівої границі до центру,

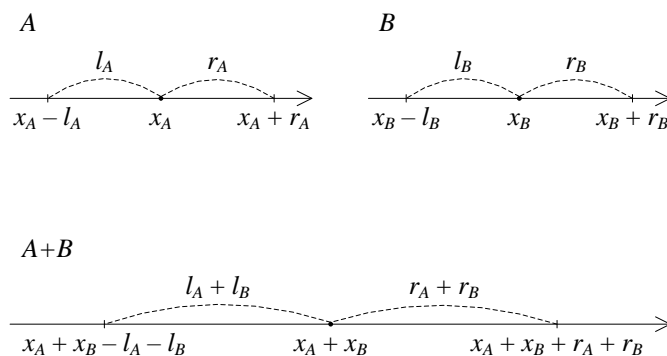
$r$  – відстань від правої границі до центру:



Клас **FuzzyNumber** містить три поля:  $\{x, l, r\}$ .

Додавання двох нечітких чисел  $A = (x_A - l_A, x_A, x_A + r_A)$  та  $B = (x_B - l_B, x_B, x_B + r_B)$  описується формулою

$$A + B = (x_A + x_B - l_A - l_B, x_A + x_B, x_A + x_B + r_A + r_B)$$



Тобто, для суми двох нечітких чисел  $A+B$ :

$x_A + x_B$  — координата центру,  
 $x_A + x_B - l_A - l_B$  — координата лівої границі,  
 $x_A + x_B + r_A + r_B$  — координата правої границі.

Відповідно,

$l_A + l_B$  — відстань від лівої границі до центру,  
 $r_A + r_B$  — відстань від правої границі до центру.

Таким чином, якщо об'єкт-число  $A$  містить поля  $\{x_A, l_A, r_A\}$ , а об'єкт-число  $B$  — поля  $\{x_B, l_B, r_B\}$ , то об'єкт-сума містить поля  $\{x_A + x_B, l_A + l_B, r_A + r_B\}$ .

## Варіант 7.

Номінали гривень можуть приймати значення 1, 2, 5, 10, 20, 50, 100, 200, 500. Копійки представити як 0.01 (1 копійка), 0.02 (2 копійки), 0.05 (5 копійок), 0.1 (10 копійок), 0.25 (25 копійок), 0.5 (50 копійок).

Створити клас **Money** для роботи з грошовими сумами. Сума має бути представлена полями-номіналами, значеннями яких має бути кількість купюр відповідного номіналу. Поля:

- кількість банкнот по 500 грн.
- кількість банкнот по 200 грн.
- кількість банкнот по 100 грн.
- кількість банкнот по 50 грн.
- кількість банкнот по 20 грн.
- кількість банкнот по 10 грн.
- кількість банкнот по 5 грн.
- кількість банкнот по 2 грн.
- кількість банкнот по 1 грн.
- кількість монет по 50 коп.

- кількість монет по 25 коп.
- кількість монет по 10 коп.
- кількість монет по 5 коп.
- кількість монет по 2 коп.
- кількість монет по 1 коп.

Реалізувати:

- додавання сум,
- віднімання сум,
- множення суми на дробове число.

Дробова частина (копійки) при виводі на екран має бути відокремлена від цілої частини комою.

### Варіант 8.

Створити клас `Fraction` для роботи з дробовими числами. Число має бути представлене двома полями:

- ціла частина – довге ціле із знаком,
- дробова частина – без-знакове коротке ціле.

Реалізувати методи – арифметичні операції:

- додавання,
- множення.

### Варіант 9.

Створити клас `BitString` для роботи з 64-бітовими рядками. Бітовий рядок має бути представлений двома полями типу `long`. Мають бути визначені методи, які реалізують всі традиційні операції для роботи з бітами:

- `not`,
- `and`,
- `or`.

### Варіант 10.\*

Створити клас `LongLong` для роботи з 64-розрядними цілими числами. Число має бути представлене двома полями:

- типу `long` – старша частина,
- типу `long` – молодша частина.

Мають бути реалізовані методи, які представляють:

- арифметичні операції, присутні в мові програмування (без присвоєння):
  - \* додавання,
  - \* множення;
- операції порівняння:
  - менше, не менше, більше.

### Варіант 11.

Створити клас **Vector2D**, що задається парою координат. Поля

- $x$
- $y$

Обов'язково мають бути реалізовані:

- скалярний добуток векторів,
- множення на скаляр,
- обчислення довжини вектора,
- порівняння довжин векторів.

### Варіант 12.

Комплексне число представляються парою дійсних чисел  $(x, y)$ , де поля

- $x$  – дійсна частина,
- $y$  – мніма частина.

Реалізувати клас **Complex** для роботи з комплексними числами. Обов'язково мають бути реалізовані методи:

- віднімання **sub()**  $(x_1, y_1) - (x_2, y_2) = (x_1 - x_2, y_1 - y_2)$ ;
- ділення **div()**  $(x_1, y_1) / (x_2, y_2) = (x_1 \cdot x_2 + y_1 \cdot y_2, x_2 \cdot y_1 - x_1 \cdot y_2) / (x_2^2 + y_2^2)$ ;
- комплексно спряжене число **conj()**  $\text{conj}(x, y) = (x, -y)$ .

### Варіант 13.

Створити клас **Vector3D**, що задається трійкою координат. Поля

- $x$
- $y$
- $z$

Обов'язково мають бути реалізовані:

- множення на скаляр,
- порівняння векторів,
- обчислення довжини вектора,

- порівняння довжин векторів.

#### Варіант 14.

Створити клас **Money** для роботи з грошовими сумами. Число має бути представлене двома полями:

- типу `long` для гривень і
- типу `byte` – для копійок.

Дробова частина (копійки) при виводі на екран має бути відокремлена від цілої частини комою. Реалізувати методи:

- віднімання сум,
- множення на дробове число,
- операції порівняння сум.

#### Варіант 15.

Створити клас **Point** для роботи з точками на площині. Координати точки – декартові.

Поля:

- $x$
- $y$

Обов'язково мають бути реалізовані:

- перетворення у полярні координати,
- визначення відстані до початку координат,
- порівняння на рівність та нерівність.

#### Варіант 16.\*

Раціональний (нескоротний) дріб представляється парою цілих чисел  $(a, b)$ , де поля:

- $a$  – чисельник,
- $b$  – знаменник.

Створити клас **Rational** для роботи з раціональними дробами. Обов'язково мають бути реалізовані наступні методи:

Унарна операція (аргументом є поточний об'єкт):

- обчислення значення `value()`,  $a / b$ ;

```
double Rational::value(){
 return 1.*a/b;
}
```

```
Rational z;
...
```

```
double x = z.value();
```

бінарні операції (перший аргумент – поточний об’єкт, другий аргумент – об’єкт-параметр):

- ділення `div()`,  $(a_1, b_1) / (a_2, b_2) = (a_1 \cdot b_2, a_2 \cdot b_1)$ ;
- порівняння «чи рівне» `equal()`;
- порівняння «чи більше» `great()`;
- порівняння «чи менше» `less()`.

\* Має бути реалізована приватна функція скорочення дробу `Reduce()`, яка обов’язково викликається при виконанні арифметичних операцій.

## Пояснення Rational

Реалізація додавання за допомогою методу класу:

```
class Rational
{
private:
 int a, b;
public:
 /* ... */
 Rational add(Rational& r);
};

Rational Rational::add(Rational& r)
{
 Rational tmp;

 tmp.a = a * r.b + b * r.a;
 tmp.b = b * r.b;

 return tmp;
}
```

Використання додавання як методу класу:

```
Rational z1, z2, z3;
/* ... */
z3 = z1.add(z2);
```

Реалізація додавання за допомогою дружньої функції:

```
class Rational
{
private:
 int a, b;
public:
 /* ... */
 friend Rational add(Rational& l, Rational& r);
};

Rational add(Rational& l, Rational& r)
{
 Rational tmp;
```

```

 tmp.a = l.a * r.b + l.b * r.a;
 tmp.b = l.b * r.b;

 return tmp;
 }

```

Використання додавання як дружньої функції:

```

Rational z1, z2, z3;
/* ... */
z3 = add(z1, z2);

```

## Варіант 17.

Реалізувати клас **FuzzyNumber** для роботи з нечіткими числами, які представляються трійками чисел  $(x - l, x, x + r)$ . Поля:

- $x$
- $l$
- $r$

Для чисел  $A = (x_A - l_A, x_A, x_A + r_A)$  та  $B = (x_B - l_B, x_B, x_B + r_B)$  арифметичні операції виконуються за наступними формулами:

- віднімання

$$A - B = (x_A - x_B - l_A - l_B, x_A - x_B, x_A - x_B + r_A + r_B);$$

- зворотне число

$$1 / A = (1/(x_A + r_A), 1 / x_A, 1/(x_A - l_A)), x_A > 0;$$

- ділення

$$A / B = ((x_A - l_A)/(x_B + r_B), x_A / x_B, (x_A + r_A)/(x_B - l_B)), x_B > 0.$$

## Пояснення FuzzyNumber

Нечіткі числа подаються трійками  $(x - l, x, x + r)$ , де

$x$  – координата центру,

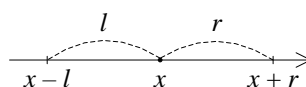
$x - l$  – координата лівої границі,

$x + r$  – координата правої границі.

Відповідно:

$l$  – відстань від лівої границі до центру,

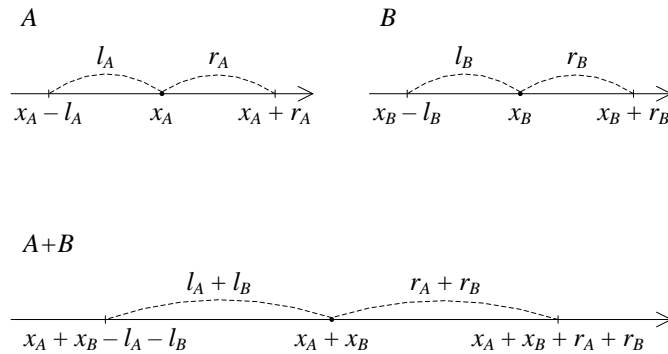
$r$  – відстань від правої границі до центру:



Клас **FuzzyNumber** містить три поля:  $\{x, l, r\}$ .

Додавання двох нечітких чисел  $A = (x_A - l_A, x_A, x_A + r_A)$  та  $B = (x_B - l_B, x_B, x_B + r_B)$  описується формулою

$$A + B = (x_A + x_B - l_A - l_B, x_A + x_B, x_A + x_B + r_A + r_B)$$



Тобто, для суми двох нечітких чисел  $A+B$ :

- $x_A + x_B$  — координата центру,
- $x_A + x_B - l_A - l_B$  — координата лівої границі,
- $x_A + x_B + r_A + r_B$  — координата правої границі.

Відповідно,

- $l_A + l_B$  — відстань від лівої границі до центру,
- $r_A + r_B$  — відстань від правої границі до центру.

Таким чином, якщо об'єкт-число  $A$  містить поля  $\{x_A, l_A, r_A\}$ , а об'єкт-число  $B$  — поля  $\{x_B, l_B, r_B\}$ , то об'єкт-сума містить поля  $\{x_A + x_B, l_A + l_B, r_A + r_B\}$ .

## Варіант 18.

Номінали гривень можуть приймати значення 1, 2, 5, 10, 20, 50, 100, 200, 500. Копійки представити як 0.01 (1 копійка), 0.02 (2 копійки), 0.05 (5 копійок), 0.1 (10 копійок), 0.25 (25 копійок), 0.5 (50 копійок).

Створити клас `Money` для роботи з грошовими сумами. Сума має бути представлена полями-номіналами, значеннями яких має бути кількість купюр відповідного номіналу. Поля:

- кількість банкнот по 500 грн.
- кількість банкнот по 200 грн.
- кількість банкнот по 100 грн.
- кількість банкнот по 50 грн.
- кількість банкнот по 20 грн.
- кількість банкнот по 10 грн.
- кількість банкнот по 5 грн.
- кількість банкнот по 2 грн.
- кількість банкнот по 1 грн.



- кількість монет по 50 коп.
- кількість монет по 25 коп.
- кількість монет по 10 коп.
- кількість монет по 5 коп.
- кількість монет по 2 коп.
- кількість монет по 1 коп.

Реалізувати:

- ділення сум,
- ділення суми на дробове число,
- операції порівняння сум.

Дробова частина (копійки) при виводі на екран має бути відокремлена від цілої частини комою.

### Варіант 19.

Створити клас `Fraction` для роботи з дробовими числами. Число має бути представлене двома полями:

- ціла частина – довге ціле із знаком,
- дробова частина – без-знакове коротке ціле.

Реалізувати методи – арифметичні операції:

- віднімання,
- операції порівняння.

### Варіант 20.\*

Створити клас `BitString` для роботи з 64-бітовими рядками. Бітовий рядок має бути представлений двома полями типу `long`. Мають бути визначені методи, які реалізують всі традиційні операції для роботи з бітами:

- `xor`,
- \* зсув ліворуч `shiftLeft` на задану кількість бітів,
- \* зсув праворуч `shiftRight` на задану кількість бітів.

### Варіант 21.\*

Створити клас `LongLong` для роботи з 64-розрядними цілими числами. Число має бути представлене двома полями:

- типу `long` – старша частина,
- типу `long` – молодша частина.

Мають бути реалізовані методи, які представляють:

- арифметичні операції (без присвоєння):
  - \* віднімання,
  - \* ділення;
- операції порівняння:
  - не більше, дорівнює, не дорівнює.

## Варіант 22.

Створити клас **VectorN**, що задається групою  $N$  дійсних чисел – координат вектора. Поля

- $N$  – розмірність вектора,
- $a$  – масив дійсних чисел, який реалізує вектор.

Обов'язково мають бути реалізовані:

- додавання векторів,
- віднімання векторів,
- скалярний добуток векторів.

## Варіант 23.

Створити клас **VectorN**, що задається групою  $N$  дійсних чисел – координат вектора. Поля

- $N$  – розмірність вектора,
- $a$  – масив дійсних чисел, який реалізує вектор.

Обов'язково мають бути реалізовані:

- множення на скаляр,
- порівняння векторів,
- обчислення довжини вектора,
- порівняння довжин векторів.

## Варіант 24.

Створити клас **Matrix** – реалізує матрицю цілих елементів, який містить закриті поля:

- $m$  – двовимірний масив,
- $R$  – кількість рядків,
- $C$  – кількість стовпців.

Визначити методи для:

- повернення значення елемента, який має індекси  $(i, j)$ ;
- виведення матриці;
- додавання матриць;

- віднімання матриць;
- множення матриць;
- множення матриці на число.

### Варіант 25.

Розробити клас `CharLine` – реалізує рядок  $N$  символів. У закритій частині визначити поля:

- $N$  – довжина рядка (кількість символів);
- $s$  – масив, який вміщує  $N$  символів.

Визначити методи:

- введення-виведення рядка,
- виведення символу у вказаній позиції,
- перевірки входження заданого символу у рядок.
- конкатенації,
- порівняння рядків,
- перевірки входження під-рядка у рядок.

### Варіант 26.

Комплексне число представляються парою дійсних чисел  $(x, y)$ , де поля

- $x$  – дійсна частина,
- $y$  – мніма частина.

Реалізувати клас `Complex` для роботи з комплексними числами. Обов'язково мають бути реалізовані методи:

- додавання `add()`  $(x_1, y_1) + (x_2, y_2) = (x_1 + x_2, y_1 + y_2)$ ;
- множення `mul()`  $(x_1, y_1) \times (x_2, y_2) = (x_1 \cdot x_2 - y_1 \cdot y_2, x_1 \cdot y_2 + x_2 \cdot y_1)$ ;
- порівняння `equ()`  $(x_1, y_1) = (x_2, y_2)$ , якщо  $(x_1 = x_2)$  і  $(y_1 = y_2)$ .

### Варіант 27.

Створити клас `Vector3D`, що задається трійкою координат. Поля

- $x$
- $y$
- $z$

Обов'язково мають бути реалізовані:

- додавання векторів,
- віднімання векторів,

- скалярний добуток векторів.

### Варіант 28.

Створити клас **Money** для роботи з грошовими сумами. Число має бути представлене двома полями:

- типу **long** для гривень і
- типу **byte** – для копійок.

Дробова частина (копійки) при виводі на екран має бути відокремлена від цілої частини комою. Реалізувати методи:

- додавання сум,
- ділення сум,
- ділення суми на дробове число.

### Варіант 29.

Створити клас **Point** для роботи з точками на площині. Координати точки – декартові.

Поля:

- $x$
- $y$

Обов'язково мають бути реалізовані:

- переміщення точки по осі  $X$ ,
- переміщення по осі  $Y$ ,
- визначення відстані між двома точками.

### Варіант 30.\*

Раціональний (нескоротний) дріб представляється парою цілих чисел  $(a, b)$ , де поля:

- $a$  – чисельник,
- $b$  – знаменник.

Створити клас **Rational** для роботи з раціональними дробами. Обов'язково мають бути реалізовані наступні методи:

Унарна операція (аргументом є поточний об'єкт):

- обчислення значення `value()`,  $a / b$ ;

```
double Rational::value(){
 return 1.*a/b;
}

Rational z;
...
```

```
double x = z.value();
```

бінарні операції (перший аргумент – поточний об'єкт, другий аргумент – об'єкт-параметр):

- додавання  $\text{add}()$ ,  $(a_1, b_1) + (a_2, b_2) = (a_1 \cdot b_2 + a_2 \cdot b_1, b_1 \cdot b_2)$ ;
- віднімання  $\text{sub}()$ ,  $(a_1, b_1) - (a_2, b_2) = (a_1 \cdot b_2 - a_2 \cdot b_1, b_1 \cdot b_2)$ ;
- множення  $\text{mul}()$ ,  $(a_1, b_1) \times (a_2, b_2) = (a_1 \cdot a_2, b_1 \cdot b_2)$ .

\* Має бути реалізована приватна функція скорочення дробу `Reduce()`, яка обов'язково викликається при виконанні арифметичних операцій.

## Пояснення Rational

Реалізація додавання за допомогою методу класу:

```
class Rational {
private:
 int a, b;
public:
 /* ... */
 Rational add(Rational& r);
};

Rational Rational::add(Rational& r) {
 Rational tmp;

 tmp.a = a * r.b + b * r.a;
 tmp.b = b * r.b;

 return tmp;
}
```

Використання додавання як методу класу:

```
Rational z1, z2, z3;
/* ... */
z3 = z1.add(z2);
```

Реалізація додавання за допомогою дружньої функції:

```
class Rational {
private:
 int a, b;
public:
 /* ... */
 friend Rational add(Rational& l, Rational& r);
};

Rational add(Rational& l, Rational& r) {
 Rational tmp;

 tmp.a = l.a * r.b + l.b * r.a;
 tmp.b = l.b * r.b;

 return tmp;
}
```

Використання додавання як дружньої функції:

```
Rational z1, z2, z3;
/* ... */
z3 = add(z1, z2);
```

### Варіант 31.

Реалізувати клас **FuzzyNumber** для роботи з нечіткими числами, які представляються трійками чисел  $(x - l, x, x + r)$ . Поля:

- $x$
- $l$
- $r$

Для чисел  $A = (x_A - l_A, x_A, x_A + r_A)$  та  $B = (x_B - l_B, x_B, x_B + r_B)$  арифметичні операції виконуються за наступними формулами:

- додавання

$$A + B = (x_A + x_B - l_A - l_B, x_A + x_B, x_A + x_B + r_A + r_B);$$

- множення

$$A \times B = (x_A \times x_B - x_B \times l_A - x_A \times l_B - l_A \times l_B, x_A \times x_B, x_A \times x_B + x_B \times r_A + x_A \times r_B + r_A \times r_B).$$

### Пояснення FuzzyNumber

Нечіткі числа подаються трійками  $(x - l, x, x + r)$ , де

$x$  — координата центру,

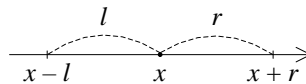
$x - l$  — координата лівої границі,

$x + r$  — координата правої границі.

Відповідно:

$l$  — відстань від лівої границі до центру,

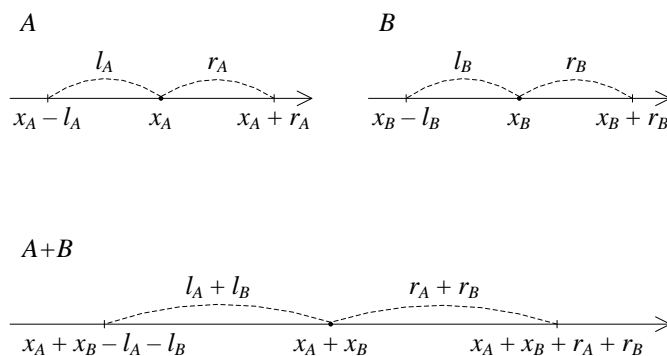
$r$  — відстань від правої границі до центру:



Клас **FuzzyNumber** містить три поля:  $\{x, l, r\}$ .

Додавання двох нечітких чисел  $A = (x_A - l_A, x_A, x_A + r_A)$  та  $B = (x_B - l_B, x_B, x_B + r_B)$  описується формулою

$$A + B = (x_A + x_B - l_A - l_B, x_A + x_B, x_A + x_B + r_A + r_B)$$



Тобто, для суми двох нечітких чисел  $A+B$ :

$x_A + x_B$  — координата центру,  
 $x_A + x_B - l_A - l_B$  — координата лівої границі,  
 $x_A + x_B + r_A + r_B$  — координата правої границі.

Відповідно,

$l_A + l_B$  — відстань від лівої границі до центру,  
 $r_A + r_B$  — відстань від правої границі до центру.

Таким чином, якщо об'єкт-число  $A$  містить поля  $\{x_A, l_A, r_A\}$ , а об'єкт-число  $B$  — поля  $\{x_B, l_B, r_B\}$ , то об'єкт-сума містить поля  $\{x_A + x_B, l_A + l_B, r_A + r_B\}$ .

### Варіант 32.

Номінали гривень можуть приймати значення 1, 2, 5, 10, 20, 50, 100, 200, 500. Копійки представити як 0.01 (1 копійка), 0.02 (2 копійки), 0.05 (5 копійок), 0.1 (10 копійок), 0.25 (25 копійок), 0.5 (50 копійок).

Створити клас **Money** для роботи з грошовими сумами. Сума має бути представлена полями-номіналами, значеннями яких має бути кількість купюр відповідного номіналу.

Поля:

- кількість банкнот по 500 грн.
- кількість банкнот по 200 грн.
- кількість банкнот по 100 грн.
- кількість банкнот по 50 грн.
- кількість банкнот по 20 грн.
- кількість банкнот по 10 грн.
- кількість банкнот по 5 грн.
- кількість банкнот по 2 грн.
- кількість банкнот по 1 грн.
- кількість монет по 50 коп.

- кількість монет по 25 коп.
- кількість монет по 10 коп.
- кількість монет по 5 коп.
- кількість монет по 2 коп.
- кількість монет по 1 коп.

Реалізувати:

- додавання сум,
- віднімання сум,
- множення суми на дробове число.

Дробова частина (копійки) при виводі на екран має бути відокремлена від цілої частини комою.

### Варіант 33.

Створити клас `Fraction` для роботи з дробовими числами. Число має бути представлене двома полями:

- ціла частина – довге ціле із знаком,
- дробова частина – без-знакове коротке ціле.

Реалізувати методи – арифметичні операції:

- додавання,
- множення.

### Варіант 34.

Створити клас `BitString` для роботи з 64-бітовими рядками. Бітовий рядок має бути представлений двома полями типу `long`. Мають бути визначені методи, які реалізують всі традиційні операції для роботи з бітами:

- `not`,
- `and`,
- `or`.

### Варіант 35.\*

Створити клас `LongLong` для роботи з 64-розрядними цілими числами. Число має бути представлене двома полями:

- типу `long` – старша частина,
- типу `long` – молодша частина.

Мають бути реалізовані методи, які представляють:



- арифметичні операції, присутні в мові програмування (без присвоєння):
  - \* додавання,
  - \* множення;
- операції порівняння:
  - менше, не менше, більше.

### Варіант 36.

Створити клас **Vector2D**, що задається парою координат. Поля

- $x$
- $y$

Обов'язково мають бути реалізовані:

- скалярний добуток векторів,
- множення на скаляр,
- обчислення довжини вектора,
- порівняння довжин векторів.

### Варіант 37.

Комплексне число представляються парою дійсних чисел  $(x, y)$ , де поля

- $x$  – дійсна частина,
- $y$  – мніма частина.

Реалізувати клас **Complex** для роботи з комплексними числами. Обов'язково мають бути реалізовані методи:

- віднімання **sub()**  $(x_1, y_1) - (x_2, y_2) = (x_1 - x_2, y_1 - y_2)$ ;
- ділення **div()**  $(x_1, y_1) / (x_2, y_2) = (x_1 \cdot x_2 + y_1 \cdot y_2, x_2 \cdot y_1 - x_1 \cdot y_2) / (x_2^2 + y_2^2)$ ;
- комплексно спряжене число **conj()**  $\text{conj}(x, y) = (x, -y)$ .

### Варіант 38.

Створити клас **Vector3D**, що задається трійкою координат. Поля

- $x$
- $y$
- $z$

Обов'язково мають бути реалізовані:

- множення на скаляр,
- порівняння векторів,
- обчислення довжини вектора,

- порівняння довжин векторів.

### Варіант 39.

Створити клас **Money** для роботи з грошовими сумами. Число має бути представлене двома полями:

- типу **long** для гривень і
- типу **byte** – для копійок.

Дробова частина (копійки) при виводі на екран має бути відокремлена від цілої частини комою. Реалізувати методи:

- віднімання сум,
- множення на дробове число,
- операції порівняння сум.

### Варіант 40.

Створити клас **Point** для роботи з точками на площині. Координати точки – декартові.

Поля:

- $x$
- $y$

Обов'язково мають бути реалізовані:

- перетворення у полярні координати,
- визначення відстані до початку координат,
- порівняння на рівність та нерівність.

## Завдання D

Реалізувати основні класи із завдання свого варіанту Лабораторної роботи № 1.5 «Композиція класів та об'єктів» з конструкторами та перевантаженням операцій як класи-нащадки від класів із завдання Лабораторної роботи № 1.5 (агрегований клас слід використовувати як базовий, а клас-контейнер стане похідним класом).

Оскільки слід реалізувати конструктори та операції, то зручніше за основу брати завдання свого варіанту Лабораторної роботи № 2.5 «Конструктори та перевантаження операцій для класів з композицією».

Реалізувати два види успадкування – відкрите та закрите.

### Лабораторна робота № 2.5:

В кожній лабораторній роботі цієї теми потрібно реалізувати в тому або іншому вигляді визначення нового класу. У всіх завданнях необхідно реалізувати:

- конструктор ініціалізації (один або декілька),
- конструктор без аргументів і
- конструктор копіювання.

Вказані в завданні операції реалізуються за допомогою перевантаження підходящих операцій. У всіх завданнях обов'язково мають бути реалізовані відповідні операції:

- присвоєння,
- введення з клавіатури,
- виводу на екран,
- приведення типу – перетворення у літерний рядок.

Також треба реалізувати операції

- інкременту в обох формах (префіксній та постфіксній) і
- декременту в обох формах (префіксній та постфіксній), – для числових полів (наприклад: так, як вказано у варіантах завдань Лабораторної роботи № 2.3).

Перевантаження операцій виконується таким чином: підходящі операції реалізуються як методи класу, а інші – як зовнішні дружні функції.

Для демонстрації роботи з об'єктами нового типу у всіх завданнях потрібно написати головну функцію. У програмі обов'язково мають бути продемонстровані різні способи створення об'єктів і масивів об'єктів. Програма має демонструвати використання всіх функцій і методів. Вона має виводити на екран розмір класу в режимі `#pragma pack(1)` і без нього.

Визначення класів та реалізації методів слід розмістити в окремих модулях.

### **Завдання наступне:**

Виконати завдання свого варіанту Лабораторної роботи № 1.5 (Композиція класів та об'єктів) з конструкторами і перевантаженням операцій.

### **Лабораторна робота № 1.5:**

У всіх завданнях потрібно реалізувати по два-три класи. Один клас є «контейнером», всі решту – описують поля, які містяться в «контейнері». Класи, що описують поля класу-«контейнера», мають бути визначені як незалежні.

Визначення класів та реалізації методів слід розмістити в окремих модулях.

Варіанти завдань наступні:

#### **Варіант 1.**

Створити клас **Car** (машина), що характеризується торговою маркою (літерний рядок), кількістю циліндрів, потужністю. Визначити методи пере-присвоєння та зміни потужності.

Створити похідний клас **Lorry** (вантажівка), що характеризується також вантажопідйомністю кузова. Визначити функції пере-присвоєння марки та зміни вантажопідйомності.

#### **Варіант 2.**

Створити клас **Pair** (пара чисел); визначити методи зміни полів і порівняння пар: пара p1 більше пари p2, якщо

$(p1.first > p2.first)$  або  $(p1.first = p2.first)$  і  $(p1.second > p2.second)$ .

Визначити похідний клас **Fraction**. Визначити повний набір методів порівняння.

#### **Варіант 3.**

Створити клас **Liquid** (рідина), що має поля «назва» і «густина». Визначити методи перепризначення і зміни густини.

Створити похідний клас **Alcohol** (спирт), що містить поле «міцність». Визначити методи пере-присвоєння та зміни міцності.

#### **Варіант 4.**

Створити клас **Pair** (пара чисел); визначити методи зміни полів та обчислення добутку чисел.

Визначити похідний клас **Rectangle** (прямокутник). Визначити методи обчислення периметру та площі прямокутника.

### Варіант 5.

Створити клас **Man** (людина) з полями: ім'я, вік, стать і вага. Визначити методи пере-присвоєння імені, зміни віку і зміни ваги.

Створити похідний клас **Student**, що має поле «рік навчання». Визначити методи пере-присвоєння та збільшення року навчання.

### Варіант 6.

Створити клас **Triad** (трійка чисел); визначити методи зміни полів і обчислення суми чисел.

Визначити похідний клас **Triangle** (трикутник). Визначити методи обчислення кутів і площі трикутника.

### Варіант 7.

Створити клас **Triangle** (трикутник) з полями-сторонами. Визначити методи зміни сторін, обчислення кутів, обчислення периметру.

Створити похідний клас **Equilateral** (рівносторонній трикутник), що має поле «площа». Визначити метод обчислення площі.

### Варіант 8.

Створити клас **Triangle** (трикутник) з полями-сторонами. Визначити методи зміни сторін, обчислення кутів, обчислення периметру.

Створити похідний клас **RightAngled** (прямокутний трикутник), що має поле «площа». Визначити метод обчислення площі.

### Варіант 9.

Створити клас **Pair** (пара чисел); визначити методи зміни полів і обчислення добутку чисел.

Визначити похідний клас **RightAngled** (прямокутний трикутник). Визначити методи обчислення гіпотенузи і площі трикутника.

### Варіант 10.

Створити клас **Triad** (трійка чисел); визначити метод порівняння тріад:  
тріада **t1** більше тріади **t2**, якщо

$(t1.first > t2.first)$  або  $(t1.first = t2.first)$  і  $(t1.second > t2.second)$

або  $(t1.first = t2.first)$  і  $(t1.second = t2.second)$  і  $(t1.third > t2.third)$ .

Визначити похідний клас **Date**. Визначити повний набір методів порівняння дат.

### Варіант 11.

Створити клас **Triad** (трійка чисел); визначити метод порівняння тріад:  
тріада **t1** більше тріади **t2**, якщо

(**t1.first** > **t2.first**) або (**t1.first** = **t2.first**) і (**t1.second** > **t2.second**)  
або (**t1.first** = **t2.first**) і (**t1.second** = **t2.second**) і (**t1.third** > **t2.third**).

Визначити похідний клас **Time**. Визначити повний набір методів порівняння моментів часу.

### Варіант 12.

Реалізувати клас-оболонку **Number** для числового типу **float**. Реалізувати методи додавання і ділення.

Створити похідний клас **Real**, в класі **Real** реалізувати метод піднесення до довільного степеню, та метод для обчислення логарифму числа.

### Варіант 13.

Створити клас **Triad** (трійка чисел); визначити методи збільшення полів на 1.

Визначити похідний клас **Date**. Перевизначити методи збільшення полів на 1 і визначити метод збільшення дати на *n* днів.

### Варіант 14.

Реалізувати клас-оболонку **Number** для числового типу **double**. Реалізувати методи множення і віднімання.

Створити похідний клас **Real**, в класі **Real** реалізувати метод, що обчислює корінь довільного степеню, і метод для обчислення числа  $\pi$  в заданій степені.

### Варіант 15.

Створити клас **Triad** (трійка чисел); визначити методи збільшення полів на 1.

Визначити похідний клас **Time**. Перевизначити методи збільшення полів на 1 і визначити методи збільшення часу на *n* секунд і хвилин.

### Варіант 16.

Створити клас **Pair** (пара цілих чисел) з операціями перевірки на рівність і множення полів. Реалізувати операцію віднімання пар за формулою

$$(a, b) - (c, d) = (a - c, b - d).$$

Створити похідний клас **Rational**; визначити нові операції:

додавання

$$(a, b) + (c, d) = (ad + bc, bd)$$

і ділення

$$(a, b) / (c, d) = (ad, bc);$$

перевизначити операцію віднімання

$$(a, b) - (c, d) = (ad - bc, bd).$$

### Варіант 17.

Створити клас **Pair** (пара чисел); визначити метод множення полів та операцію додавання пар

$$(a, b) + (c, d) = (a + c, b + d).$$

Визначити похідний клас **Complex**. Визначити методи множення

$$(a, b) \times (c, d) = (ac - bd, ad + bc)$$

і віднімання

$$(a, b) - (c, d) = (a - c, b - d).$$

### Варіант 18.\*

Створити клас **Pair** (пара цілих чисел); визначити методи зміни полів і операцію додавання пар

$$(a, b) + (c, d) = (a + c, b + d).$$

Визначити похідний клас **Long**. Перевизначити операцію додавання і визначити методи множення і віднімання.

### Варіант 19.

Створити клас **Triad** (трійка чисел) з операціями: додавання числа, множення на число, перевірки на рівність.

Створити похідний клас **Vector3D**. Мають бути реалізовані: операція додавання векторів, скалярний добуток векторів.

### Варіант 20.

Створити клас **Pair** (пара цілих чисел); визначити метод множення на число і операцію додавання пар

$$(a, b) + (c, d) = (a + c, b + d).$$

Визначити похідний клас **Money**. Перевизначити операцію додавання і визначити методи віднімання і ділення грошових сум.

### Варіант 21.

Створити клас **Car** (машина), що характеризується торговою маркою (літерний рядок), кількістю циліндрів, потужністю. Визначити методи пере-присвоєння та зміни потужності.

Створити похідний клас **Bus** (автобус), що містить поле «кількість пасажирських місць». Визначити функції пере-присвоєння марки та зміни кількості пасажирських місць.

### Варіант 22.

Створити клас **Pair** (пара чисел); визначити методи зміни полів і порівняння пар: пара  $p1$  більше пари  $p2$ , якщо

$(p1.first > p2.first)$  або  $(p1.first = p2.first)$  і  $(p1.second > p2.second)$ .

Визначити похідний клас **Rational** (дріб). Визначити повний набір методів порівняння.

### Варіант 23.

Створити клас **Liquid** (рідина), що має поля «назва» і «густина». Визначити методи перепризначення і зміни густини.

Створити похідний клас **Solution** (розчин), що містить поле «відносна кількість речовини». Визначити методи пере-присвоєння та зміни відносної кількості речовини.

### Варіант 24.

Створити клас **Pair** (пара чисел); визначити методи зміни полів та обчислення добутку чисел.

Визначити похідний клас **Ellipse** (еліпс). Визначити методи обчислення периметру та площі еліпсу.

### Варіант 25.

Створити клас **Man** (людина) з полями: ім'я, вік, стать і вага. Визначити методи пере-присвоєння імені, зміни віку і зміни ваги.

Створити похідний клас **Student**, що має поле «курс». Визначити методи пере-присвоєння та збільшення курсу.

### Варіант 26.

Створити клас **Car** (машина), що характеризується торговою маркою (літерний рядок), кількістю циліндрів, потужністю. Визначити методи пере-присвоєння та зміни



потужності.

Створити похідний клас **Lorry** (вантажівка), що характеризується також вантажопідйомністю кузова. Визначити функції пере-присвоєння марки та зміни вантажопідйомності.

### **Варіант 27.**

Створити клас **Pair** (пара чисел); визначити методи зміни полів і порівняння пар: пара **p1** більше пари **p2**, якщо  $(p1.first > p2.first)$  або  $(p1.first = p2.first)$  і  $(p1.second > p2.second)$ .

Визначити похідний клас **Fraction**. Визначити повний набір методів порівняння.

### **Варіант 28.**

Створити клас **Liquid** (рідина), що має поля «назва» і «густина». Визначити методи перепризначення і зміни густини.

Створити похідний клас **Alcohol** (спирт), що містить поле «міцність». Визначити методи пере-присвоєння та зміни міцності.

### **Варіант 29.**

Створити клас **Pair** (пара чисел); визначити методи зміни полів та обчислення добутку чисел.

Визначити похідний клас **Rectangle** (прямокутник). Визначити методи обчислення периметру та площі прямокутника.

### **Варіант 30.**

Створити клас **Man** (людина) з полями: ім'я, вік, стать і вага. Визначити методи пере-присвоєння імені, зміни віку і зміни ваги.

Створити похідний клас **Student**, що має поле «рік навчання». Визначити методи пере-присвоєння та збільшення року навчання.

### **Варіант 31.**

Створити клас **Triad** (трійка чисел); визначити методи зміни полів і обчислення суми чисел.

Визначити похідний клас **Triangle** (трикутник). Визначити методи обчислення кутів і площі трикутника.

### Варіант 32.

Створити клас **Triangle** (трикутник) з полями-сторонами. Визначити методи зміни сторін, обчислення кутів, обчислення периметру.

Створити похідний клас **Equilateral** (рівносторонній трикутник), що має поле «площа». Визначити метод обчислення площі.

### Варіант 33.

Створити клас **Triangle** (трикутник) з полями-сторонами. Визначити методи зміни сторін, обчислення кутів, обчислення периметру.

Створити похідний клас **RightAngled** (прямокутний трикутник), що має поле «площа». Визначити метод обчислення площі.

### Варіант 34.

Створити клас **Pair** (пара чисел); визначити методи зміни полів і обчислення добутку чисел.

Визначити похідний клас **RightAngled** (прямокутний трикутник). Визначити методи обчислення гіпотенузи і площі трикутника.

### Варіант 35.

Створити клас **Triad** (трійка чисел); визначити метод порівняння тріад:  
тріада **t1** більше тріади **t2**, якщо

$(t1.first > t2.first)$  або  $(t1.first = t2.first) \wedge (t1.second > t2.second)$   
або  $(t1.first = t2.first) \wedge (t1.second = t2.second) \wedge (t1.third > t2.third)$ .

Визначити похідний клас **Date**. Визначити повний набір методів порівняння дат.

### Варіант 36.

Створити клас **Triad** (трійка чисел); визначити метод порівняння тріад:  
тріада **t1** більше тріади **t2**, якщо

$(t1.first > t2.first)$  або  $(t1.first = t2.first) \wedge (t1.second > t2.second)$   
або  $(t1.first = t2.first) \wedge (t1.second = t2.second) \wedge (t1.third > t2.third)$ .

Визначити похідний клас **Time**. Визначити повний набір методів порівняння моментів часу.

### Варіант 37.

Реалізувати клас-оболонку **Number** для числового типу **float**. Реалізувати методи

додавання і ділення.

Створити похідний клас **Real**, в класі **Real** реалізувати метод піднесення до довільного степеню, та метод для обчислення логарифму числа.

### **Варіант 38.**

Створити клас **Triad** (трійка чисел); визначити методи збільшення полів на 1.

Визначити похідний клас **Date**. Перевизначити методи збільшення полів на 1 і визначити метод збільшення дати на  $n$  днів.

### **Варіант 39.**

Реалізувати клас-оболонку **Number** для числового типу **double**. Реалізувати методи множення і віднімання.

Створити похідний клас **Real**, в класі **Real** реалізувати метод, що обчислює корінь довільного степеню, і метод для обчислення числа  $\pi$  в заданій степені.

### **Варіант 40.**

Створити клас **Triad** (трійка чисел); визначити методи збільшення полів на 1.

Визначити похідний клас **Time**. Перевизначити методи збільшення полів на 1 і визначити методи збільшення часу на  $n$  секунд і хвилин.

## Завдання Е

Реалізувати завдання свого варіанту Лабораторної роботи № 1.5 «Композиція класів та об'єктів» з конструкторами та перевантаженням операцій як класи-нащадки від класів із завдання Лабораторної роботи № 1.5 (агрегований клас слід використовувати як базовий, а клас-контейнер стане похідним класом).

Оскільки слід реалізувати конструктори та операції, то зручніше за основу брати завдання свого варіанту Лабораторної роботи № 2.5 «Конструктори та перевантаження операцій для класів з композицією».

При цьому допоміжний клас слід реалізувати як відкритий клас-нащадок від базового класу `Object`. Клас `Object` реалізує лічильник кількості створених об'єктів.

### Лабораторна робота № 2.5:

В кожній лабораторній роботі цієї теми потрібно реалізувати в тому або іншому вигляді визначення нового класу. У всіх завданнях необхідно реалізувати:

- конструктор ініціалізації (один або декілька),
- конструктор без аргументів і
- конструктор копіювання.

Вказані в завданні операції реалізуються за допомогою перевантаження підходящих операцій. У всіх завданнях обов'язково мають бути реалізовані відповідні операції:

- присвоєння,
- введення з клавіатури,
- виводу на екран,
- приведення типу – перетворення у літерний рядок.

Також треба реалізувати операції

- інкременту в обох формах (префіксній та постфіксній) і
- декременту в обох формах (префіксній та постфіксній), – для числових полів (наприклад: так, як вказано у варіантах завдань Лабораторної роботи № 2.3).

Перевантаження операцій виконується таким чином: підходящі операції реалізуються як методи класу, а інші – як зовнішні дружні функції.

Для демонстрації роботи з об'єктами нового типу у всіх завданнях потрібно написати головну функцію. У програмі обов'язково мають бути продемонстровані різні способи створення об'єктів і масивів об'єктів. Програма має демонструвати використання всіх функцій і методів. Вона має виводити на екран розмір класу в режимі `#pragma pack(1)` і без нього.

Визначення класів та реалізації методів слід розмістити в окремих модулях.

**Завдання наступне:**

Виконати завдання свого варіанту Лабораторної роботи № 1.5 (Композиція класів та об'єктів) з конструкторами і перевантаженням операцій.

**Лабораторна робота № 1.5:**

У всіх завданнях потрібно реалізувати по два-три класи. Один клас є «контейнером», всі решту – описують поля, які містяться в «контейнері». Класи, що описують поля класу-«контейнера», мають бути визначені як незалежні.

Визначення класів та реалізації методів слід розмістити в окремих модулях.

Варіанти завдань наступні:

**Варіант 1.**

Створити клас **Car** (машина), що характеризується торговою маркою (літерний рядок), кількістю циліндрів, потужністю. Визначити методи пере-присвоєння та зміни потужності.

Створити похідний клас **Lorry** (вантажівка), що характеризується також вантажопідйомністю кузова. Визначити функції пере-присвоєння марки та зміни вантажопідйомності.

**Варіант 2.**

Створити клас **Pair** (пара чисел); визначити методи зміни полів і порівняння пар: пара  $p1$  більше пари  $p2$ , якщо  $(p1.first > p2.first)$  або  $(p1.first = p2.first)$  і  $(p1.second > p2.second)$ .

Визначити похідний клас **Fraction**. Визначити повний набір методів порівняння.

**Варіант 3.**

Створити клас **Liquid** (рідина), що має поля «назва» і «густина». Визначити методи перепризначення і зміни густини.

Створити похідний клас **Alcohol** (спирт), що містить поле «міцність». Визначити методи пере-присвоєння та зміни міцності.

**Варіант 4.**

Створити клас **Pair** (пара чисел); визначити методи зміни полів та обчислення добутку чисел.

Визначити похідний клас **Rectangle** (прямокутник). Визначити методи обчислення

периметру та площі прямокутника.

#### **Варіант 5.**

Створити клас **Man** (людина) з полями: ім'я, вік, стать і вага. Визначити методи пере-присвоєння імені, зміни віку і зміни ваги.

Створити похідний клас **Student**, що має поле «рік навчання». Визначити методи пере-присвоєння та збільшення року навчання.

#### **Варіант 6.**

Створити клас **Triad** (трійка чисел); визначити методи зміни полів і обчислення суми чисел.

Визначити похідний клас **Triangle** (трикутник). Визначити методи обчислення кутів і площі трикутника.

#### **Варіант 7.**

Створити клас **Triangle** (трикутник) з полями-сторонами. Визначити методи зміни сторін, обчислення кутів, обчислення периметру.

Створити похідний клас **Equilateral** (рівносторонній трикутник), що має поле «площа». Визначити метод обчислення площі.

#### **Варіант 8.**

Створити клас **Triangle** (трикутник) з полями-сторонами. Визначити методи зміни сторін, обчислення кутів, обчислення периметру.

Створити похідний клас **RightAngled** (прямокутний трикутник), що має поле «площа». Визначити метод обчислення площі.

#### **Варіант 9.**

Створити клас **Pair** (пара чисел); визначити методи зміни полів і обчислення добутку чисел.

Визначити похідний клас **RightAngled** (прямокутний трикутник). Визначити методи обчислення гіпотенузи і площі трикутника.

#### **Варіант 10.**

Створити клас **Triad** (трійка чисел); визначити метод порівняння тріад:  
тріада t1 більше тріади t2, якщо

$(t1.first > t2.first)$  або  $(t1.first = t2.first)$  і  $(t1.second > t2.second)$

або  $(t1.first = t2.first) \text{ і } (t1.second = t2.second) \text{ і } (t1.third > t2.third)$ .

Визначити похідний клас **Date**. Визначити повний набір методів порівняння дат.

### Варіант 11.

Створити клас **Triad** (трійка чисел); визначити метод порівняння тріад:  
тріада **t1** більше тріади **t2**, якщо

$(t1.first > t2.first)$  або  $(t1.first = t2.first) \text{ і } (t1.second > t2.second)$

або  $(t1.first = t2.first) \text{ і } (t1.second = t2.second) \text{ і } (t1.third > t2.third)$ .

Визначити похідний клас **Time**. Визначити повний набір методів порівняння моментів часу.

### Варіант 12.

Реалізувати клас-оболонку **Number** для числового типу **float**. Реалізувати методи додавання і ділення.

Створити похідний клас **Real**, в класі **Real** реалізувати метод піднесення до довільного степеню, та метод для обчислення логарифму числа.

### Варіант 13.

Створити клас **Triad** (трійка чисел); визначити методи збільшення полів на 1.

Визначити похідний клас **Date**. Перевизначити методи збільшення полів на 1 і визначити метод збільшення дати на  $n$  днів.

### Варіант 14.

Реалізувати клас-оболонку **Number** для числового типу **double**. Реалізувати методи множення і віднімання.

Створити похідний клас **Real**, в класі **Real** реалізувати метод, що обчислює корінь довільного степеню, і метод для обчислення числа  $\pi$  в заданій степені.

### Варіант 15.

Створити клас **Triad** (трійка чисел); визначити методи збільшення полів на 1.

Визначити похідний клас **Time**. Перевизначити методи збільшення полів на 1 і визначити методи збільшення часу на  $n$  секунд і хвилин.

### Варіант 16.

Створити клас **Pair** (пара цілих чисел) з операціями перевірки на рівність і множення полів. Реалізувати операцію віднімання пар за формулою

$$(a, b) - (c, d) = (a - c, b - d).$$

Створити похідний клас **Rational**; визначити нові операції:  
додавання

$$(a, b) + (c, d) = (ad + bc, bd)$$

і ділення

$$(a, b) / (c, d) = (ad, bc);$$

перевизначити операцію віднімання

$$(a, b) - (c, d) = (ad - bc, bd).$$

### Варіант 17.

Створити клас **Pair** (пара чисел); визначити метод множення полів та операцію додавання пар

$$(a, b) + (c, d) = (a + c, b + d).$$

Визначити похідний клас **Complex**. Визначити методи множення

$$(a, b) \times (c, d) = (ac - bd, ad + bc)$$

і віднімання

$$(a, b) - (c, d) = (a - c, b - d).$$

### Варіант 18.\*

Створити клас **Pair** (пара цілих чисел); визначити методи зміни полів і операцію додавання пар

$$(a, b) + (c, d) = (a + c, b + d).$$

Визначити похідний клас **Long**. Перевизначити операцію додавання і визначити методи множення і віднімання.

### Варіант 19.

Створити клас **Triad** (трійка чисел) з операціями: додавання числа, множення на число, перевірки на рівність.

Створити похідний клас **Vector3D**. Мають бути реалізовані: операція додавання векторів, скалярний добуток векторів.

### Варіант 20.

Створити клас **Pair** (пара цілих чисел); визначити метод множення на число і операцію додавання пар

$$(a, b) + (c, d) = (a + c, b + d).$$

Визначити похідний клас **Money**. Перевизначити операцію додавання і визначити



методи віднімання і ділення грошових сум.

### **Варіант 21.**

Створити клас **Car** (машина), що характеризується торговою маркою (літерний рядок), кількістю циліндрів, потужністю. Визначити методи пере-присвоєння та зміни потужності.

Створити похідний клас **Bus** (автобус), що містить поле «кількість пасажирських місць». Визначити функції пере-присвоєння марки та зміни кількості пасажирських місць.

### **Варіант 22.**

Створити клас **Pair** (пара чисел); визначити методи зміни полів і порівняння пар: пара  $p_1$  більше пари  $p_2$ , якщо

$(p_1.first > p_2.first)$  або  $(p_1.first = p_2.first)$  і  $(p_1.second > p_2.second)$ .

Визначити похідний клас **Rational** (дріб). Визначити повний набір методів порівняння.

### **Варіант 23.**

Створити клас **Liquid** (рідина), що має поля «назва» і «густина». Визначити методи перепризначення і зміни густини.

Створити похідний клас **Solution** (розчин), що містить поле «відносна кількість речовини». Визначити методи пере-присвоєння та зміни відносної кількості речовини.

### **Варіант 24.**

Створити клас **Pair** (пара чисел); визначити методи зміни полів та обчислення добутку чисел.

Визначити похідний клас **Ellipse** (еліпс). Визначити методи обчислення периметру та площі еліпсу.

### **Варіант 25.**

Створити клас **Man** (людина) з полями: ім'я, вік, стать і вага. Визначити методи пере-присвоєння імені, зміни віку і зміни ваги.

Створити похідний клас **Student**, що має поле «курс». Визначити методи пере-присвоєння та збільшення курсу.

### **Варіант 26.**

Створити клас **Car** (машина), що характеризується торговою маркою (літерний

рядок), кількістю циліндрів, потужністю. Визначити методи пере-присвоєння та зміни потужності.

Створити похідний клас **Lorry** (вантажівка), що характеризується також вантажопідйомністю кузова. Визначити функції пере-присвоєння марки та зміни вантажопідйомності.

### **Варіант 27.**

Створити клас **Pair** (пара чисел); визначити методи зміни полів і порівняння пар: пара  $p1$  більше пари  $p2$ , якщо

$(p1.first > p2.first)$  або  $(p1.first = p2.first)$  і  $(p1.second > p2.second)$ .

Визначити похідний клас **Fraction**. Визначити повний набір методів порівняння.

### **Варіант 28.**

Створити клас **Liquid** (рідина), що має поля «назва» і «густина». Визначити методи перепризначення і зміни густини.

Створити похідний клас **Alcohol** (спирт), що містить поле «міцність». Визначити методи пере-присвоєння та зміни міцності.

### **Варіант 29.**

Створити клас **Pair** (пара чисел); визначити методи зміни полів та обчислення добутку чисел.

Визначити похідний клас **Rectangle** (прямокутник). Визначити методи обчислення периметру та площі прямокутника.

### **Варіант 30.**

Створити клас **Man** (людина) з полями: ім'я, вік, стать і вага. Визначити методи пере-присвоєння імені, зміни віку і зміни ваги.

Створити похідний клас **Student**, що має поле «рік навчання». Визначити методи пере-присвоєння та збільшення року навчання.

### **Варіант 31.**

Створити клас **Triad** (трійка чисел); визначити методи зміни полів і обчислення суми чисел.

Визначити похідний клас **Triangle** (трикутник). Визначити методи обчислення кутів і площі трикутника.

### Варіант 32.

Створити клас **Triangle** (трикутник) з полями-сторонами. Визначити методи зміни сторін, обчислення кутів, обчислення периметру.

Створити похідний клас **Equilateral** (рівносторонній трикутник), що має поле «площа». Визначити метод обчислення площі.

### Варіант 33.

Створити клас **Triangle** (трикутник) з полями-сторонами. Визначити методи зміни сторін, обчислення кутів, обчислення периметру.

Створити похідний клас **RightAngled** (прямокутний трикутник), що має поле «площа». Визначити метод обчислення площі.

### Варіант 34.

Створити клас **Pair** (пара чисел); визначити методи зміни полів і обчислення добутку чисел.

Визначити похідний клас **RightAngled** (прямокутний трикутник). Визначити методи обчислення гіпотенузи і площі трикутника.

### Варіант 35.

Створити клас **Triad** (трійка чисел); визначити метод порівняння тріад:  
тріада **t1** більше тріади **t2**, якщо

$(t1.first > t2.first)$  або  $(t1.first = t2.first) \text{ і } (t1.second > t2.second)$   
або  $(t1.first = t2.first) \text{ і } (t1.second = t2.second) \text{ і } (t1.third > t2.third)$ .

Визначити похідний клас **Date**. Визначити повний набір методів порівняння дат.

### Варіант 36.

Створити клас **Triad** (трійка чисел); визначити метод порівняння тріад:  
тріада **t1** більше тріади **t2**, якщо

$(t1.first > t2.first)$  або  $(t1.first = t2.first) \text{ і } (t1.second > t2.second)$   
або  $(t1.first = t2.first) \text{ і } (t1.second = t2.second) \text{ і } (t1.third > t2.third)$ .

Визначити похідний клас **Time**. Визначити повний набір методів порівняння моментів часу.

### Варіант 37.

Реалізувати клас-оболонку **Number** для числового типу **float**. Реалізувати методи

додавання і ділення.

Створити похідний клас **Real**, в класі **Real** реалізувати метод піднесення до довільного степеню, та метод для обчислення логарифму числа.

#### **Варіант 38.**

Створити клас **Triad** (трійка чисел); визначити методи збільшення полів на 1.

Визначити похідний клас **Date**. Перевизначити методи збільшення полів на 1 і визначити метод збільшення дати на  $n$  днів.

#### **Варіант 39.**

Реалізувати клас-оболонку **Number** для числового типу **double**. Реалізувати методи множення і віднімання.

Створити похідний клас **Real**, в класі **Real** реалізувати метод, що обчислює корінь довільного степеню, і метод для обчислення числа  $\pi$  в заданій степені.

#### **Варіант 40.**

Створити клас **Triad** (трійка чисел); визначити методи збільшення полів на 1.

Визначити похідний клас **Time**. Перевизначити методи збільшення полів на 1 і визначити методи збільшення часу на  $n$  секунд і хвилин.

## Завдання F

Реалізувати основні класи із завдання свого варіанту Лабораторної роботи № 1.7 «Композиція класів та об'єктів» з конструкторами та перевантаженням операцій як класи-нащадки від класів із завдання Лабораторної роботи № 1.7 (агрегований клас слід використовувати як базовий, а клас-контейнер стане похідним класом).

Оскільки слід реалізувати конструктори та операції, то зручніше за основу брати завдання свого варіанту Лабораторної роботи № 2.7 «Конструктори та перевантаження операцій для класів з композицією – складніші завдання».

Реалізувати два види успадкування – відкрите та закрите.

### Лабораторна робота № 2.7:

В кожній лабораторній роботі цієї теми потрібно реалізувати в тому або іншому вигляді визначення нового класу. У всіх завданнях необхідно реалізувати:

- конструктор ініціалізації (один або декілька),
- конструктор без аргументів і
- конструктор копіювання.

Вказані в завданні операції реалізуються за допомогою перевантаження підходящих операцій. У всіх завданнях обов'язково мають бути реалізовані відповідні операції:

- присвоєння,
- введення з клавіатури,
- виводу на екран,
- приведення типу – перетворення у літерний рядок.

Також треба реалізувати операції

- інкременту в обох формах (префіксній та постфіксній) і
- декременту в обох формах (префіксній та постфіксній), – для числових полів (наприклад: так, як вказано у варіантах завдань Лабораторної роботи № 2.3).

Перевантаження операцій виконується таким чином: підходящі операції реалізуються як методи класу, а інші – як зовнішні дружні функції.

Для демонстрації роботи з об'єктами нового типу у всіх завданнях потрібно написати головну функцію. У програмі обов'язково мають бути продемонстровані різні способи створення об'єктів і масивів об'єктів. Програма має демонструвати використання всіх функцій і методів. Вона має виводити на екран розмір класу в режимі `#pragma pack(1)` і без нього.

Визначення класів та реалізації методів слід розмістити в окремих модулях.

### **Завдання наступне:**

Виконати завдання свого варіанту Лабораторної роботи № 1.7 (Композиція класів та об'єктів) з конструкторами і перевантаженням операцій.

### **Лабораторна робота № 1.7:**

У всіх завданнях потрібно реалізувати по два-три класи. Один клас є основним, всі решту – допоміжні. Допоміжні класи мають бути визначені як незалежні. Об'єкти допоміжних класів мають використовуватися як поля основного класу.

Визначення класів та реалізації методів слід розмістити в окремих модулях.

Варіанти завдань наступні:

#### **Варіант 1.**

Реалізувати клас **Account**:

Клас **Account**, – банківський рахунок. У класі мають бути чотири поля:

- прізвище власника,
- номер рахунку,
- відсоток нарахування і
- сума в гривнях.

Відкриття нового рахунку виконується операцією ініціалізації. Необхідно виконувати наступні операції:

- змінити власника рахунку,
- зняти деяку суму грошей з рахунку,
- покласти гроші на рахунок,
- нарахувати відсотки,
- перевести суму в долари,
- перевести суму в євро,
- отримати суму прописом (число перетворити у літерний рядок, наприклад 1992,28 → «одна тисяча дев'ятсот дев'яносто дві грн. 28 коп.»).

Для представлення суми використовувати клас **Money**:

Клас **Money** – для роботи з грошовими сумами. Сума має бути представлене двома полями:

- типу **long** для гривень і
- типу **unsigned char** – для копійок.

Дробова частина (копійки) при виводі на екран має бути відокремлена від цілої

частини комою. Реалізувати операції:

- додавання,
- віднімання,
- ділення сум,
- ділення суми на дробове число,
- множення на дробове число,
- операції порівняння.

## Варіант 2.

Реалізувати клас **Account**:

Клас **Account**, – банківський рахунок. У класі мають бути чотири поля:

- прізвище власника,
- номер рахунку,
- відсоток нарахування і
- сума в гривнях.

Відкриття нового рахунку виконується операцією ініціалізації. Необхідно виконувати наступні операції:

- змінити власника рахунку,
- зняти деяку суму грошей з рахунку,
- покласти гроші на рахунок,
- нарахувати відсотки,
- перевести суму в долари,
- перевести суму в євро,
- отримати суму прописом (число перетворити у літерний рядок, наприклад 1992,28 → «одна тисяча дев'яност дев'яносто дві грн. 28 коп.»).

Для представлення суми використовувати клас **Money**:

Номінали гривень можуть приймати значення 1, 2, 5, 10, 20, 50, 100, 200, 500. Копійки представити як 0.01 (1 копійка), 0.02 (2 копійки), 0.05 (5 копійок), 0.1 (10 копійок), 0.25 (25 копійок), 0.5 (50 копійок). Створити клас **Money** для роботи з грошовими сумами. Сума має бути представлена полями-номіналами, значеннями яких має бути кількість купюр відповідного номіналу. Поля:

- кількість банкнот по 500 грн.
- кількість банкнот по 200 грн.

- кількість банкнот по 100 грн.
- кількість банкнот по 50 грн.
- кількість банкнот по 20 грн.
- кількість банкнот по 10 грн.
- кількість банкнот по 5 грн.
- кількість банкнот по 2 грн.
- кількість банкнот по 1 грн.
- кількість монет по 50 коп.
- кількість монет по 25 коп.
- кількість монет по 10 коп.
- кількість монет по 5 коп.
- кількість монет по 2 коп.
- кількість монет по 1 коп.

Реалізувати:

- додавання сум,
- віднімання сум,
- ділення сум,
- ділення суми на дробове число,
- множення на дробове число,
- операції порівняння.

Дробова частина (копійки) при виводі на екран має бути відокремлена від цілої частини комою.

### Варіант 3.

Реалізувати клас `Calculator` з повним набором арифметичних операцій, використовуючи клас `Fraction`:

Клас `Fraction` – для роботи з дробовими числами. Число має бути представлене двома полями:

- ціла частина – довге ціле із знаком,
- дробова частина – без-знакове коротке ціле.

Реалізувати арифметичні операції:

- додавання,
- віднімання,
- множення,



- операції порівняння.

#### Варіант 4.

Реалізувати клас **Bankomat**:

Клас **Bankomat**, – моделює роботу банкомату. У класі мають міститися поля для зберігання:

- ідентифікаційного номера банкомату,
- інформації про поточну суму грошей, що залишилася у банкоматі,
- мінімальній і
- максимальній сумах, які дозволяється зняти клієнтові в один день.

Реалізувати:

- метод ініціалізації банкомату,
- метод завантаження купюр в банкомат,
- метод зняття певної суми грошей.

Метод зняття грошей має виконувати перевірку на коректність суми, що знімається: вона не може бути менше мінімального значення і не може перевищувати максимальне значення.

- метод `toString()` має перетворити у літерний рядок суму грошей, що залишилася в банкоматі.

Для представлення суми використовувати клас **Money**:

Номінали гривень можуть приймати значення 1, 2, 5, 10, 20, 50, 100, 200, 500. Копійки представити як 0.01 (1 копійка), 0.02 (2 копійки), 0.05 (5 копійок), 0.1 (10 копійок), 0.25 (25 копійок), 0.5 (50 копійок). Створити клас **Money** для роботи з грошовими сумами. Сума має бути представлена полями-номіналами, значеннями яких має бути кількість купюр відповідного номіналу. Поля:

- кількість банкнот по 500 грн.
- кількість банкнот по 200 грн.
- кількість банкнот по 100 грн.
- кількість банкнот по 50 грн.
- кількість банкнот по 20 грн.
- кількість банкнот по 10 грн.
- кількість банкнот по 5 грн.
- кількість банкнот по 2 грн.

- кількість банкнот по 1 грн.
- кількість монет по 50 коп.
- кількість монет по 25 коп.
- кількість монет по 10 коп.
- кількість монет по 5 коп.
- кількість монет по 2 коп.
- кількість монет по 1 коп.

Реалізувати:

- додавання сум,
- віднімання сум,
- ділення сум,
- ділення суми на дробове число,
- множення на дробове число,
- операції порівняння.

Дробова частина (копійки) при виводі на екран має бути відокремлена від цілої частини комою.

### Варіант 5\*.

Реалізувати клас `Fraction`:

Клас `Fraction` – для роботи з дробовими числами. Число має бути представлене двома полями:

- ціла частина – класу `DigitString`,
- дробова частина – без-знакове коротке ціле.

Реалізувати арифметичні операції:

- \* додавання,
- \* віднімання,
- \* множення,
- операції порівняння.

Для представлення цілої частини використовувати клас `DigitString`, а для представлення дробової частини без-знакове коротке ціле.

Клас `DigitString` – для роботи з цілими числами. Число має бути представлене символами-цифрами, які утворюють літерний рядок.

Мають бути реалізовані:

- \* всі арифметичні операції, присутні в C++ (без присвоєння), та
- операції порівняння.

### Варіант 6\*.

Реалізувати клас **Calculator** з повним набором арифметичних операцій, на основі класу **Fraction**:

Клас **Fraction** – для роботи з дробовими числами. Число має бути представлене двома полями:

- ціла частина – довге ціле із знаком,
- дробова частина – без-знакове коротке ціле.

Реалізувати арифметичні операції:

- \* додавання,
- \* віднімання,
- \* множення,
- операції порівняння.

Для представлення цілої частини використовувати клас **LongLong**, а для представлення дробової частини додатне дробове число типу **double**:

Клас **LongLong** – для роботи з 64 бітовими цілими числами. Число має бути представлене двома полями:

- типу **long** – старша частина,
- типу **unsigned long** – молодша частина.

Мають бути реалізовані:

- \* всі арифметичні операції, присутні в C++ (без присвоєння), та
- операції порівняння.

### Варіант 7.

Реалізувати клас **Triangle**:

Клас **Triangle** – для представлення трикутника. Поля даних:

- $a$ ,
- $b$ ,
- $c$  – сторони;
- $A$ ,
- $B$ ,
- $C$  – протилежні кути.

– мають включати кути і сторони. Потрібно реалізувати операції:

- отримання полів даних,
- зміни полів даних,
- обчислення площі,
- обчислення периметру,
- обчислення висот,
- визначення виду трикутника (рівносторонній, рівнобедрений або прямокутний).

Для представлення кутів використовувати клас **Angle**:

Клас **Angle** – для роботи з кутами на площині, що задаються величиною в градусах і хвилинах. Поля:

- *grades*
- *minutes*

Обов'язково мають бути реалізовані:

- переведення в радіани,
- приведення до діапазону  $0^\circ - 360^\circ$ ,
- збільшення кута на задану величину,
- зменшення кута на задану величину,
- отримання синуса,
- порівняння кутів.

## Варіант 8.

Реалізувати клас **Goods**:

Клас **Goods** – товари. У класі мають бути представлені поля:

- найменування товару,
- дата оформлення,
- ціна товару,
- кількість одиниць товару,
- номер накладної, по якій товар поступив на склад.

Реалізувати методи:

- зміни ціни товару,
- зміни кількості товару (збільшення і зменшення),
- обчислення вартості товару.
- Метод `toString()` має повертати у вигляді літерного рядка вартість товару.

Для представлення ціни використовувати клас **Money**:

Клас **Money** – для роботи з грошовими сумами. Число має бути представлене двома полями:

- типу **long** для гривень і
- типу **unsigned char** – для копійок.

Дробова частина (копійки) при виводі на екран має бути відокремлена від цілої частини комою. Реалізувати операції:

- додавання,
- віднімання,
- ділення сум,
- ділення суми на дробове число,
- множення на дробове число,
- операції порівняння.

Реалізувати метод уцінки товару, зменшуючи ціну на 1% за кожен день прострочення терміну придатності.

## Варіант 9.

Реалізувати клас **Triangle** з полями – координатами вершин:

Клас **Triangle** – для представлення трикутника. Поля даних:

- $P_1$ ,
- $P_2$ ,
- $P_3$  – точки (вершини трикутника),

Потрібно реалізувати операції:

- отримання полів даних,
- зміни полів даних,
- обчислення площі,
- обчислення периметру,
- обчислення висот,
- визначення виду трикутника (рівносторонній, рівнобедрений або прямокутний).
- $get\_a()$ ,
- $get\_b()$ ,
- $get\_c()$  – обчислення довжин сторін;
- $get\_A()$ ,

- `get_B( )`,
- `get_C( )` – обчислення величин протилежних кутів.

Для представлення координат вершин використовуйте клас **Point**:

Клас **Point** – для роботи з точками на площині. Координати точки – декартові. Поля:

- `x`
- `y`

Обов'язково мають бути реалізовані:

- переміщення точки по осі X,
- переміщення по осі Y,
- визначення відстані до початку координат,
- відстані між двома точками,
- перетворення у полярні координати,
- порівняння на рівність та нерівність.

## Варіант 10.

Реалізувати клас **Payment**:

Клас **Payment** – зарплата. У класі мають бути представлені поля:

- прізвище-ім'я-побатькові,
- ставка,
- рік поступлення на роботу,
- відсоток надбавки,
- прибутковий податок,
- кількість відпрацьованих днів в місяці,
- кількість робочих днів в місяці,
- нарахована і
- утримана суми.

Реалізувати методи:

- обчислення нарахованої суми,
- обчислення утриманої суми,
- обчислення суми, що видається на руки,
- обчислення стажу.

Стаж обчислюється як повна кількість років, що пройшли від року прийому на роботу, до поточного року. Нарахування є сумою, нарахованою за відпрацьовані дні, і

надбавки, тобто долі від першої суми. Утриманнями є відрахування до пенсійного фонду (1% від нарахованої суми) і прибутковий податок. Прибутковий податок складає 13% від нарахованої суми без відрахувань в пенсійний фонд.

Для представлення полів нарахувань і утримань використовувати клас **Money**:

Клас **Money** для роботи з грошовими сумами. Число має бути представлене двома полями:

- типу **long** для гривень і
- типу **unsigned char** – для копійок.

Дробова частина (копійки) при виводі на екран має бути відокремлена від цілої частини комою. Реалізувати операції:

- додавання,
- віднімання,
- ділення сум,
- ділення суми на дробове число,
- множення на дробове число,
- операції порівняння.

## **Варіант 11.**

Реалізувати клас **Money**:

Клас **Money** – для роботи з грошовими сумами. Число має бути представлене вкладеним об'єктом класу **Fraction**.

Дробова частина (копійки) при виводі на екран має бути відокремлена від цілої частини комою. Реалізувати операції:

- додавання,
- віднімання,
- ділення сум,
- ділення суми на дробове число,
- множення на дробове число,
- операції порівняння.

Для представлення величини грошової суми використовувати клас **Fraction**:

Клас **Fraction** – для роботи з дробовими числами. Число має бути представлене двома полями:

- ціла частина – довге ціле із знаком,

- дробова частина – без-знакове коротке ціле.

Реалізувати арифметичні операції:

- додавання,
- віднімання,
- множення,
- операції порівняння.

## Варіант 12.

Реалізувати клас `ModelWindow`, додавши поле для курсору:

Створити клас `ModelWindow` для роботи з моделями екранних вікон. В якості полів задаються:

- заголовок вікна,
- координати лівого верхнього кута,
- розмір по горизонталі,
- розмір по вертикалі,
- колір вікна,
- стан «видиме / невидиме»,
- стан «з рамкою / без рамки».

Координати і розміри вказуються в цілих числах. Реалізувати операції:

- пересування вікна по горизонталі,
- пересування вікна по вертикалі;
- зміна висоти і/або ширини вікна;
- зміна кольору;
- встановлення стану,
- отримання значення стану.

Операції пересування і зміни розміру мають здійснювати перевірку на перетин меж екрану. Функція виводу на екран має змінювати стан полів об'єкту.

Для представлення поля курсору використовуйте клас `Cursor`:

Клас `Cursor`. Полями є:

- $x$
- $y$  – координати курсору по горизонталі і вертикалі – цілі додатні числа,
- вид курсору – горизонтальний або вертикальний,
- розмір курсору – ціле число від 1 до 15.



Реалізувати методи:

- зміни координат курсору,
- зміни виду курсору,
- зміни розміру курсору,
- метод гасіння і
- метод відновлення курсору.

### Варіант 13\*.

Реалізувати клас `Set` (множина) не більше ніж з 64 елементів цілих чисел, використовуючи клас `BitString`:

Клас `BitString` – для роботи з 64-бітовими рядками. Бітовий рядок має бути представлений двома полями типу `unsigned long`. Мають бути реалізовані всі традиційні операції для роботи з бітами:

- `and`,
- `or`,
- `xor`,
- `not`.
- \* зсув ліворуч `shiftLeft` та
- \* зсув праворуч `shiftRight` на задану кількість бітів.

Клас `Set` (множина) має забезпечувати операції:

- включення елемента в множину,
- виключення елемента з множини,
- об'єднання,
- перетин і
- різницю множин,
- обчислення кількості елементів в множині.

### Варіант 14\*.

Реалізувати клас `Rational`:

Раціональний (нескоротний) дріб представляється парою цілих чисел  $(a, b)$ , де поля:

- $a$  – чисельник,
- $b$  – знаменник.

Клас `Rational` – для роботи з раціональними дробами. Обов'язково мають бути

реалізовані наступні операції:

Припустимо, що  $(a, b)$  – перше число  $= a/b$  – перший об’єкт;  $(c, d)$  – друге число  $= c/d$  – другий об’єкт.

- \* додавання `add()`,  $(a, b) + (c, d) = (ad + bc, bd) = (ad + bc)/bd$ ;
- \* віднімання `sub()`,  $(a, b) - (c, d) = (ad - bc, bd)$ ;
- \* множення `mul()`,  $(a, b) \times (c, d) = (ac, bd)$ ;
- \* ділення `div()`,  $(a, b) / (c, d) = (ad, bc)$ ;
- порівняння `equal()`, `great()`, `less()`.

Має бути реалізована приватна функція скорочення дробу `Reduce()`, яка обов’язково викликається при виконанні арифметичних операцій.

Для представлення чисельника і знаменника використовувати клас `LongLong`:

Клас `LongLong` – для роботи з 64 бітовими цілими числами. Число має бути представлене двома полями:

- типу `long` – старша частина,
- типу `unsigned long` – молодша частина.

Мають бути реалізовані:

- \* всі арифметичні операції, присутні в C++ (без присвоєння), та
- операції порівняння.

## Варіант 15\*.

Реалізувати клас `Money`:

Клас `Money` – для роботи з грошовими сумами. Число має бути представлене двома полями:

- типу `LongLong` для гривень (див. далі) і
- типу `unsigned char` – для копійок.

Дробова частина (копійки) при виводі на екран має бути відокремлена від цілої частини комою. Реалізувати операції:

- \* додавання,
- \* віднімання,
- \* ділення сум,
- \* ділення суми на дробове число,
- \* множення на дробове число,
- операції порівняння.

Для представлення гривень використовувати клас **LongLong**:

Клас **LongLong** – для роботи з 64 бітовими цілими числами. Число має бути представлене двома полями:

- типу **long** – старша частина,
- типу **unsigned long** – молодша частина.

Мають бути реалізовані:

- \* всі арифметичні операції, присутні в C++ (без присвоєння), та
- операції порівняння.

### **Варіант 16\*.**

Реалізувати клас **Cursor**:

Клас **Cursor**. Полями є:

- *x*
- *y* – координати курсору по горизонталі і вертикалі – цілі додатні числа,
- вид курсору – горизонтальний або вертикальний,
- розмір курсору – ціле число від 1 до 15.

Реалізувати методи:

- \* зміни координат курсору,
- зміни виду курсору,
- зміни розміру курсору,
- метод гасіння і
- метод відновлення курсору.

Для представлення координат використовувати клас **LongLong**:

Клас **LongLong** для роботи з 64 бітовими цілими числами. Число має бути представлене двома полями:

- типу **long** – старша частина,
- типу **unsigned long** – молодша частина.

Мають бути реалізовані:

- \* всі арифметичні операції, присутні в C++ (без присвоєння), та
- операції порівняння.

### **Варіант 17.\***

Реалізувати клас **Account**:

Клас **Account**, – банківський рахунок. У класі мають бути поля:

- прізвище власника,
- номер рахунку,
- дата відкриття,
- відсоток нарахування і
- сума в гривнях.

Відкриття нового рахунку виконується операцією ініціалізації. Необхідно виконувати наступні операції:

- змінити власника рахунку,
- зняти деяку суму грошей з рахунку,
- покласти гроші на рахунок,
- нарахувати відсотки,
- перевести суму в долари,
- перевести суму в євро,
- отримати суму прописом (число перетворити у літерний рядок, наприклад 1992,28 → «одна тисяча дев'ятсот дев'яносто дві грн. 28 коп.»).

Додати поле – дату відкриття рахунку, використовуючи клас **Date**:

Клас **Date** – для роботи з датами у форматі «рік.місяць.день» з трьома полями типу **unsigned int**:

- рік,
- місяць і
- номер дня.

Клас має включати не менше трьох функцій ініціалізації:

- числами,
- літерним рядком виду «рік.місяць.день» (наприклад, «2004.08.31») і
- датою.

Обов'язковими операціями є:

- \* обчислення дати через задану кількість днів,
- \* віднімання заданої кількості днів з дати,
- визначення, чи рік – високосний,
- присвоєння,
- отримання окремих частин (рік, місяць, день),
- порівняння дат (рівно, до, після),

- \* обчислення кількості днів між датами.

Додати метод, що обчислює кількість днів, що пройшли з початку відкриття рахунку, і що додає по 0,01 % до відсотку нарахування за кожен день.

### Варіант 18.\*

Реалізувати клас **Goods**, додавши поле – дату надходження товару на склад.

Клас **Goods** – товари. У класі мають бути представлені поля:

- найменування товару,
- дата оформлення,
- ціна товару,
- кількість одиниць товару,
- номер накладної,
- по якій товар поступив на склад.

Реалізувати методи:

- зміни ціни товару,
- зміни кількості товару (збільшення і зменшення),
- обчислення вартості товару.
- Метод `toString()` має повертати у вигляді літерного рядка вартість товару.

Використовувати клас **Date**:

Клас **Date** – для роботи з датами у форматі «рік.місяць.день» з трьома полями типу `unsigned int`:

- рік,
- місяць і
- номер дня.

Клас має включати не менше трьох функцій ініціалізації:

- числами,
- літерним рядком виду «рік.місяць.день» (наприклад, «2004.08.31») і
- датою.

Обов'язковими операціями є:

- \* обчислення дати через задану кількість днів,
- \* віднімання заданої кількості днів з дати,
- визначення, чи рік – високосний,
- присвоєння,

- отримання окремих частин (рік, місяць, день),
- порівняння дат (рівно, до, після),
- \* обчислення кількості днів між датами.

Реалізувати метод, що обчислює термін зберігання товару.

### Варіант 19.\*

Реалізувати клас **Payment**:

Клас **Payment** – зарплата. У класі мають бути представлені поля:

- прізвище-ім'я-побатькові,
- ставка,
- рік поступлення на роботу,
- відсоток надбавки,
- прибутковий податок,
- кількість відпрацьованих днів в місяці,
- кількість робочих днів в місяці,
- нарахована і
- утримана суми.

Реалізувати методи:

- обчислення нарахованої суми,
- обчислення утриманої суми,
- обчислення суми, що видається на руки,
- обчислення стажу.

Стаж обчислюється як повна кількість років, що пройшли від року прийому на роботу, до поточного року. Нарахування є сумою, нарахованою за відпрацьовані дні, і надбавки, тобто долі від першої суми. Утриманнями є відрахування до пенсійного фонду (1% від нарахованої суми) і прибутковий податок. Прибутковий податок складає 13% від нарахованої суми без відрахувань в пенсійний фонд.

Замість поля-року використовувати поле-дату класу **Date**:

Клас **Date** – для роботи з датами у форматі «рік.місяць.день» з трьома полями типу `unsigned int`:

- рік,
- місяць і
- номер дня.

Клас має включати не менше трьох функцій ініціалізації:

- числами,
- літерним рядком виду «рік.місяць.день» (наприклад, «2004.08.31») і
- датою.

Обов'язковими операціями є:

- \* обчислення дати через задану кількість днів,
- \* віднімання заданої кількості днів з дати,
- визначення, чи рік – високосний,
- присвоєння,
- отримання окремих частин (рік, місяць, день),
- порівняння дат (рівно, до, після),
- \* обчислення кількості днів між датами.

Стаж слід обчислювати, використовуючи методи класу `Date`.

## **Варіант 20.\***

Реалізувати клас `Bill`, що є разовим платежем за телефонну розмову. Клас має включати поля:

- прізвище платника,
- номер телефону,
- тариф за хвилину розмови,
- знижка (у відсотках),
- час початку розмови,
- час закінчення розмови,
- сума до оплати.

Для представлення часу використовуйте клас `Time`:

Клас `Time` – для роботи з часом у форматі «година:хвилина:секунда» з трьома полями типу `unsigned int`:

- година,
- хвилина і
- секунда.

Клас має включати не менше чотирьох функцій ініціалізації:

- числами,
- літерним рядком (наприклад, «23:59:59»),

- секундами від початку доби і
- часом.

Реалізувати методи:

- \* обчислення різниці між двома моментами часу в секундах,
- \* додавання часу і заданої кількості секунд,
- \* віднімання з часу заданої кількості секунд,
- порівняння моментів часу,
- переведення в секунди,
- переведення в хвилини (з округленням до цілої хвилини);
- отримання і зміни значень полів. Час розмови, який підлягає оплаті, обчислюється в хвилинах; неповна хвилина вважається за повну;
- метод `toString()` має видавати суму в гривнях.

## Варіант 21.

Реалізувати клас `Account`:

Клас `Account`, – банківський рахунок. У класі мають бути чотири поля:

- прізвище власника,
- номер рахунку,
- відсоток нарахування і
- сума в гривнях.

Відкриття нового рахунку виконується операцією ініціалізації. Необхідно виконувати наступні операції:

- змінити власника рахунку,
- зняти деяку суму грошей з рахунку,
- покласти гроші на рахунок,
- нарахувати відсотки,
- перевести суму в долари,
- перевести суму в євро,
- отримати суму прописом (число перетворити у літерний рядок, наприклад 1992,28 → «одна тисяча дев'ятсот дев'яносто дві грн. 28 коп.»).

Для представлення суми використовувати клас `Money`:

Клас `Money` – для роботи з грошовими сумами. Сума має бути представлене двома полями:



- типу `long` для гривень і
- типу `unsigned char` – для копійок.

Дробова частина (копійки) при виводі на екран має бути відокремлена від цілої частини комою. Реалізувати операції:

- додавання,
- віднімання,
- ділення сум,
- ділення суми на дробове число,
- множення на дробове число,
- операції порівняння.

## Варіант 22.

Реалізувати клас `Account`:

Клас `Account`, – банківський рахунок. У класі мають бути чотири поля:

- прізвище власника,
- номер рахунку,
- відсоток нарахування і
- сума в гривнях.

Відкриття нового рахунку виконується операцією ініціалізації. Необхідно виконувати наступні операції:

- змінити власника рахунку,
- зняти деяку суму грошей з рахунку,
- покласти гроші на рахунок,
- нарахувати відсотки,
- перевести суму в долари,
- перевести суму в євро,
- отримати суму прописом (число перетворити у літерний рядок, наприклад 1992,28 → «одна тисяча дев'ятсот дев'яносто дві грн. 28 коп.»).

Для представлення суми використовувати клас `Money`:

Номінали гривень можуть приймати значення 1, 2, 5, 10, 20, 50, 100, 200, 500. Копійки представити як 0.01 (1 копійка), 0.02 (2 копійки), 0.05 (5 копійок), 0.1 (10 копійок), 0.25 (25 копійок), 0.5 (50 копійок). Створити клас `Money` для роботи з грошовими сумами. Сума має бути представлена полями-номіналами, значеннями яких має бути кількість купюр

відповідного номіналу. Поля:

- кількість банкнот по 500 грн.
- кількість банкнот по 200 грн.
- кількість банкнот по 100 грн.
- кількість банкнот по 50 грн.
- кількість банкнот по 20 грн.
- кількість банкнот по 10 грн.
- кількість банкнот по 5 грн.
- кількість банкнот по 2 грн.
- кількість банкнот по 1 грн.
- кількість монет по 50 коп.
- кількість монет по 25 коп.
- кількість монет по 10 коп.
- кількість монет по 5 коп.
- кількість монет по 2 коп.
- кількість монет по 1 коп.

Реалізувати:

- додавання сум,
- віднімання сум,
- ділення сум,
- ділення суми на дробове число,
- множення на дробове число,
- операції порівняння.

Дробова частина (копійки) при виводі на екран має бути відокремлена від цілої частини комою.

### **Варіант 23.**

Реалізувати клас `Calculator` з повним набором арифметичних операцій, використовуючи клас `Fraction`:

Клас `Fraction` – для роботи з дробовими числами. Число має бути представлене двома полями:

- ціла частина – довге ціле із знаком,
- дробова частина – без-знакове коротке ціле.

Реалізувати арифметичні операції:

- додавання,
- віднімання,
- множення,
- операції порівняння.

## Варіант 24.

Реалізувати клас **Bankomat**:

Клас **Bankomat**, – моделює роботу банкомату. У класі мають міститися поля для зберігання:

- ідентифікаційного номера банкомату,
- інформації про поточну суму грошей, що залишилися у банкоматі,
- мінімальній і
- максимальній сумах, які дозволяється зняти клієнтові в один день.

Реалізувати:

- метод ініціалізації банкомату,
- метод завантаження купюр в банкомат,
- метод зняття певної суми грошей.

Метод зняття грошей має виконувати перевірку на коректність суми, що знімається: вона не може бути менше мінімального значення і не може перевищувати максимальне значення.

- метод `toString()` має перетворити у літерний рядок суму грошей, що залишилися в банкоматі.

Для представлення суми використовувати клас **Money**:

Номінали гривень можуть приймати значення 1, 2, 5, 10, 20, 50, 100, 200, 500. Копійки представити як 0.01 (1 копійка), 0.02 (2 копійки), 0.05 (5 копійок), 0.1 (10 копійок), 0.25 (25 копійок), 0.5 (50 копійок). Створити клас **Money** для роботи з грошовими сумами. Сума має бути представлена полями-номіналами, значеннями яких має бути кількість купюр відповідного номіналу. Поля:

- кількість банкнот по 500 грн.
- кількість банкнот по 200 грн.
- кількість банкнот по 100 грн.
- кількість банкнот по 50 грн.
- кількість банкнот по 20 грн.

- кількість банкнот по 10 грн.
- кількість банкнот по 5 грн.
- кількість банкнот по 2 грн.
- кількість банкнот по 1 грн.
- кількість монет по 50 коп.
- кількість монет по 25 коп.
- кількість монет по 10 коп.
- кількість монет по 5 коп.
- кількість монет по 2 коп.
- кількість монет по 1 коп.

Реалізувати:

- додавання сум,
- віднімання сум,
- ділення сум,
- ділення суми на дробове число,
- множення на дробове число,
- операції порівняння.

Дробова частина (копійки) при виводі на екран має бути відокремлена від цілої частини комою.

## Варіант 25\*.

Реалізувати клас **Fraction**:

Клас **Fraction** – для роботи з дробовими числами. Число має бути представлене двома полями:

- ціла частина – класу **DigitString**,
- дробова частина – без-знакове коротке ціле.

Реалізувати арифметичні операції:

- \* додавання,
- \* віднімання,
- \* множення,
- операції порівняння.

Для представлення цілої частини використовувати клас **DigitString**, а для представлення дробової частини без-знакове коротке ціле.

Клас `DigitString` – для роботи з цілими числами. Число має бути представлене символами-цифрами, які утворюють літерний рядок.

Мають бути реалізовані:

- \* всі арифметичні операції, присутні в C++ (без присвоєння), та
- операції порівняння.

### Варіант 26\*.

Реалізувати клас `Calculator` з повним набором арифметичних операцій, на основі класу `Fraction`:

Клас `Fraction` – для роботи з дробовими числами. Число має бути представлене двома полями:

- ціла частина – довге ціле із знаком,
- дробова частина – без-знакове коротке ціле.

Реалізувати арифметичні операції:

- \* додавання,
- \* віднімання,
- \* множення,
- операції порівняння.

Для представлення цілої частини використовувати клас `LongLong`, а для представлення дробової частини додатне дробове число типу `double`:

Клас `LongLong` – для роботи з 64 бітовими цілими числами. Число має бути представлене двома полями:

- типу `long` – старша частина,
- типу `unsigned long` – молодша частина.

Мають бути реалізовані:

- \* всі арифметичні операції, присутні в C++ (без присвоєння), та
- операції порівняння.

### Варіант 27.

Реалізувати клас `Triangle`:

Клас `Triangle` – для представлення трикутника. Поля даних:

- $a$ ,
- $b$ ,
- $c$  – сторони;

- $A$ ,
- $B$ ,
- $C$  – протилежні кути.

– мають включати кути і сторони. Потрібно реалізувати операції:

- отримання полів даних,
- зміни полів даних,
- обчислення площі,
- обчислення периметру,
- обчислення висот,
- визначення виду трикутника (рівносторонній, рівнобедрений або прямокутний).

Для представлення кутів використовувати клас **Angle**:

Клас **Angle** – для роботи з кутами на площині, що задаються величиною в градусах і хвилинах. Поля:

- *grades*
- *minutes*

Обов'язково мають бути реалізовані:

- переведення в радіани,
- приведення до діапазону  $0^\circ - 360^\circ$ ,
- збільшення кута на задану величину,
- зменшення кута на задану величину,
- отримання синуса,
- порівняння кутів.

## Варіант 28.

Реалізувати клас **Goods**:

Клас **Goods** – товари. У класі мають бути представлені поля:

- найменування товару,
- дата оформлення,
- ціна товару,
- кількість одиниць товару,
- номер накладної, по якій товар поступив на склад.

Реалізувати методи:

- зміни ціни товару,

- зміни кількості товару (збільшення і зменшення),
- обчислення вартості товару.
- Метод `toString()` має повертати у вигляді літерного рядка вартість товару.

Для представлення ціни використовувати клас `Money`:

Клас `Money` – для роботи з грошовими сумами. Число має бути представлене двома полями:

- типу `long` для гривень і
- типу `unsigned char` – для копійок.

Дробова частина (копійки) при виводі на екран має бути відокремлена від цілої частини комою. Реалізувати операції:

- додавання,
- віднімання,
- ділення сум,
- ділення суми на дробове число,
- множення на дробове число,
- операції порівняння.

Реалізувати метод уцінки товару, зменшуючи ціну на 1% за кожен день прострочення терміну придатності.

## Варіант 29.

Реалізувати клас `Triangle` з полями – координатами вершин:

Клас `Triangle` – для представлення трикутника. Поля даних:

- $P_1$ ,
- $P_2$ ,
- $P_3$  – точки (вершини трикутника),

Потрібно реалізувати операції:

- отримання полів даних,
- зміни полів даних,
- обчислення площі,
- обчислення периметру,
- обчислення висот,
- визначення виду трикутника (рівносторонній, рівнобедрений або прямокутний).
- `get_a()`,

- `get_b( )`,
- `get_c( )` – обчислення довжин сторін;
- `get_A( )`,
- `get_B( )`,
- `get_C( )` – обчислення величин протилежних кутів.

Для представлення координат вершин використовуйте клас **Point**:

Клас **Point** – для роботи з точками на площині. Координати точки – декартові. Поля:

- `x`
- `y`

Обов'язково мають бути реалізовані:

- переміщення точки по осі X,
- переміщення по осі Y,
- визначення відстані до початку координат,
- відстані між двома точками,
- перетворення у полярні координати,
- порівняння на рівність та нерівність.

### Варіант 30.

Реалізувати клас **Payment**:

Клас **Payment** – зарплата. У класі мають бути представлені поля:

- прізвище-ім'я-побатькові,
- ставка,
- рік поступлення на роботу,
- відсоток надбавки,
- прибутковий податок,
- кількість відпрацьованих днів в місяці,
- кількість робочих днів в місяці,
- нарахована і
- утримана суми.

Реалізувати методи:

- обчислення нарахованої суми,
- обчислення утриманої суми,
- обчислення суми, що видається на руки,



- обчислення стажу.

Стаж обчислюється як повна кількість років, що пройшли від року прийому на роботу, до поточного року. Нарахування є сумою, нарахованою за відпрацьовані дні, і надбавки, тобто долі від першої суми. Утриманнями є відрахування до пенсійного фонду (1% від нарахованої суми) і прибутковий податок. Прибутковий податок складає 13% від нарахованої суми без відрахувань в пенсійний фонд.

Для представлення полів нарахувань і утримань використовувати клас **Money**:

Клас **Money** для роботи з грошовими сумами. Число має бути представлене двома полями:

- типу **long** для гривень і
- типу **unsigned char** – для копійок.

Дробова частина (копійки) при виводі на екран має бути відокремлена від цілої частини комою. Реалізувати операції:

- додавання,
- віднімання,
- ділення сум,
- ділення суми на дробове число,
- множення на дробове число,
- операції порівняння.

### **Варіант 31.**

Реалізувати клас **Money**:

Клас **Money** – для роботи з грошовими сумами. Число має бути представлене вкладеним об'єктом класу **Fraction**.

Дробова частина (копійки) при виводі на екран має бути відокремлена від цілої частини комою. Реалізувати операції:

- додавання,
- віднімання,
- ділення сум,
- ділення суми на дробове число,
- множення на дробове число,
- операції порівняння.

Для представлення величини грошової суми використовувати клас **Fraction**:

Клас **Fraction** – для роботи з дробовими числами. Число має бути представлене двома полями:

- ціла частина – довге ціле із знаком,
- дробова частина – без-знакове коротке ціле.

Реалізувати арифметичні операції:

- додавання,
- віднімання,
- множення,
- операції порівняння.

## Варіант 32.

Реалізувати клас **ModelWindow**, додавши поле для курсору:

Створити клас **ModelWindow** для роботи з моделями екранних вікон. В якості полів задаються:

- заголовок вікна,
- координати лівого верхнього кута,
- розмір по горизонталі,
- розмір по вертикалі,
- колір вікна,
- стан «видиме / невидиме»,
- стан «з рамкою / без рамки».

Координати і розміри вказуються в цілих числах. Реалізувати операції:

- пересування вікна по горизонталі,
- пересування вікна по вертикалі;
- зміна висоти і/або ширини вікна;
- зміна кольору;
- встановлення стану,
- отримання значення стану.

Операції пересування і зміни розміру мають здійснювати перевірку на перетин меж екрану. Функція виводу на екран має змінювати стан полів об'єкту.

Для представлення поля курсору використовуйте клас **Cursor**:

Клас **Cursor**. Полями є:

- $x$

- $y$  – координати курсору по горизонталі і вертикалі – цілі додатні числа,
- вид курсору – горизонтальний або вертикальний,
- розмір курсору – ціле число від 1 до 15.

Реалізувати методи:

- зміни координат курсору,
- зміни виду курсору,
- зміни розміру курсору,
- метод гасіння і
- метод відновлення курсору.

### Варіант 33\*.

Реалізувати клас **Set** (множина) не більше ніж з 64 елементів цілих чисел, використовуючи клас **BitString**:

Клас **BitString** – для роботи з 64-бітовими рядками. Бітовий рядок має бути представлений двома полями типу **unsigned long**. Мають бути реалізовані всі традиційні операції для роботи з бітами:

- **and**,
- **or**,
- **xor**,
- **not**.
- \* зсув ліворуч **shiftLeft** та
- \* зсув праворуч **shiftRight** на задану кількість бітів.

Клас **Set** (множина) має забезпечувати операції:

- включення елемента в множину,
- виключення елемента з множини,
- об'єднання,
- перетин і
- різницю множин,
- обчислення кількості елементів в множині.

### Варіант 34\*.

Реалізувати клас **Rational**:

Раціональний (нескоротний) дріб представляється парою цілих чисел  $(a, b)$ , де поля:

- $a$  – чисельник,
- $b$  – знаменник.

Клас **Rational** – для роботи з раціональними дробами. Обов’язково мають бути реалізовані наступні операції:

Припустимо, що  $(a, b)$  – перше число  $= a/b$  – перший об’єкт;  $(c, d)$  – друге число  $= c/d$  – другий об’єкт.

- \* додавання **add()**,  $(a, b) + (c, d) = (ad + bc, bd) = (ad + bc)/(bd)$ ;
- \* віднімання **sub()**,  $(a, b) - (c, d) = (ad - bc, bd)$ ;
- \* множення **mul()**,  $(a, b) \times (c, d) = (ac, bd)$ ;
- \* ділення **div()**,  $(a, b) / (c, d) = (ad, bc)$ ;
- порівняння **equal()**, **great()**, **less()**.

Має бути реалізована приватна функція скорочення дробу **Reduce()**, яка обов’язково викликається при виконанні арифметичних операцій.

Для представлення чисельника і знаменника використовувати клас **LongLong**:

Клас **LongLong** – для роботи з 64 бітовими цілими числами. Число має бути представлено двома полями:

- типу **long** – старша частина,
- типу **unsigned long** – молодша частина.

Мають бути реалізовані:

- \* всі арифметичні операції, присутні в C++ (без присвоєння), та
- операції порівняння.

### Варіант 35\*.

Реалізувати клас **Money**:

Клас **Money** – для роботи з грошовими сумами. Число має бути представлено двома полями:

- типу **LongLong** для гривень (див. далі) і
- типу **unsigned char** – для копійок.

Дробова частина (копійки) при виводі на екран має бути відокремлена від цілої частини комою. Реалізувати операції:

- \* додавання,
- \* віднімання,
- \* ділення сум,

- \* ділення суми на дробове число,
- \* множення на дробове число,
- операції порівняння.

Для представлення гривень використовувати клас `LongLong`:

Клас `LongLong` – для роботи з 64 бітовими цілими числами. Число має бути представлене двома полями:

- типу `long` – старша частина,
- типу `unsigned long` – молодша частина.

Мають бути реалізовані:

- \* всі арифметичні операції, присутні в C++ (без присвоєння), та
- операції порівняння.

### **Варіант 36\*.**

Реалізувати клас `Cursor`:

Клас `Cursor`. Полями є:

- $x$
- $y$  – координати курсору по горизонталі і вертикалі – цілі додатні числа,
- вид курсору – горизонтальний або вертикальний,
- розмір курсору – ціле число від 1 до 15.

Реалізувати методи:

- \* зміни координат курсору,
- зміни виду курсору,
- зміни розміру курсору,
- метод гасіння і
- метод відновлення курсору.

Для представлення координат використовувати клас `LongLong`:

Клас `LongLong` для роботи з 64 бітовими цілими числами. Число має бути представлене двома полями:

- типу `long` – старша частина,
- типу `unsigned long` – молодша частина.

Мають бути реалізовані:

- \* всі арифметичні операції, присутні в C++ (без присвоєння), та

- операції порівняння.

### Варіант 37.\*

Реалізувати клас Account:

Клас Account, – банківський рахунок. У класі мають бути поля:

- прізвище власника,
- номер рахунку,
- дата відкриття,
- відсоток нарахування і
- сума в гривнях.

Відкриття нового рахунку виконується операцією ініціалізації. Необхідно виконувати наступні операції:

- змінити власника рахунку,
- зняти деяку суму грошей з рахунку,
- покласти гроші на рахунок,
- нарахувати відсотки,
- перевести суму в долари,
- перевести суму в євро,
- отримати суму прописом (число перетворити у літерний рядок, наприклад 1992,28 → «одна тисяча дев'ятсот дев'яносто дві грн. 28 коп.»).

Додати поле – дату відкриття рахунку, використовуючи клас Date:

Клас Date – для роботи з датами у форматі «рік.місяць.день» з трьома полями типу unsigned int:

- рік,
- місяць і
- номер дня.

Клас має включати не менше трьох функцій ініціалізації:

- числами,
- літерним рядком виду «рік.місяць.день» (наприклад, «2004.08.31») і
- датою.

Обов'язковими операціями є:

- \* обчислення дати через задану кількість днів,
- \* віднімання заданої кількості днів з дати,

- визначення, чи рік – високосний,
- присвоєння,
- отримання окремих частин (рік, місяць, день),
- порівняння дат (рівно, до, після),
- \* обчислення кількості днів між датами.

Додати метод, що обчислює кількість днів, що пройшли з початку відкриття рахунку, і що додає по 0,01 % до відсотку нарахування за кожен день.

### Варіант 38.\*

Реалізувати клас **Goods**, додавши поле – дату надходження товару на склад.

Клас **Goods** – товари. У класі мають бути представлені поля:

- найменування товару,
- дата оформлення,
- ціна товару,
- кількість одиниць товару,
- номер накладної,
- по якій товар поступив на склад.

Реалізувати методи:

- зміни ціни товару,
- зміни кількості товару (збільшення і зменшення),
- обчислення вартості товару.
- Метод `toString()` має повертати у вигляді літерного рядка вартість товару.

Використовувати клас **Date**:

Клас **Date** – для роботи з датами у форматі «рік.місяць.день» з трьома полями типу `unsigned int`:

- рік,
- місяць і
- номер дня.

Клас має включати не менше трьох функцій ініціалізації:

- числами,
- літерним рядком виду «рік.місяць.день» (наприклад, «2004.08.31») і
- датою.

Обов'язковими операціями є:

- \* обчислення дати через задану кількість днів,
- \* віднімання заданої кількості днів з дати,
- визначення, чи рік – високосний,
- присвоєння,
- отримання окремих частин (рік, місяць, день),
- порівняння дат (рівно, до, після),
- \* обчислення кількості днів між датами.

Реалізувати метод, що обчислює термін зберігання товару.

### Варіант 39.\*

Реалізувати клас **Payment**:

Клас **Payment** – зарплата. У класі мають бути представлені поля:

- прізвище-ім'я-побатькові,
- ставка,
- рік поступлення на роботу,
- відсоток надбавки,
- прибутковий податок,
- кількість відпрацьованих днів в місяці,
- кількість робочих днів в місяці,
- нарахована і
- утримана суми.

Реалізувати методи:

- обчислення нарахованої суми,
- обчислення утриманої суми,
- обчислення суми, що видається на руки,
- обчислення стажу.

Стаж обчислюється як повна кількість років, що пройшли від року прийому на роботу, до поточного року. Нарахування є сумою, нарахованою за відпрацьовані дні, і надбавки, тобто долі від першої суми. Утриманнями є відрахування до пенсійного фонду (1% від нарахованої суми) і прибутковий податок. Прибутковий податок складає 13% від нарахованої суми без відрахувань в пенсійний фонд.

Замість поля-року використовувати поле-дату класу **Date**:

Клас **Date** – для роботи з датами у форматі «рік.місяць.день» з трьома полями типу



unsigned int:

- рік,
- місяць і
- номер дня.

Клас має включати не менше трьох функцій ініціалізації:

- числами,
- літерним рядком виду «рік.місяць.день» (наприклад, «2004.08.31») і
- датою.

Обов'язковими операціями є:

- \* обчислення дати через задану кількість днів,
- \* віднімання заданої кількості днів з дати,
- визначення, чи рік – високосний,
- присвоєння,
- отримання окремих частин (рік, місяць, день),
- порівняння дат (рівно, до, після),
- \* обчислення кількості днів між датами.

Стаж слід обчислювати, використовуючи методи класу Date.

#### **Варіант 40.\***

Реалізувати клас Bill, що є разовим платежем за телефонну розмову. Клас має включати поля:

- прізвище платника,
- номер телефону,
- тариф за хвилину розмови,
- знижка (у відсотках),
- час початку розмови,
- час закінчення розмови,
- сума до оплати.

Для представлення часу використовуйте клас Time:

Клас Time – для роботи з часом у форматі «година:хвилина:секунда» з трьома полями типу unsigned int:

- година,
- хвилина і

- секунда.

Клас має включати не менше чотирьох функцій ініціалізації:

- числами,
- літерним рядком (наприклад, «23:59:59»),
- секундами від початку доби і
- часом.

Реалізувати методи:

- \* обчислення різниці між двома моментами часу в секундах,
- \* додавання часу і заданої кількості секунд,
- \* віднімання з часу заданої кількості секунд,
- порівняння моментів часу,
- переведення в секунди,
- переведення в хвилини (з округленням до цілої хвилини);
- отримання і зміни значень полів. Час розмови, який підлягає оплаті, обчислюється в хвилинах; неповна хвилина вважається за повну;
- метод `toString()` має видавати суму в гривнях.

## Завдання G

Реалізувати завдання свого варіанту Лабораторної роботи № 1.7 «Композиція класів та об'єктів» з конструкторами та перевантаженням операцій як класи-нащадки від класів із завдання Лабораторної роботи № 1.7 (агрегований клас слід використовувати як базовий, а клас-контейнер стане похідним класом).

Оскільки слід реалізувати конструктори та операції, то зручніше за основу брати завдання свого варіанту Лабораторної роботи № 2.7 «Конструктори та перевантаження операцій для класів з композицією – складніші завдання».

При цьому допоміжний клас слід реалізувати як відкритий клас-нащадок від базового класу Object. Клас Object реалізує лічильник кількості створених об'єктів.

### Лабораторна робота № 2.7:

В кожній лабораторній роботі цієї теми потрібно реалізувати в тому або іншому вигляді визначення нового класу. У всіх завданнях необхідно реалізувати:

- конструктор ініціалізації (один або декілька),
- конструктор без аргументів і
- конструктор копіювання.

Вказані в завданні операції реалізуються за допомогою перевантаження підходящих операцій. У всіх завданнях обов'язково мають бути реалізовані відповідні операції:

- присвоєння,
- введення з клавіатури,
- виводу на екран,
- приведення типу – перетворення у літерний рядок.

Також треба реалізувати операції

- інкременту в обох формах (префіксній та постфіксній) і
- декременту в обох формах (префіксній та постфіксній), – для числових полів (наприклад: так, як вказано у варіантах завдань Лабораторної роботи № 2.3).

Перевантаження операцій виконується таким чином: підходящі операції реалізуються як методи класу, а інші – як зовнішні дружні функції.

Для демонстрації роботи з об'єктами нового типу у всіх завданнях потрібно написати головну функцію. У програмі обов'язково мають бути продемонстровані різні способи створення об'єктів і масивів об'єктів. Програма має демонструвати використання всіх функцій і методів. Вона має виводити на екран розмір класу в режимі `#pragma pack(1)` і без нього.

Визначення класів та реалізації методів слід розмістити в окремих модулях.

**Завдання наступне:**

Виконати завдання свого варіанту Лабораторної роботи № 1.7 (Композиція класів та об'єктів) з конструкторами і перевантаженням операцій.

**Лабораторна робота № 1.7:**

У всіх завданнях потрібно реалізувати по два-три класи. Один клас є основним, всі решту – допоміжні. Допоміжні класи мають бути визначені як незалежні. Об'єкти допоміжних класів мають використовуватися як поля основного класу.

Визначення класів та реалізації методів слід розмістити в окремих модулях.

Варіанти завдань наступні:

**Варіант 1.**

Реалізувати клас **Account**:

Клас **Account**, – банківський рахунок. У класі мають бути чотири поля:

- прізвище власника,
- номер рахунку,
- відсоток нарахування і
- сума в гривнях.

Відкриття нового рахунку виконується операцією ініціалізації. Необхідно виконувати наступні операції:

- змінити власника рахунку,
- зняти деяку суму грошей з рахунку,
- покласти гроші на рахунок,
- нарахувати відсотки,
- перевести суму в долари,
- перевести суму в євро,
- отримати суму прописом (число перетворити у літерний рядок, наприклад 1992,28 → «одна тисяча дев'ятсот дев'яносто дві грн. 28 коп.»).

Для представлення суми використовувати клас **Money**:

Клас **Money** – для роботи з грошовими сумами. Сума має бути представлене двома полями:

- типу **long** для гривень і
- типу **unsigned char** – для копійок.

Дробова частина (копійки) при виводі на екран має бути відокремлена від цілої частини комою. Реалізувати операції:

- додавання,
- віднімання,
- ділення сум,
- ділення суми на дробове число,
- множення на дробове число,
- операції порівняння.

## Варіант 2.

Реалізувати клас Account:

Клас Account, – банківський рахунок. У класі мають бути чотири поля:

- прізвище власника,
- номер рахунку,
- відсоток нарахування і
- сума в гривнях.

Відкриття нового рахунку виконується операцією ініціалізації. Необхідно виконувати наступні операції:

- змінити власника рахунку,
- зняти деяку суму грошей з рахунку,
- покласти гроші на рахунок,
- нарахувати відсотки,
- перевести суму в долари,
- перевести суму в євро,
- отримати суму прописом (число перетворити у літерний рядок, наприклад 1992,28 → «одна тисяча дев'ятсот дев'яносто дві грн. 28 коп.»).

Для представлення суми використовувати клас Money:

Номінали гривень можуть приймати значення 1, 2, 5, 10, 20, 50, 100, 200, 500. Копійки представити як 0.01 (1 копійка), 0.02 (2 копійки), 0.05 (5 копійок), 0.1 (10 копійок), 0.25 (25 копійок), 0.5 (50 копійок). Створити клас Money для роботи з грошовими сумами. Сума має бути представлена полями-номіналами, значеннями яких має бути кількість купюр відповідного номіналу. Поля:

- кількість банкнот по 500 грн.

- кількість банкнот по 200 грн.
- кількість банкнот по 100 грн.
- кількість банкнот по 50 грн.
- кількість банкнот по 20 грн.
- кількість банкнот по 10 грн.
- кількість банкнот по 5 грн.
- кількість банкнот по 2 грн.
- кількість банкнот по 1 грн.
- кількість монет по 50 коп.
- кількість монет по 25 коп.
- кількість монет по 10 коп.
- кількість монет по 5 коп.
- кількість монет по 2 коп.
- кількість монет по 1 коп.

Реалізувати:

- додавання сум,
- віднімання сум,
- ділення сум,
- ділення суми на дробове число,
- множення на дробове число,
- операції порівняння.

Дробова частина (копійки) при виводі на екран має бути відокремлена від цілої частини комою.

### Варіант 3.

Реалізувати клас `Calculator` з повним набором арифметичних операцій, використовуючи клас `Fraction`:

Клас `Fraction` – для роботи з дробовими числами. Число має бути представлене двома полями:

- ціла частина – довге ціле із знаком,
- дробова частина – без-знакове коротке ціле.

Реалізувати арифметичні операції:

- додавання,
- віднімання,

- множення,
- операції порівняння.

#### Варіант 4.

Реалізувати клас **Bankomat**:

Клас **Bankomat**, – моделює роботу банкомату. У класі мають міститися поля для зберігання:

- ідентифікаційного номера банкомату,
- інформації про поточну суму грошей, що залишилися у банкоматі,
- мінімальній і
- максимальній сумах, які дозволяється зняти клієнтові в один день.

Реалізувати:

- метод ініціалізації банкомату,
- метод завантаження купюр в банкомат,
- метод зняття певної суми грошей.

Метод зняття грошей має виконувати перевірку на коректність суми, що знімається: вона не може бути менше мінімального значення і не може перевищувати максимальне значення.

- метод `toString()` має перетворити у літерний рядок суму грошей, що залишилися в банкоматі.

Для представлення суми використовувати клас **Money**:

Номінали гривень можуть приймати значення 1, 2, 5, 10, 20, 50, 100, 200, 500. Копійки представити як 0.01 (1 копійка), 0.02 (2 копійки), 0.05 (5 копійок), 0.1 (10 копійок), 0.25 (25 копійок), 0.5 (50 копійок). Створити клас **Money** для роботи з грошовими сумами. Сума має бути представлена полями-номіналами, значеннями яких має бути кількість купюр відповідного номіналу. Поля:

- кількість банкнот по 500 грн.
- кількість банкнот по 200 грн.
- кількість банкнот по 100 грн.
- кількість банкнот по 50 грн.
- кількість банкнот по 20 грн.
- кількість банкнот по 10 грн.
- кількість банкнот по 5 грн.

- кількість банкнот по 2 грн.
- кількість банкнот по 1 грн.
- кількість монет по 50 коп.
- кількість монет по 25 коп.
- кількість монет по 10 коп.
- кількість монет по 5 коп.
- кількість монет по 2 коп.
- кількість монет по 1 коп.

Реалізувати:

- додавання сум,
- віднімання сум,
- ділення сум,
- ділення суми на дробове число,
- множення на дробове число,
- операції порівняння.

Дробова частина (копійки) при виводі на екран має бути відокремлена від цілої частини комою.

### Варіант 5\*.

Реалізувати клас `Fraction`:

Клас `Fraction` – для роботи з дробовими числами. Число має бути представлене двома полями:

- ціла частина – класу `DigitString`,
- дробова частина – без-знакове коротке ціле.

Реалізувати арифметичні операції:

- \* додавання,
- \* віднімання,
- \* множення,
- операції порівняння.

Для представлення цілої частини використовувати клас `DigitString`, а для представлення дробової частини без-знакове коротке ціле.

Клас `DigitString` – для роботи з цілими числами. Число має бути представлене символами-цифрами, які утворюють літерний рядок.



Мають бути реалізовані:

- \* всі арифметичні операції, присутні в C++ (без присвоєння), та
- операції порівняння.

### Варіант 6\*.

Реалізувати клас `Calculator` з повним набором арифметичних операцій, на основі класу `Fraction`:

Клас `Fraction` – для роботи з дробовими числами. Число має бути представлене двома полями:

- ціла частина – довге ціле із знаком,
- дробова частина – без-знакове коротке ціле.

Реалізувати арифметичні операції:

- \* додавання,
- \* віднімання,
- \* множення,
- операції порівняння.

Для представлення цілої частини використовувати клас `LongLong`, а для представлення дробової частини додатне дробове число типу `double`:

Клас `LongLong` – для роботи з 64 бітовими цілими числами. Число має бути представлене двома полями:

- типу `long` – старша частина,
- типу `unsigned long` – молодша частина.

Мають бути реалізовані:

- \* всі арифметичні операції, присутні в C++ (без присвоєння), та
- операції порівняння.

### Варіант 7.

Реалізувати клас `Triangle`:

Клас `Triangle` – для представлення трикутника. Поля даних:

- $a$ ,
- $b$ ,
- $c$  – сторони;
- $A$ ,
- $B$ ,

- $C$  – протилежні кути.

– мають включати кути і сторони. Потрібно реалізувати операції:

- отримання полів даних,
- зміни полів даних,
- обчислення площі,
- обчислення периметру,
- обчислення висот,
- визначення виду трикутника (рівносторонній, рівнобедрений або прямокутний).

Для представлення кутів використовувати клас **Angle**:

Клас **Angle** – для роботи з кутами на площині, що задаються величиною в градусах і хвилинах. Поля:

- *grades*
- *minutes*

Обов'язково мають бути реалізовані:

- переведення в радіани,
- приведення до діапазону  $0^\circ - 360^\circ$ ,
- збільшення кута на задану величину,
- зменшення кута на задану величину,
- отримання синуса,
- порівняння кутів.

## Варіант 8.

Реалізувати клас **Goods**:

Клас **Goods** – товари. У класі мають бути представлені поля:

- найменування товару,
- дата оформлення,
- ціна товару,
- кількість одиниць товару,
- номер накладної, по якій товар поступив на склад.

Реалізувати методи:

- зміни ціни товару,
- зміни кількості товару (збільшення і зменшення),
- обчислення вартості товару.

- Метод `toString()` має повертати у вигляді літерного рядка вартість товару.

Для представлення ціни використовувати клас `Money`:

Клас `Money` – для роботи з грошовими сумами. Число має бути представлене двома полями:

- типу `long` для гривень і
- типу `unsigned char` – для копійок.

Дробова частина (копійки) при виводі на екран має бути відокремлена від цілої частини комою. Реалізувати операції:

- додавання,
- віднімання,
- ділення сум,
- ділення суми на дробове число,
- множення на дробове число,
- операції порівняння.

Реалізувати метод уцінки товару, зменшуючи ціну на 1% за кожен день прострочення терміну придатності.

## Варіант 9.

Реалізувати клас `Triangle` з полями – координатами вершин:

Клас `Triangle` – для представлення трикутника. Поля даних:

- $P_1$ ,
- $P_2$ ,
- $P_3$  – точки (вершини трикутника),

Потрібно реалізувати операції:

- отримання полів даних,
- зміни полів даних,
- обчислення площі,
- обчислення периметру,
- обчислення висот,
- визначення виду трикутника (рівносторонній, рівнобедрений або прямокутний).
- `get_a()`,
- `get_b()`,
- `get_c()` – обчислення довжин сторін;

- `get_A( )`,
- `get_B( )`,
- `get_C( )` – обчислення величин протилежних кутів.

Для представлення координат вершин використовуйте клас **Point**:

Клас **Point** – для роботи з точками на площині. Координати точки – декартові. Поля:

- `x`
- `y`

Обов'язково мають бути реалізовані:

- переміщення точки по осі X,
- переміщення по осі Y,
- визначення відстані до початку координат,
- відстані між двома точками,
- перетворення у полярні координати,
- порівняння на рівність та нерівність.

## Варіант 10.

Реалізувати клас **Payment**:

Клас **Payment** – зарплата. У класі мають бути представлені поля:

- прізвище-ім'я-побатькові,
- ставка,
- рік поступлення на роботу,
- відсоток надбавки,
- прибутковий податок,
- кількість відпрацьованих днів в місяці,
- кількість робочих днів в місяці,
- нарахована і
- утримана суми.

Реалізувати методи:

- обчислення нарахованої суми,
- обчислення утриманої суми,
- обчислення суми, що видається на руки,
- обчислення стажу.

Стаж обчислюється як повна кількість років, що пройшли від року прийому на

роботу, до поточного року. Нарахування є сумою, нарахованою за відпрацьовані дні, і надбавки, тобто долі від першої суми. Утриманнями є відрахування до пенсійного фонду (1% від нарахованої суми) і прибутковий податок. Прибутковий податок складає 13% від нарахованої суми без відрахувань в пенсійний фонд.

Для представлення полів нарахувань і утримань використовувати клас **Money**:

Клас **Money** для роботи з грошовими сумами. Число має бути представлене двома полями:

- типу **long** для гривень і
- типу **unsigned char** – для копійок.

Дробова частина (копійки) при виводі на екран має бути відокремлена від цілої частини комою. Реалізувати операції:

- додавання,
- віднімання,
- ділення сум,
- ділення суми на дробове число,
- множення на дробове число,
- операції порівняння.

## **Варіант 11.**

Реалізувати клас **Money**:

Клас **Money** – для роботи з грошовими сумами. Число має бути представлене вкладеним об'єктом класу **Fraction**.

Дробова частина (копійки) при виводі на екран має бути відокремлена від цілої частини комою. Реалізувати операції:

- додавання,
- віднімання,
- ділення сум,
- ділення суми на дробове число,
- множення на дробове число,
- операції порівняння.

Для представлення величини грошової суми використовувати клас **Fraction**:

Клас **Fraction** – для роботи з дробовими числами. Число має бути представлене двома полями:

- ціла частина – довге ціле із знаком,
- дробова частина – без-знакове коротке ціле.

Реалізувати арифметичні операції:

- додавання,
- віднімання,
- множення,
- операції порівняння.

## Варіант 12.

Реалізувати клас `ModelWindow`, додавши поле для курсору:

Створити клас `ModelWindow` для роботи з моделями екранних вікон. В якості полів задаються:

- заголовок вікна,
- координати лівого верхнього кута,
- розмір по горизонталі,
- розмір по вертикалі,
- колір вікна,
- стан «видиме / невидиме»,
- стан «з рамкою / без рамки».

Координати і розміри вказуються в цілих числах. Реалізувати операції:

- пересування вікна по горизонталі,
- пересування вікна по вертикалі;
- зміна висоти і/або ширини вікна;
- зміна кольору;
- встановлення стану,
- отримання значення стану.

Операції пересування і зміни розміру мають здійснювати перевірку на перетин меж екрану. Функція виводу на екран має змінювати стан полів об'єкту.

Для представлення поля курсору використовуйте клас `Cursor`:

Клас `Cursor`. Полями є:

- $x$
- $y$  – координати курсору по горизонталі і вертикалі – цілі додатні числа,
- вид курсору – горизонтальний або вертикальний,

- розмір курсору – ціле число від 1 до 15.

Реалізувати методи:

- зміни координат курсору,
- зміни виду курсору,
- зміни розміру курсору,
- метод гасіння і
- метод відновлення курсору.

### Варіант 13\*.

Реалізувати клас **Set** (множина) не більше ніж з 64 елементів цілих чисел, використовуючи клас **BitString**:

Клас **BitString** – для роботи з 64-бітовими рядками. Бітовий рядок має бути представлений двома полями типу **unsigned long**. Мають бути реалізовані всі традиційні операції для роботи з бітами:

- **and**,
- **or**,
- **xor**,
- **not**.
- \* зсув ліворуч **shiftLeft** та
- \* зсув праворуч **shiftRight** на задану кількість бітів.

Клас **Set** (множина) має забезпечувати операції:

- включення елемента в множину,
- виключення елемента з множини,
- об'єднання,
- перетин і
- різницю множин,
- обчислення кількості елементів в множині.

### Варіант 14\*.

Реалізувати клас **Rational**:

Раціональний (нескоротний) дріб представляється парою цілих чисел  $(a, b)$ , де поля:

- $a$  – чисельник,
- $b$  – знаменник.

Клас **Rational** – для роботи з раціональними дробами. Обов’язково мають бути реалізовані наступні операції:

Припустимо, що  $(a, b)$  – перше число  $= a/b$  – перший об’єкт;  $(c, d)$  – друге число  $= c/d$  – другий об’єкт.

- \* додавання **add()**,  $(a, b) + (c, d) = (ad + bc, bd) = (ad + bc)/(bd)$ ;
- \* віднімання **sub()**,  $(a, b) - (c, d) = (ad - bc, bd)$ ;
- \* множення **mul()**,  $(a, b) \times (c, d) = (ac, bd)$ ;
- \* ділення **div()**,  $(a, b) / (c, d) = (ad, bc)$ ;
- порівняння **equal()**, **great()**, **less()**.

Має бути реалізована приватна функція скорочення дробу **Reduce()**, яка обов’язково викликається при виконанні арифметичних операцій.

Для представлення чисельника і знаменника використовувати клас **LongLong**:

Клас **LongLong** – для роботи з 64 бітовими цілими числами. Число має бути представлено двома полями:

- типу **long** – старша частина,
- типу **unsigned long** – молодша частина.

Мають бути реалізовані:

- \* всі арифметичні операції, присутні в C++ (без присвоєння), та
- операції порівняння.

## Варіант 15\*.

Реалізувати клас **Money**:

Клас **Money** – для роботи з грошовими сумами. Число має бути представлено двома полями:

- типу **LongLong** для гривень (див. далі) і
- типу **unsigned char** – для копійок.

Дробова частина (копійки) при виводі на екран має бути відокремлена від цілої частини комою. Реалізувати операції:

- \* додавання,
- \* віднімання,
- \* ділення сум,
- \* ділення суми на дробове число,
- \* множення на дробове число,



- операції порівняння.

Для представлення гривень використовувати клас `LongLong`:

Клас `LongLong` – для роботи з 64 бітовими цілими числами. Число має бути представлене двома полями:

- типу `long` – старша частина,
- типу `unsigned long` – молодша частина.

Мають бути реалізовані:

- \* всі арифметичні операції, присутні в C++ (без присвоєння), та
- операції порівняння.

### Варіант 16\*.

Реалізувати клас `Cursor`:

Клас `Cursor`. Полями є:

- $x$
- $y$  – координати курсору по горизонталі і вертикалі – цілі додатні числа,
- вид курсору – горизонтальний або вертикальний,
- розмір курсору – ціле число від 1 до 15.

Реалізувати методи:

- \* зміни координат курсору,
- зміни виду курсору,
- зміни розміру курсору,
- метод гасіння і
- метод відновлення курсору.

Для представлення координат використовувати клас `LongLong`:

Клас `LongLong` для роботи з 64 бітовими цілими числами. Число має бути представлене двома полями:

- типу `long` – старша частина,
- типу `unsigned long` – молодша частина.

Мають бути реалізовані:

- \* всі арифметичні операції, присутні в C++ (без присвоєння), та
- операції порівняння.

## Варіант 17.\*

Реалізувати клас `Account`:

Клас `Account`, – банківський рахунок. У класі мають бути поля:

- прізвище власника,
- номер рахунку,
- дата відкриття,
- відсоток нарахування і
- сума в гривнях.

Відкриття нового рахунку виконується операцією ініціалізації. Необхідно виконувати наступні операції:

- змінити власника рахунку,
- зняти деяку суму грошей з рахунку,
- покласти гроші на рахунок,
- нарахувати відсотки,
- перевести суму в долари,
- перевести суму в євро,
- отримати суму прописом (число перетворити у літерний рядок, наприклад 1992,28 → «одна тисяча дев'ятсот дев'яносто дві грн. 28 коп.»).

Додати поле – дату відкриття рахунку, використовуючи клас `Date`:

Клас `Date` – для роботи з датами у форматі «рік.місяць.день» з трьома полями типу `unsigned int`:

- рік,
- місяць і
- номер дня.

Клас має включати не менше трьох функцій ініціалізації:

- числами,
- літерним рядком виду «рік.місяць.день» (наприклад, «2004.08.31») і
- датою.

Обов'язковими операціями є:

- \* обчислення дати через задану кількість днів,
- \* віднімання заданої кількості днів з дати,
- визначення, чи рік – високосний,
- присвоєння,

- отримання окремих частин (рік, місяць, день),
- порівняння дат (рівно, до, після),
- \* обчислення кількості днів між датами.

Додати метод, що обчислює кількість днів, що пройшли з початку відкриття рахунку, і що додає по 0,01 % до відсотку нарахування за кожен день.

### Варіант 18.\*

Реалізувати клас **Goods**, додавши поле – дату надходження товару на склад.

Клас **Goods** – товари. У класі мають бути представлені поля:

- найменування товару,
- дата оформлення,
- ціна товару,
- кількість одиниць товару,
- номер накладної,
- по якій товар поступив на склад.

Реалізувати методи:

- зміни ціни товару,
- зміни кількості товару (збільшення і зменшення),
- обчислення вартості товару.
- Метод `toString()` має повертати у вигляді літерного рядка вартість товару.

Використовувати клас **Date**:

Клас **Date** – для роботи з датами у форматі «рік.місяць.день» з трьома полями типу `unsigned int`:

- рік,
- місяць і
- номер дня.

Клас має включати не менше трьох функцій ініціалізації:

- числами,
- літерним рядком виду «рік.місяць.день» (наприклад, «2004.08.31») і
- датою.

Обов'язковими операціями є:

- \* обчислення дати через задану кількість днів,
- \* віднімання заданої кількості днів з дати,

- визначення, чи рік – високосний,
- присвоєння,
- отримання окремих частин (рік, місяць, день),
- порівняння дат (рівно, до, після),
- \* обчислення кількості днів між датами.

Реалізувати метод, що обчислює термін зберігання товару.

## Варіант 19.\*

Реалізувати клас `Payment`:

Клас `Payment` – зарплата. У класі мають бути представлені поля:

- прізвище-ім'я-побатькові,
- ставка,
- рік поступлення на роботу,
- відсоток надбавки,
- прибутковий податок,
- кількість відпрацьованих днів в місяці,
- кількість робочих днів в місяці,
- нарахована і
- утримана суми.

Реалізувати методи:

- обчислення нарахованої суми,
- обчислення утриманої суми,
- обчислення суми, що видається на руки,
- обчислення стажу.

Стаж обчислюється як повна кількість років, що пройшли від року прийому на роботу, до поточного року. Нарахування є сумою, нарахованою за відпрацьовані дні, і надбавки, тобто долі від першої суми. Утриманнями є відрахування до пенсійного фонду (1% від нарахованої суми) і прибутковий податок. Прибутковий податок складає 13% від нарахованої суми без відрахувань в пенсійний фонд.

Замість поля-року використовувати поле-дату класу `Date`:

Клас `Date` – для роботи з датами у форматі «рік.місяць.день» з трьома полями типу `unsigned int`:

- рік,

- місяць і
- номер дня.

Клас має включати не менше трьох функцій ініціалізації:

- числами,
- літерним рядком виду «рік.місяць.день» (наприклад, «2004.08.31») і
- датою.

Обов'язковими операціями є:

- \* обчислення дати через задану кількість днів,
- \* віднімання заданої кількості днів з дати,
- визначення, чи рік – високосний,
- присвоєння,
- отримання окремих частин (рік, місяць, день),
- порівняння дат (рівно, до, після),
- \* обчислення кількості днів між датами.

Стаж слід обчислювати, використовуючи методи класу **Date**.

## Варіант 20.\*

Реалізувати клас **Bill**, що є разовим платежем за телефонну розмову. Клас має включати поля:

- прізвище платника,
- номер телефону,
- тариф за хвилину розмови,
- знижка (у відсотках),
- час початку розмови,
- час закінчення розмови,
- сума до оплати.

Для представлення часу використовуйте клас **Time**:

Клас **Time** – для роботи з часом у форматі «година:хвилина:секунда» з трьома полями типу **unsigned int**:

- година,
- хвилина і
- секунда.

Клас має включати не менше чотирьох функцій ініціалізації:

- числами,
- літерним рядком (наприклад, «23:59:59»),
- секундами від початку доби і
- часом.

Реалізувати методи:

- \* обчислення різниці між двома моментами часу в секундах,
- \* додавання часу і заданої кількості секунд,
- \* віднімання з часу заданої кількості секунд,
- порівняння моментів часу,
- переведення в секунди,
- переведення в хвилини (з округленням до цілої хвилини);
- отримання і зміни значень полів. Час розмови, який підлягає оплаті, обчислюється в хвилинах; неповна хвилина вважається за повну;
- метод `toString()` має видавати суму в гривнях.

## Варіант 21.

Реалізувати клас `Account`:

Клас `Account`, – банківський рахунок. У класі мають бути чотири поля:

- прізвище власника,
- номер рахунку,
- відсоток нарахування і
- сума в гривнях.

Відкриття нового рахунку виконується операцією ініціалізації. Необхідно виконувати наступні операції:

- змінити власника рахунку,
- зняти деяку суму грошей з рахунку,
- покласти гроші на рахунок,
- нарахувати відсотки,
- перевести суму в долари,
- перевести суму в євро,
- отримати суму прописом (число перетворити у літерний рядок, наприклад 1992,28 → «одна тисяча дев'ятсот дев'яносто дві грн. 28 коп.»).

Для представлення суми використовувати клас `Money`:

Клас **Money** – для роботи з грошовими сумами. Сума має бути представлене двома полями:

- типу **long** для гривень і
- типу **unsigned char** – для копійок.

Дробова частина (копійки) при виводі на екран має бути відокремлена від цілої частини комою. Реалізувати операції:

- додавання,
- віднімання,
- ділення сум,
- ділення суми на дробове число,
- множення на дробове число,
- операції порівняння.

## **Варіант 22.**

Реалізувати клас **Account**:

Клас **Account**, – банківський рахунок. У класі мають бути чотири поля:

- прізвище власника,
- номер рахунку,
- відсоток нарахування і
- сума в гривнях.

Відкриття нового рахунку виконується операцією ініціалізації. Необхідно виконувати наступні операції:

- змінити власника рахунку,
- зняти деяку суму грошей з рахунку,
- покласти гроші на рахунок,
- нарахувати відсотки,
- перевести суму в долари,
- перевести суму в євро,
- отримати суму прописом (число перетворити у літерний рядок, наприклад 1992,28 → «одна тисяча дев'ятсот дев'яносто дві грн. 28 коп.»).

Для представлення суми використовувати клас **Money**:

Номінали гривень можуть приймати значення 1, 2, 5, 10, 20, 50, 100, 200, 500. Копійки представити як 0.01 (1 копійка), 0.02 (2 копійки), 0.05 (5 копійок), 0.1 (10 копійок), 0.25 (25

копійок), 0.5 (50 копійок). Створити клас **Money** для роботи з грошовими сумами. Сума має бути представлена полями-номіналами, значеннями яких має бути кількість купюр відповідного номіналу. Поля:

- кількість банкнот по 500 грн.
- кількість банкнот по 200 грн.
- кількість банкнот по 100 грн.
- кількість банкнот по 50 грн.
- кількість банкнот по 20 грн.
- кількість банкнот по 10 грн.
- кількість банкнот по 5 грн.
- кількість банкнот по 2 грн.
- кількість банкнот по 1 грн.
- кількість монет по 50 коп.
- кількість монет по 25 коп.
- кількість монет по 10 коп.
- кількість монет по 5 коп.
- кількість монет по 2 коп.
- кількість монет по 1 коп.

Реалізувати:

- додавання сум,
- віднімання сум,
- ділення сум,
- ділення суми на дробове число,
- множення на дробове число,
- операції порівняння.

Дробова частина (копійки) при виводі на екран має бути відокремлена від цілої частини комою.

### **Варіант 23.**

Реалізувати клас **Calculator** з повним набором арифметичних операцій, використовуючи клас **Fraction**:

Клас **Fraction** – для роботи з дробовими числами. Число має бути представлене двома полями:

- ціла частина – довге ціле із знаком,



- дробова частина – без-знакове коротке ціле.

Реалізувати арифметичні операції:

- додавання,
- віднімання,
- множення,
- операції порівняння.

## Варіант 24.

Реалізувати клас **Bankomat**:

Клас **Bankomat**, – моделює роботу банкомату. У класі мають міститися поля для зберігання:

- ідентифікаційного номера банкомату,
- інформації про поточну суму грошей, що залишилася у банкоматі,
- мінімальній і
- максимальній сумах, які дозволяється зняти клієнтові в один день.

Реалізувати:

- метод ініціалізації банкомату,
- метод завантаження купюр в банкомат,
- метод зняття певної суми грошей.

Метод зняття грошей має виконувати перевірку на коректність суми, що знімається: вона не може бути менше мінімального значення і не може перевищувати максимальне значення.

- метод `toString()` має перетворити у літерний рядок суму грошей, що залишилася в банкоматі.

Для представлення суми використовувати клас **Money**:

Номінали гривень можуть приймати значення 1, 2, 5, 10, 20, 50, 100, 200, 500. Копійки представити як 0.01 (1 копійка), 0.02 (2 копійки), 0.05 (5 копійок), 0.1 (10 копійок), 0.25 (25 копійок), 0.5 (50 копійок). Створити клас **Money** для роботи з грошовими сумами. Сума має бути представлена полями-номіналами, значеннями яких має бути кількість купюр відповідного номіналу. Поля:

- кількість банкнот по 500 грн.
- кількість банкнот по 200 грн.
- кількість банкнот по 100 грн.

- кількість банкнот по 50 грн.
- кількість банкнот по 20 грн.
- кількість банкнот по 10 грн.
- кількість банкнот по 5 грн.
- кількість банкнот по 2 грн.
- кількість банкнот по 1 грн.
- кількість монет по 50 коп.
- кількість монет по 25 коп.
- кількість монет по 10 коп.
- кількість монет по 5 коп.
- кількість монет по 2 коп.
- кількість монет по 1 коп.

Реалізувати:

- додавання сум,
- віднімання сум,
- ділення сум,
- ділення суми на дробове число,
- множення на дробове число,
- операції порівняння.

Дробова частина (копійки) при виводі на екран має бути відокремлена від цілої частини комою.

### Варіант 25\*.

Реалізувати клас **Fraction**:

Клас **Fraction** – для роботи з дробовими числами. Число має бути представлене двома полями:

- ціла частина – класу **DigitString**,
- дробова частина – без-знакове коротке ціле.

Реалізувати арифметичні операції:

- \* додавання,
- \* віднімання,
- \* множення,
- операції порівняння.

Для представлення цілої частини використовувати клас `DigitString`, а для представлення дробової частини без-знакове коротке ціле.

Клас `DigitString` – для роботи з цілими числами. Число має бути представлене символами-цифрами, які утворюють літерний рядок.

Мають бути реалізовані:

- \* всі арифметичні операції, присутні в C++ (без присвоєння), та
- операції порівняння.

### **Варіант 26\*.**

Реалізувати клас `Calculator` з повним набором арифметичних операцій, на основі класу `Fraction`:

Клас `Fraction` – для роботи з дробовими числами. Число має бути представлене двома полями:

- ціла частина – довге ціле із знаком,
- дробова частина – без-знакове коротке ціле.

Реалізувати арифметичні операції:

- \* додавання,
- \* віднімання,
- \* множення,
- операції порівняння.

Для представлення цілої частини використовувати клас `LongLong`, а для представлення дробової частини додатне дробове число типу `double`:

Клас `LongLong` – для роботи з 64 бітовими цілими числами. Число має бути представлене двома полями:

- типу `long` – старша частина,
- типу `unsigned long` – молодша частина.

Мають бути реалізовані:

- \* всі арифметичні операції, присутні в C++ (без присвоєння), та
- операції порівняння.

### **Варіант 27.**

Реалізувати клас `Triangle`:

Клас `Triangle` – для представлення трикутника. Поля даних:

- $a$ ,

- $b$ ,
- $c$  – сторони;
- $A$ ,
- $B$ ,
- $C$  – протилежні кути.

– мають включати кути і сторони. Потрібно реалізувати операції:

- отримання полів даних,
- зміни полів даних,
- обчислення площі,
- обчислення периметру,
- обчислення висот,
- визначення виду трикутника (рівносторонній, рівнобедрений або прямокутний).

Для представлення кутів використовувати клас **Angle**:

Клас **Angle** – для роботи з кутами на площині, що задаються величиною в градусах і хвилинах. Поля:

- *grades*
- *minutes*

Обов'язково мають бути реалізовані:

- переведення в радіани,
- приведення до діапазону  $0^\circ - 360^\circ$ ,
- збільшення кута на задану величину,
- зменшення кута на задану величину,
- отримання синуса,
- порівняння кутів.

## Варіант 28.

Реалізувати клас **Goods**:

Клас **Goods** – товари. У класі мають бути представлені поля:

- найменування товару,
- дата оформлення,
- ціна товару,
- кількість одиниць товару,
- номер накладної, по якій товар поступив на склад.

Реалізувати методи:

- зміни ціни товару,
- зміни кількості товару (збільшення і зменшення),
- обчислення вартості товару.
- Метод `toString()` має повертати у вигляді літерного рядка вартість товару.

Для представлення ціни використовувати клас `Money`:

Клас `Money` – для роботи з грошовими сумами. Число має бути представлене двома полями:

- типу `long` для гривень і
- типу `unsigned char` – для копійок.

Дробова частина (копійки) при виводі на екран має бути відокремлена від цілої частини комою. Реалізувати операції:

- додавання,
- віднімання,
- ділення сум,
- ділення суми на дробове число,
- множення на дробове число,
- операції порівняння.

Реалізувати метод уцінки товару, зменшуючи ціну на 1% за кожен день прострочення терміну придатності.

## Варіант 29.

Реалізувати клас `Triangle` з полями – координатами вершин:

Клас `Triangle` – для представлення трикутника. Поля даних:

- $P_1$ ,
- $P_2$ ,
- $P_3$  – точки (вершини трикутника),

Потрібно реалізувати операції:

- отримання полів даних,
- зміни полів даних,
- обчислення площі,
- обчислення периметру,
- обчислення висот,

- визначення виду трикутника (рівносторонній, рівнобедрений або прямокутний).
- *get\_a( )*,
- *get\_b( )*,
- *get\_c( )* – обчислення довжин сторін;
- *get\_A( )*,
- *get\_B( )*,
- *get\_C( )* – обчислення величин протилежних кутів.

Для представлення координат вершин використовуйте клас **Point**:

Клас **Point** – для роботи з точками на площині. Координати точки – декартові. Поля:

- *x*
- *y*

Обов'язково мають бути реалізовані:

- переміщення точки по осі X,
- переміщення по осі Y,
- визначення відстані до початку координат,
- відстані між двома точками,
- перетворення у полярні координати,
- порівняння на рівність та нерівність.

### Варіант 30.

Реалізувати клас **Payment**:

Клас **Payment** – зарплата. У класі мають бути представлені поля:

- прізвище-ім'я-побатькові,
- ставка,
- рік поступлення на роботу,
- відсоток надбавки,
- прибутковий податок,
- кількість відпрацьованих днів в місяці,
- кількість робочих днів в місяці,
- нарахована і
- утримана суми.

Реалізувати методи:

- обчислення нарахованої суми,

- обчислення утриманої суми,
- обчислення суми, що видається на руки,
- обчислення стажу.

Стаж обчислюється як повна кількість років, що пройшли від року прийому на роботу, до поточного року. Нарахування є сумою, нарахованою за відпрацьовані дні, і надбавки, тобто долі від першої суми. Утриманнями є відрахування до пенсійного фонду (1% від нарахованої суми) і прибутковий податок. Прибутковий податок складає 13% від нарахованої суми без відрахувань в пенсійний фонд.

Для представлення полів нарахувань і утримань використовувати клас **Money**:

Клас **Money** для роботи з грошовими сумами. Число має бути представлене двома полями:

- типу **long** для гривень і
- типу **unsigned char** – для копійок.

Дробова частина (копійки) при виводі на екран має бути відокремлена від цілої частини комою. Реалізувати операції:

- додавання,
- віднімання,
- ділення сум,
- ділення суми на дробове число,
- множення на дробове число,
- операції порівняння.

### **Варіант 31.**

Реалізувати клас **Money**:

Клас **Money** – для роботи з грошовими сумами. Число має бути представлене вкладеним об'єктом класу **Fraction**.

Дробова частина (копійки) при виводі на екран має бути відокремлена від цілої частини комою. Реалізувати операції:

- додавання,
- віднімання,
- ділення сум,
- ділення суми на дробове число,
- множення на дробове число,

- операції порівняння.

Для представлення величини грошової суми використовувати клас **Fraction**:

Клас **Fraction** – для роботи з дробовими числами. Число має бути представлене двома полями:

- ціла частина – довге ціле із знаком,
- дробова частина – без-знакове коротке ціле.

Реалізувати арифметичні операції:

- додавання,
- віднімання,
- множення,
- операції порівняння.

## Варіант 32.

Реалізувати клас **ModelWindow**, додавши поле для курсору:

Створити клас **ModelWindow** для роботи з моделями екранних вікон. В якості полів задаються:

- заголовок вікна,
- координати лівого верхнього кута,
- розмір по горизонталі,
- розмір по вертикалі,
- колір вікна,
- стан «видиме / невидиме»,
- стан «з рамкою / без рамки».

Координати і розміри вказуються в цілих числах. Реалізувати операції:

- пересування вікна по горизонталі,
- пересування вікна по вертикалі;
- зміна висоти і/або ширини вікна;
- зміна кольору;
- встановлення стану,
- отримання значення стану.

Операції пересування і зміни розміру мають здійснювати перевірку на перетин меж екрану. Функція виводу на екран має змінювати стан полів об'єкту.



Для представлення поля курсору використовуйте клас **Cursor**:

Клас **Cursor**. Полями є:

- $x$
- $y$  – координати курсору по горизонталі і вертикалі – цілі додатні числа,
- вид курсору – горизонтальний або вертикальний,
- розмір курсору – ціле число від 1 до 15.

Реалізувати методи:

- зміни координат курсору,
- зміни виду курсору,
- зміни розміру курсору,
- метод гасіння і
- метод відновлення курсору.

### **Варіант 33\*.**

Реалізувати клас **Set** (множина) не більше ніж з 64 елементів цілих чисел, використовуючи клас **BitString**:

Клас **BitString** – для роботи з 64-бітовими рядками. Бітовий рядок має бути представлений двома полями типу **unsigned long**. Мають бути реалізовані всі традиційні операції для роботи з бітами:

- **and**,
- **or**,
- **xor**,
- **not**.
- \* зсув ліворуч **shiftLeft** та
- \* зсув праворуч **shiftRight** на задану кількість бітів.

Клас **Set** (множина) має забезпечувати операції:

- включення елемента в множину,
- виключення елемента з множини,
- об'єднання,
- перетин і
- різницю множин,
- обчислення кількості елементів в множині.

### Варіант 34\*.

Реалізувати клас **Rational**:

Раціональний (нескоротний) дріб представляється парою цілих чисел  $(a, b)$ , де поля:

- $a$  – чисельник,
- $b$  – знаменник.

Клас **Rational** – для роботи з раціональними дробами. Обов'язково мають бути реалізовані наступні операції:

Припустимо, що  $(a, b)$  – перше число  $= a/b$  – перший об'єкт;  $(c, d)$  – друге число  $= c/d$  – другий об'єкт.

- \* додавання **add()**,  $(a, b) + (c, d) = (ad + bc, bd) = (ad + bc)/(bd)$ ;
- \* віднімання **sub()**,  $(a, b) - (c, d) = (ad - bc, bd)$ ;
- \* множення **mul()**,  $(a, b) \times (c, d) = (ac, bd)$ ;
- \* ділення **div()**,  $(a, b) / (c, d) = (ad, bc)$ ;
- порівняння **equal()**, **great()**, **less()**.

Має бути реалізована приватна функція скорочення дробу **Reduce()**, яка обов'язково викликається при виконанні арифметичних операцій.

Для представлення чисельника і знаменника використовувати клас **LongLong**:

Клас **LongLong** – для роботи з 64 бітовими цілими числами. Число має бути представлене двома полями:

- типу **long** – старша частина,
- типу **unsigned long** – молодша частина.

Мають бути реалізовані:

- \* всі арифметичні операції, присутні в C++ (без присвоєння), та
- операції порівняння.

### Варіант 35\*.

Реалізувати клас **Money**:

Клас **Money** – для роботи з грошовими сумами. Число має бути представлене двома полями:

- типу **LongLong** для гривень (див. далі) і
- типу **unsigned char** – для копійок.

Дробова частина (копійки) при виводі на екран має бути відокремлена від цілої частини комою. Реалізувати операції:

- \* додавання,
- \* віднімання,
- \* ділення сум,
- \* ділення суми на дробове число,
- \* множення на дробове число,
- операції порівняння.

Для представлення гривень використовувати клас `LongLong`:

Клас `LongLong` – для роботи з 64 бітовими цілими числами. Число має бути представлене двома полями:

- типу `long` – старша частина,
- типу `unsigned long` – молодша частина.

Мають бути реалізовані:

- \* всі арифметичні операції, присутні в C++ (без присвоєння), та
- операції порівняння.

### **Варіант 36\*.**

Реалізувати клас `Cursor`:

Клас `Cursor`. Полями є:

- $x$
- $y$  – координати курсору по горизонталі і вертикалі – цілі додатні числа,
- вид курсору – горизонтальний або вертикальний,
- розмір курсору – ціле число від 1 до 15.

Реалізувати методи:

- \* зміни координат курсору,
- зміни виду курсору,
- зміни розміру курсору,
- метод гасіння і
- метод відновлення курсору.

Для представлення координат використовувати клас `LongLong`:

Клас `LongLong` для роботи з 64 бітовими цілими числами. Число має бути представлене двома полями:

- типу `long` – старша частина,

- типу `unsigned long` – молодша частина.

Мають бути реалізовані:

- \* всі арифметичні операції, присутні в C++ (без присвоєння), та
- операції порівняння.

### Варіант 37.\*

Реалізувати клас `Account`:

Клас `Account`, – банківський рахунок. У класі мають бути поля:

- прізвище власника,
- номер рахунку,
- дата відкриття,
- відсоток нарахування і
- сума в гривнях.

Відкриття нового рахунку виконується операцією ініціалізації. Необхідно виконувати наступні операції:

- змінити власника рахунку,
- зняти деяку суму грошей з рахунку,
- покласти гроші на рахунок,
- нарахувати відсотки,
- перевести суму в долари,
- перевести суму в євро,
- отримати суму прописом (число перетворити у літерний рядок, наприклад 1992,28 → «одна тисяча дев'ятсот дев'яносто дві грн. 28 коп.»).

Додати поле – дату відкриття рахунку, використовуючи клас `Date`:

Клас `Date` – для роботи з датами у форматі «рік.місяць.день» з трьома полями типу `unsigned int`:

- рік,
- місяць і
- номер дня.

Клас має включати не менше трьох функцій ініціалізації:

- числами,
- літерним рядком виду «рік.місяць.день» (наприклад, «2004.08.31») і
- датою.

Обов'язковими операціями є:

- \* обчислення дати через задану кількість днів,
- \* віднімання заданої кількості днів з дати,
- визначення, чи рік – високосний,
- присвоєння,
- отримання окремих частин (рік, місяць, день),
- порівняння дат (рівно, до, після),
- \* обчислення кількості днів між датами.

Додати метод, що обчислює кількість днів, що пройшли з початку відкриття рахунку, і що додає по 0,01 % до відсотку нарахування за кожен день.

### Варіант 38.\*

Реалізувати клас **Goods**, додавши поле – дату надходження товару на склад.

Клас **Goods** – товари. У класі мають бути представлені поля:

- найменування товару,
- дата оформлення,
- ціна товару,
- кількість одиниць товару,
- номер накладної,
- по якій товар поступив на склад.

Реалізувати методи:

- зміни ціни товару,
- зміни кількості товару (збільшення і зменшення),
- обчислення вартості товару.
- Метод `toString()` має повертати у вигляді літерного рядка вартість товару.

Використовувати клас **Date**:

Клас **Date** – для роботи з датами у форматі «рік.місяць.день» з трьома полями типу `unsigned int`:

- рік,
- місяць і
- номер дня.

Клас має включати не менше трьох функцій ініціалізації:

- числами,

- літерним рядком виду «рік.місяць.день» (наприклад, «2004.08.31») і
- датою.

Обов'язковими операціями є:

- \* обчислення дати через задану кількість днів,
- \* віднімання заданої кількості днів з дати,
- визначення, чи рік – високосний,
- присвоєння,
- отримання окремих частин (рік, місяць, день),
- порівняння дат (рівно, до, після),
- \* обчислення кількості днів між датами.

Реалізувати метод, що обчислює термін зберігання товару.

### Варіант 39.\*

Реалізувати клас **Payment**:

Клас **Payment** – зарплата. У класі мають бути представлені поля:

- прізвище-ім'я-побатькові,
- ставка,
- рік поступлення на роботу,
- відсоток надбавки,
- прибутковий податок,
- кількість відпрацьованих днів в місяці,
- кількість робочих днів в місяці,
- нарахована і
- утримана суми.

Реалізувати методи:

- обчислення нарахованої суми,
- обчислення утриманої суми,
- обчислення суми, що видається на руки,
- обчислення стажу.

Стаж обчислюється як повна кількість років, що пройшли від року прийому на роботу, до поточного року. Нарахування є сумою, нарахованою за відпрацьовані дні, і надбавки, тобто долі від першої суми. Утриманнями є відрахування до пенсійного фонду (1% від нарахованої суми) і прибутковий податок. Прибутковий податок складає 13% від нарахованої суми без відрахувань в пенсійний фонд.

Замість поля-року використовувати поле-дату класу **Date**:

Клас **Date** – для роботи з датами у форматі «рік.місяць.день» з трьома полями типу **unsigned int**:

- рік,
- місяць і
- номер дня.

Клас має включати не менше трьох функцій ініціалізації:

- числами,
- літерним рядком виду «рік.місяць.день» (наприклад, «2004.08.31») і
- датою.

Обов'язковими операціями є:

- \* обчислення дати через задану кількість днів,
- \* віднімання заданої кількості днів з дати,
- визначення, чи рік – високосний,
- присвоєння,
- отримання окремих частин (рік, місяць, день),
- порівняння дат (рівно, до, після),
- \* обчислення кількості днів між датами.

Стаж слід обчислювати, використовуючи методи класу **Date**.

#### **Варіант 40.\***

Реалізувати клас **Bill**, що є разовим платежем за телефонну розмову. Клас має включати поля:

- прізвище платника,
- номер телефону,
- тариф за хвилину розмови,
- знижка (у відсотках),
- час початку розмови,
- час закінчення розмови,
- сума до оплати.

Для представлення часу використовуйте клас **Time**:

Клас **Time** – для роботи з часом у форматі «година:хвилина:секунда» з трьома

полями типу `unsigned int`:

- година,
- хвилина і
- секунда.

Клас має включати не менше чотирьох функцій ініціалізації:

- числами,
- літерним рядком (наприклад, «23:59:59»),
- секундами від початку доби і
- часом.

Реалізувати методи:

- \* обчислення різниці між двома моментами часу в секундах,
- \* додавання часу і заданої кількості секунд,
- \* віднімання з часу заданої кількості секунд,
- порівняння моментів часу,
- переведення в секунди,
- переведення в хвилини (з округленням до цілої хвилини);
- отримання і зміни значень полів. Час розмови, який підлягає оплаті, обчислюється в хвилинах; неповна хвилина вважається за повну;
- метод `toString( )` має видавати суму в гривнях.



## Лабораторна робота № 3.4. Демонстрація простого успадкування

### **Мета роботи**

Освоїти використання успадкування.

### **Питання, які необхідно вивчити та пояснити на захисті**

- 1) Поняття та призначення успадкування.
- 2) Поняття базового / похідного класу, клас-предок / клас-нащадок.
- 3) Загальний синтаксис директиви успадкування.
- 4) Види успадкування.
- 5) Ключі успадкування та директиви доступу.
- 6) Просте успадкування.
- 7) Відкрите успадкування – успадкування інтерфейсу.
- 8) Закрите успадкування – успадкування реалізації.
- 9) Успадкування операцій.

### **Варіанти завдань**

#### **Загальна частина завдань для варіантів 1-20**

Написати програму, яка демонструє роботу з об'єктами двох типів T1 і T2, для чого створити систему відповідних класів (T1 – базовий, T2 – похідний клас). Кожний об'єкт має мати ідентифікатор (у вигляді довільного рядку символів) і одне або декілька полів для зберігання стану (поточного значення) об'єкту.

Клієнту (функції `main`) мають бути доступні наступні основні операції (методи): створити об'єкт, видалити об'єкт, показати значення об'єкту і інші додаткові операції (які залежать від варіанта). Операції по створенню і видаленню об'єктів інкапсулювати в клас `Factory`. Передбачити меню, яке дозволяє продемонструвати задані операції.

При необхідності в класи, які розробляються, додають додаткові методи (наприклад, конструктор копіювання, операції присвоювання і т.п.) для забезпечення належного функціонування цих класів.

#### **Варіанти 1-24**

В Таблицях 1 і 2 перераховані можливі типи об'єктів і можливі додаткові операції над ними.

**Таблиця 1.** Перелік типів об'єктів

| Клас      | Об'єкт                                                     |
|-----------|------------------------------------------------------------|
| SymString | Символьний рядок (довільний рядок символів)                |
| BinString | Двійковий рядок (зображення двійкового числа) <sup>1</sup> |
| OctString | Вісімковий рядок (зображення вісімкового числа)            |
| DecString | Десятковий рядок (зображення десяткового числа)            |
| HexString | Шістнадцятковий рядок (зображення шістнадцяткового числа)  |

<sup>1</sup> Тут і далі в таблиці розглядаються тільки додатні числа.

**Таблиця 2.** Перелік додаткових операцій (методів)

| Операція (метод)                       | Опис                                                                                                                                                                                                                                                                                                |
|----------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| ShowBin()                              | Показати зображення двійкового значення об'єкта                                                                                                                                                                                                                                                     |
| ShowOct()                              | Показати зображення вісімкового значення об'єкта                                                                                                                                                                                                                                                    |
| ShowDec()                              | Показати зображення десяткового значення об'єкта                                                                                                                                                                                                                                                    |
| ShowHex()                              | Показати зображення шістнадцяткового значення об'єкта                                                                                                                                                                                                                                               |
| operator + (T& s1, T& s2) <sup>2</sup> | Для об'єктів SymString – конкатенація рядків s1 і s2; для об'єктів інших класів – додавання відповідних чисельних значень з наступним перетворенням до типу T.                                                                                                                                      |
| operator - (T& s1, T& s2)              | Для об'єктів SymString – якщо s2 являє собою підрядок в s1, то результатом являється рядок, який отриманий із s1 видаленням підрядка s2; в протилежному випадку повертається значення s1; для об'єктів інших класів – віднімання відповідних чисельних значень з наступним перетворенням до типу T. |

<sup>2</sup> Тут T – будь-який із типів T1 і T2.

Примітка: перші чотири операції можуть застосовуватися до об'єктів любых класів, за виключенням класу SymString.

Таблиця 3 містить специфікації варіантів.

**Таблиця 3.** Специфікації варіантів 1-24

| Варіант | T1        | T2        | Операції (методи)              |
|---------|-----------|-----------|--------------------------------|
| 1.      | SymString | BinString | ShowOct(), operator + (T&, T&) |
| 2.      | SymString | BinString | ShowOct(), operator - (T&, T&) |
| 3.      | SymString | BinString | ShowDec(), operator + (T&, T&) |
| 4.      | SymString | BinString | ShowDec(), operator - (T&, T&) |
| 5.      | SymString | BinString | ShowHex(), operator + (T&, T&) |
| 6.      | SymString | BinString | ShowHex(), operator - (T&, T&) |
| 7.      | SymString | OctString | ShowBin(), operator + (T&, T&) |

|     |           |           |                                |
|-----|-----------|-----------|--------------------------------|
| 8.  | SymString | OctString | ShowBin(), operator - (T&, T&) |
| 9.  | SymString | OctString | ShowDec(), operator + (T&, T&) |
| 10. | SymString | OctString | ShowDec(), operator - (T&, T&) |
| 11. | SymString | OctString | ShowHex(), operator + (T&, T&) |
| 12. | SymString | OctString | ShowHex(), operator - (T&, T&) |
| 13. | SymString | DecString | ShowBin(), operator + (T&, T&) |
| 14. | SymString | DecString | ShowBin(), operator - (T&, T&) |
| 15. | SymString | DecString | ShowOct(), operator + (T&, T&) |
| 16. | SymString | DecString | ShowOct(), operator - (T&, T&) |
| 17. | SymString | DecString | ShowHex(), operator + (T&, T&) |
| 18. | SymString | DecString | ShowHex(), operator - (T&, T&) |
| 19. | SymString | HexString | ShowBin(), operator + (T&, T&) |
| 20. | SymString | HexString | ShowBin(), operator - (T&, T&) |
| 21. | SymString | HexString | ShowOct(), operator + (T&, T&) |
| 22. | SymString | HexString | ShowOct(), operator - (T&, T&) |
| 23. | SymString | HexString | ShowDec(), operator + (T&, T&) |
| 24. | SymString | HexString | ShowDec(), operator - (T&, T&) |

## Варіанти 25-54

В Таблицях 4 і 5 перераховані можливі типи об'єктів і можливі додаткові операції над ними.

**Таблиця 4.** Перелік типів об'єктів

| Клас       | Об'єкт       |
|------------|--------------|
| Triangle   | Трикутник    |
| Quadrangle | Квадрат      |
| Rectangle  | Прямокутник  |
| Tetragon   | Чотирикутник |
| Pentagon   | П'ятикутник  |

**Таблиця 5.** Перелік додаткових операцій (методів)

| Операція (метод)            | Опис                                                           |
|-----------------------------|----------------------------------------------------------------|
| Move()                      | Перемістити об'єкт на площині                                  |
| Compare(T& ob1, T& ob2)     | Порівняти об'єкти ob1 і ob2 по площі                           |
| IsIntersect(T& ob1, T& ob2) | Визначити факт перетину об'єктів ob1 і ob2 (є перетин чи нема) |

|                           |                                                   |
|---------------------------|---------------------------------------------------|
| IsInclude(T& ob1, T& ob2) | Визначити факт включення об'єкту ob2 в об'єкт ob1 |
|---------------------------|---------------------------------------------------|

Таблиця 6 містить специфікації варіантів.

**Таблиця 6.** Специфікації варіантів 25-54

| Варіант | T1         | T2         | Операції (методи)           |
|---------|------------|------------|-----------------------------|
| 25.     | Triangle   | Quadrangle | Move(), Compare(T&, T&)     |
| 26.     | Triangle   | Quadrangle | Move(), IsIntersect(T&, T&) |
| 27.     | Triangle   | Quadrangle | Move(), IsInclude(T&, T&)   |
| 28.     | Triangle   | Rectangle  | Move(), Compare(T&, T&)     |
| 29.     | Triangle   | Rectangle  | Move(), IsIntersect(T&, T&) |
| 30.     | Triangle   | Rectangle  | Move(), IsInclude(T&, T&)   |
| 31.     | Triangle   | Tetragon   | Move(), Compare(T&, T&)     |
| 32.     | Triangle   | Tetragon   | Move(), IsIntersect(T&, T&) |
| 33.     | Triangle   | Tetragon   | Move(), IsInclude(T&, T&)   |
| 34.     | Triangle   | Pentagon   | Move(), Compare(T&, T&)     |
| 35.     | Triangle   | Pentagon   | Move(), IsIntersect(T&, T&) |
| 36.     | Triangle   | Pentagon   | Move(), IsInclude(T&, T&)   |
| 37.     | Quadrangle | Rectangle  | Move(), Compare(T&, T&)     |
| 38.     | Quadrangle | Rectangle  | Move(), IsIntersect(T&, T&) |
| 39.     | Quadrangle | Rectangle  | Move(), IsInclude(T&, T&)   |
| 40.     | Quadrangle | Tetragon   | Move(), Compare(T&, T&)     |
| 41.     | Quadrangle | Tetragon   | Move(), IsIntersect(T&, T&) |
| 42.     | Quadrangle | Tetragon   | Move(), IsInclude(T&, T&)   |
| 43.     | Quadrangle | Pentagon   | Move(), Compare(T&, T&)     |
| 44.     | Quadrangle | Pentagon   | Move(), IsIntersect(T&, T&) |
| 45.     | Quadrangle | Pentagon   | Move(), IsInclude(T&, T&)   |
| 46.     | Rectangle  | Tetragon   | Move(), Compare(T&, T&)     |
| 47.     | Rectangle  | Tetragon   | Move(), IsIntersect(T&, T&) |
| 48.     | Rectangle  | Tetragon   | Move(), IsInclude(T&, T&)   |
| 49.     | Rectangle  | Pentagon   | Move(), Compare(T&, T&)     |
| 50.     | Rectangle  | Pentagon   | Move(), IsIntersect(T&, T&) |
| 51.     | Rectangle  | Pentagon   | Move(), IsInclude(T&, T&)   |
| 52.     | Tetragon   | Pentagon   | Move(), Compare(T&, T&)     |
| 53.     | Tetragon   | Pentagon   | Move(), IsIntersect(T&, T&) |
| 54.     | Tetragon   | Pentagon   | Move(), IsInclude(T&, T&)   |

## Лабораторна робота № 3.5. Масиви та успадкування

### **Мета роботи**

Освоїти використання успадкування та масивів.

### **Питання, які необхідно вивчити та пояснити на захисті**

- 1) Поняття та призначення успадкування.
- 2) Поняття базового / похідного класу, клас-предок / клас-нащадок.
- 3) Загальний синтаксис директиви успадкування.
- 4) Види успадкування.
- 5) Ключі успадкування та директиви доступу.
- 6) Просте успадкування.
- 7) Відкрите успадкування – успадкування інтерфейсу.
- 8) Закрите успадкування – успадкування реалізації.
- 9) Обчислення кількості створених об'єктів.
- 10) Операція індексування.

### **Варіанти завдань**

У всіх завданнях реалізувати:

- операції вводу / виводу,
- методи отримання і
- методи встановлення значень полів, а також необхідні
- конструктори (якщо це не вказано в завданні явно).
- перетворення до літерного рядку реалізувати у вигляді функції перетворення `string`.

Конструктори і методи обов'язково мають перевіряти параметри на допустимість; у разі неправильних даних – виводити повідомлення про помилку і завершувати роботу.

Для демонстрації роботи з об'єктами нового типу у всіх завданнях потрібно написати головну функцію. У програмі мають бути реалізовані різні способи створення об'єктів і масивів об'єктів. Програма має демонструвати використання всіх функцій та методів.

Створити базовий клас `Array`, в якому визначити поле-масив відповідною типу і поле для зберігання кількості елементів у поточному об'єкті-масиві. Максимально можливий розмір масиву задається статичною константою.

Реалізуйте:

- конструктор ініціалізації, задаючи кількість елементів і початкове значення (за умовчанням 0);
- методи доступу до окремого елемента, перевантаживши операцію індексування [ ].

При цьому має виконуватися перевірка індексу на допустимість – реалізувати метод `rangeCheck( )` – перевірку заданого цілого числа на входження до діапазону.

Реалізувати клас свого завдання як похідний клас від класу `Array`, використовувати відкрите успадкування. У всіх завданнях необхідно реалізувати:

- конструктори ініціалізації і
- конструктор без аргументів. Вказані в завданні операції реалізуються за допомогою
- перевантаження відповідних операцій. У всіх завданнях мають бути підтримані
- відповідні операції з присвоюванням,
- ввід з клавіатури,
- вивід на екран.

**Зауваження** щодо реалізації множин:

Множина – це сукупність даних, для яких ідентифікується лише факт входження елемента у цю сукупність; це означає, що ніякої інформації про кількість однакових елементів ми не зберігаємо – тобто, дублікати не допускаються.

Множина представляється масивом, кожний елемент масиву набуває значень лише 0 або 1: 0 означає, що відповідного елемента немає у множині, 1 – що відповідний елемент у множині є. Незалежно від кількості елементів у множині, масив завжди має стільки елементів, скільки максимально може бути елементів у множині. Фактична кількість елементів у множині = кількості 1 у масиві.

Включення (добавлення) елемента у множину = записати 1 у відповідну комірку масиву. Вилучення (видалення) елемента із множини = записати 0 у відповідну комірку масиву.

Об'єднання 2-х множин = виконати операцію "або" над елементами 2-х масивів у відповідних позиціях, і т.д.

## Варіант 1.

На базі класу `Array` створити клас `Money` для роботи з грошовими сумами. Сума має бути представлена масивом, кожен елемент якого – десяткова цифра. Максимальна довжина

масиву – 100 цифр, реальна довжина задається конструктором. Молодший індекс відповідає молодшій цифрі грошової суми. Молодші дві цифри – копійки.

## **Варіант 2.**

На базі класу `Array` створити клас `Polinom` для роботи з многочленами до 100-ої степені. Коефіцієнти мають бути представлені масивом з 100 елементів-коефіцієнтів. Молодша степінь має менший індекс (нульова степінь – нульовий індекс). Розмір масиву задається як аргумент конструктора ініціалізації. Реалізувати арифметичні операції та операції порівняння, обчислення значення поліному для заданого значення  $x$ , диференціювання, інтегрування.

## **Варіант 3.**

На базі класу `Array` створити клас `BitString` для роботи з бітовими рядками довжиною не більше ніж 100 біт. Бітовий рядок має бути представлений масивом типу `unsigned char`, кожен елемент якого приймає значення 0 або 1. Реальний розмір масиву задається як аргумент конструктора ініціалізації. Мають бути реалізовані всі традиційні операції для роботи з бітовими рядками: `and`, `or`, `xor`, `not`. Реалізувати зсув ліворуч та зсув праворуч на задану кількість бітів.

## **Варіант 4.**

На базі класу `Array` створити клас `Decimal` для роботи з без-знаковими цілими десятковими числами, використовуючи для представлення числа масив з 100 елементів типу `unsigned char`, кожен з яких є десятковою цифрою. Молодша цифра має менший індекс (одиниці – в нульовому елементі масиву). Реальний розмір масиву задається як аргумент конструктора ініціалізації. Реалізувати арифметичні операції, аналогічні вбудованим в C++ операціям для цілих чисел та операції порівняння.

## **Варіант 5.**

На базі класу `Array` створити клас `Decimal` для роботи з знаковими цілими десятковими числами, використовуючи для представлення числа масив з 100 елементів типу `unsigned char`, кожен з яких є десятковою цифрою. Молодша цифра має менший індекс (одиниці – в нульовому елементі масиву). Реальний розмір масиву задається як аргумент конструктора ініціалізації. Реалізувати арифметичні операції, аналогічні вбудованим в C++ операціям для цілих чисел та операції порівняння.

Знак представити окремим полем `sign`.

## Варіант 6.

На базі класу `Array` створити клас `Hex` для роботи з без-знаковими цілими шістнадцятковими числами, використовуючи для представлення числа масив з 100 елементів типу `unsigned char`, кожний з яких є шістнадцятковою цифрою. Молодша цифра має менший індекс. Реальний розмір масиву задається як аргумент конструктора ініціалізації. Реалізувати арифметичні операції, аналогічні вбудованим в C++ операціям для цілих чисел та операції порівняння.

## Варіант 7.

На базі класу `Array` створити клас `Octal` для роботи з без-знаковими цілими вісімковими числами, використовуючи для представлення числа масив з 100 елементів типу `unsigned char`, кожен елемент якого є вісімковою цифрою. Молодша цифра має менший індекс (одиниці – в нульовому елементі масиву). Реальний розмір масиву задається як аргумент конструктора ініціалізації. Реалізувати арифметичні операції, аналогічні вбудованим в C++ для цілих чисел, та операції порівняння.

## Варіант 8.

На базі класу `Array` створити клас `Fraction` для роботи з дробовими десятковими числами. Кількість цифр в дробовій частині має задаватися в окремому полі і ініціалізуватися конструктором. Знак представити окремим полем `sign`.

## Варіант 9.

На базі класу `Array` створити клас `String` для роботи з літерними рядками, аналогічними рядкам `Turbo Pascal` (рядок представляється як масив символів із 256 байт, елементи з 1 по 255 представляють значущі символи, фактична довжина – в нульовому байті). Максимальний розмір рядка має задаватися. Обов'язково мають бути реалізовані: визначення довжини рядка, пошук підрядка у рядку, видалення підрядка з рядка, вставка підрядка у рядок, зчеплення двох рядків.

## Варіант 10.

На базі класу `Array` створити клас `Rational`, використовуючи два масиви з 100 елементів типу `unsigned char` для представлення чисельника і знаменника. Кожен елемент є десятковою цифрою. Молодша цифра має менший індекс (одиниці – в нульовому елементі масиву). Реальний розмір масиву задається як аргумент конструктора ініціалізації.



## Варіант 11.

На базі класу `Array` створити клас `Set` (множина) не більше ніж з 256 елементів-символів. Потрібно забезпечити включення елементу у множину, вилучення елементу із множини, об'єднання, перетин і різницю множин, обчислення кількості елементів в множині, перевірку наявності елемента у множині, перевірку входження однієї множини у іншу.

## Варіант 12.

На базі класу `Array` створити клас `Date`, використовуючи для представлення місяців масив структур. Структура має два поля: назва місяця (літерний рядок) та кількість днів в місяці. Індексом в масиві місяців є переліковий тип `month`. Реалізувати два варіанти класу: із звичайним масивом і статичним масивом місяців.

## Варіант 13.

На базі класу `Array` створити клас `Money`, використовуючи для представлення суми грошей масив структур. Структура має два поля: номінал купюри і кількість купюр цієї вартості. Номінали представити як переліковий тип `nominal`. Елемент масиву структур з меншим індексом містить менший номінал.

## Варіант 14.\*

Прайс-лист комп'ютерної фірми включає список моделей комп'ютерів, що продаються. Одна позиція списку (`Model`) містить марку комп'ютера, тип процесора, частоту роботи процесора, об'єм пам'яті, об'єм жорсткого диску, об'єм пам'яті відеокарти, ціну комп'ютера в умовних одиницях і кількість екземплярів, що є в наявності.

На базі класу `Array` створити клас `PriceList`, полями якого є дата його створення, номінал умовної одиниці в гривнях і список моделей комп'ютерів, що продаються. У списку не може бути двох моделей однакової марки. У класі `PriceList` реалізувати методи додавання, зміни і видалення запису про модель, метод пошуку інформації про модель по марці комп'ютера, за об'ємом пам'яті, диска і відеокарти (рівно або не менше заданого), а також метод підрахунку загальної суми.

Реалізувати методи об'єднання і перетину прайс-листів.

При реалізації має створюватися новий прайс-лист, а початкові прайс-листи не мають змінюватися. При об'єднанні новий прайс-лист має містити без повторень всі елементи, що містяться в обох прайс-листах - операндах. При перетині новий прайс-лист має складатися тільки з тих елементів, які є в обох прайс-листах - операндах. При відніманні новий прайс-лист має містити елементи першого прайс-листа - операнда, які відсутні в другому.

Методи додавання і зміни приймають як вхідний параметр об'єкт класу **Model**. Метод пошуку повертає об'єкт класу **Model** як результат.

### **Варіант 15.\***

Картка іноземного слова є структурою, що містить іноземне слово та його переклад. Для моделювання електронного словника іноземних слів слід на базі класу **Array** створити клас **Dictionary**. Цей клас має поле-назву словника і містить масив структур **WordCard**, що є картками іноземного слова. Назва словника задається при створенні нового словника, але має бути надана можливість його зміни під час роботи. Картки додаються в словник і видаляються з нього. Реалізувати пошук певного слова як окремий метод. Аргументом операції індексування має бути іноземне слово. У словнику не може бути карток-дублікатів.

Реалізувати операції об'єднання, перетину і обчислення різниці словників. При реалізації має створюватися новий словник, а початкові словники не мають змінюватися. При об'єднанні новий словник має містити без повторень всі слова, що містяться в обох словниках-операндах. При перетині новий словник має складатися тільки з тих слів, які є в обох словниках-операндах. При обчисленні різниці новий словник має містити слова першого словника-операнда, які відсутні в другому.

### **Варіант 16.\***

Одне тестове питання є структурою **Task** з наступними полями: текст питання; п'ять варіантів відповіді; номер правильної відповіді; бали за правильну відповідь, що нараховуються. Для моделювання набору тестових питань на базі класу **Array** створити клас **TestContent**, що містить масив тестових питань. Реалізувати методи додавання і видалення тестових питань, а також метод доступу до тестового завдання за його порядковим номером у списку. У масиві не може бути питань, що повторюються.

Реалізувати операцію злиття двох тестових наборів, операцію перетину і операцію обчислення різниці.

При реалізації має створюватися новий тестовий набір, а початкові тестові набори не мають змінюватися. При об'єднанні новий тестовий набір має містити без повторень всі елементи, що містяться в обох тестових наборах - операндах. При перетині новий тестовий набір має складатися тільки з тих елементів, які є в обох тестових наборах - операндах. При обчисленні різниці новий тестовий набір має містити елементи першого тестового набору - операнді, які відсутні в другому.

Додатково реалізувати операцію генерації конкретного об'єкту **Test** (об'ємом не більш **K** питань) з об'єкту типу **TestContent**.

## Варіант 17.\*

Картка персони містить прізвище і дату народження. На базі класу `Array` створити клас `ListPerson` для роботи з картотекою персоналій. Клас має містити масив карток персон. Реалізувати методи додавання і видалення карток персон, а також метод доступу до картки за прізвищем. Прізвища в масиві мають бути унікальні.

Реалізувати операції об'єднання двох картотек, операцію перетину і обчислення різниці.

При реалізації має створюватися нова картотека, а початкові картотеки не мають змінюватися. При об'єднанні нова картотека має містити без повторень всі елементи, що містяться в обох картотеках-операндах. При перетині нова картотека має складатися тільки з тих елементів, які є в обох картотеках-операндах. При обчисленні різниці нова картотека має містити елементи першої картотеки-операнді, які відсутні в другій.

Реалізувати метод, що видає за прізвищем знак зодіаку. Для цього в класі має бути оголошений масив структур `Zodiac` з полями: назва знаку зодіаку, дата початку і дата закінчення періоду. Індексом в масиві має бути переліковий тип `zodiac`. Реалізувати два варіанти класу: із звичайним масивом і статичним масивом `Zodiac`.

## Варіант 18.\*

Товарний чек містить список товарів, куплених покупцем в магазині. Один елемент списку є парою: товар-сума. Товар – це клас `Goods` з полями коду і найменування товару, ціни за одиницю товару, кількості одиниць товару, що купуються. У класі мають бути реалізовані методи доступу до полів для отримання і зміни інформації, а також метод обчислення суми оплати за товар.

Для моделювання товарного чеку на базі класу `Array` створити клас `Receipt`, полями якого є номер товарного чека, дата і час його створення, список товарів, що купуються. У класі `Receipt` реалізувати методи додавання, зміни і видалення запису про товар, що купується, метод пошуку інформації про певний вид товару за його кодом, а також метод підрахунку загальної суми, на яку були здійснені покупки. Методи додавання і зміни приймають як аргумент об'єкт класу `Goods`. Метод пошуку повертає об'єкт класу `Goods` як результат.

## Варіант 19.\*

Інформаційний запис про книгу в бібліотеці містить наступні поля: автор, назва, рік видання, видавництво, ціна.

Для моделювання облікової картки абонента на базі класу `Array` реалізувати клас `Subscriber`, що містить прізвище абонента, його бібліотечний номер і список взятих в бібліотеці книг. Один елемент списку складається з інформаційного запису про книгу, дати

видачі, необхідної дати повернення і признаку повернення. Реалізувати методи додавання книг в список і видалення книг з нього; метод пошуку книг, що підлягають поверненню; методи пошуку за автором, видавництвом та роком видання; метод обчислення вартості всіх книг, що підлягають поверненню.

Реалізувати операцію злиття двох облікових карток, операцію перетину і обчислення різниці.

При реалізації має створюватися нова картка, а початкові картки не мають змінюватися. При об'єднанні нова картка має містити без повторень всі елементи, що містяться в обох картках-операндах. При перетині нова картка має складатися тільки з тих елементів, які є в обох картках-операндах. При обчисленні різниці нова картка має містити елементи першої картки-операнді, які відсутні в другій.

Реалізувати операцію генерації конкретного об'єкту **Debt** (борг), що містить список книг, що підлягають поверненню, з об'єкту типу **Subscriber**.

## **Варіант 20.\***

Інформаційний запис про файл в каталозі містить поля: ім'я файлу, розширення, дату і час створення, атрибути «тільки зчитування», «прихований», «системний», розмір файлу на диску.

Для моделювання каталогу на базі класу **Array** створити клас **Directory**, що містить назву батьківського каталогу, кількість файлів в каталозі, список файлів в каталозі. Один елемент списку включає інформаційний запис про файл, дату останньої зміни, признак виділення і признак видалення. Реалізувати методи додавання файлів в каталог і видалення файлів з нього; метод пошуку файлу по імені, по розширенню, по даті створення; метод обчислення повного об'єму каталогу.

Реалізувати операцію об'єднання і операцію перетину каталогів.

При реалізації має створюватися новий каталог, а початкові каталоги не мають змінюватися. При об'єднанні новий каталог має містити без повторень всі елементи, що містяться в обох каталогах-операндах. При перетині новий каталог має складатися тільки з тих елементів, які є в обох каталогах-операндах. При обчисленні різниці новий каталог має містити елементи першого каталога-операнда, які відсутні в другому.

Реалізувати операцію генерації конкретного об'єкту **Group** (група), що містить список файлів, з об'єкту типу **Directory**. Має бути можливість вибирати групу файлів за ознакою видалення, по атрибутах, по даті створення (до або після), за об'ємом (менше або більше).

## Варіант 21.\*

Навантаження викладача за навчальний рік – це список дисциплін, що викладаються ним протягом року. Одна дисципліна представляється інформаційною структурою з полями: назва дисципліни, семестр проведення, кількість студентів, кількість аудиторних годин лекцій, кількість аудиторних годин практики, вид контролю (залік або іспит).

На базі класу `Array` створити клас `WorkTeacher`, що моделює бланк призначеного викладачеві навантаження. Клас містить прізвище викладача, дату затвердження, список дисциплін, що викладаються, об'єм повного навантаження в годинах і в ставках. Дисципліни в списку не можуть повторюватися. Об'єм в ставках обчислюється як результат від ділення об'єму в годинах на середню річну ставку, однакову для всіх викладачів кафедри. Елемент списку дисциплін, що викладаються, містить дисципліну, кількість годин, що виділяється на залік (0,2 год. на одного студента) або іспит (0,3 год. на студента), суму годин по дисципліні. Реалізувати додавання і видалення дисциплін; обчислення сумарного навантаження в годинах і ставках. Має здійснюватися контроль за перевищення навантаження – не більш, ніж півтори ставки.

## Варіант 22.\*

Навчальний план спеціальності є списком дисциплін, які студент має вивчити за час навчання. Одна дисципліна є структурою з полями: номер дисципліни в плані; тип дисципліни (нормативна, за вибором навчального закладу, за вибором студента); назва дисципліни; семестр, в якому дисципліна вивчається; вид підсумкового контролю (залік або іспит); загальна кількість годин, необхідна для вивчення дисципліни; кількість аудиторних годин, яка складається з лекційних годин і годин практики.

На базі класу `Array` створити клас `PlanEducation` для моделювання навчального плану спеціальності. Клас має містити код і назву спеціальності, дату затвердження, загальну кількість годин спеціальності за стандартом і список дисциплін. Один елемент списку дисциплін має містити запис про дисципліну, кількість годин для самостійної роботи (різниця між загальною кількістю годин і аудиторними годинами), ознака наявності курсової роботи, яка виконується по цій дисципліні. Реалізувати методи додавання і видалення дисциплін; метод пошуку дисципліни по семестру, за типом дисципліни, по виду підсумкового контролю; метод обчислення сумарної кількості годин всіх дисциплін; метод обчислення кількості іспитів і заліків по семестрах.

Реалізувати операцію об'єднання, операцію віднімання навчальних планів і операцію перетину навчальних планів.

При реалізації має створюватися новий навчальний план, а початкові навчальні плани не

мають змінюватися. При об'єднанні новий навчальний план має містити без повторень всі елементи, що містяться в обох навчальних планах - операндах. При перетині новий навчальний план має складатися тільки з тих елементів, які є в обох навчальних планах - операндах. При відніманні новий навчальний план має містити елементи першого навчального плану - операнді, які відсутні в другому.

Реалізувати операцію генерації конкретного об'єкту **Group** (група дисциплін), що містить список дисциплін, з об'єкту типу **PlanEducation**. Має бути можливість вибирати групу дисциплін за типом, по семестру, по вигляду підсумкового контролю, по наявності курсової роботи. Має здійснюватися контроль за сумарною кількістю годин, за кількістю іспитів в семестрі (не більше п'яти і не меншого трьох).

### **Варіант 23.\***

Один запис в списку запланованих справ є структурою **DailyItem**, яка містить час початку і закінчення роботи, опис і признак виконання.

На базі класу **Array** створити клас **DailySchedule**, що представляє собою план робіт на день. Реалізувати методи додавання, видалення і зміни планованої роботи. При додаванні перевіряти коректність часових рамок (вони не мають перетинатися з вже запланованими заходами). Реалізувати метод пошуку вільного проміжку часу. Умова пошуку задає розмір шуканого інтервалу, а також часові рамки, в які він має потрапляти. Метод пошуку повертає структуру **DailyItem** з порожнім описом виду робіт. Реалізувати операцію генерації об'єкту **Redo** (ще раз), що містить список справ, не виконаних протягом дня, з об'єкту типу **DailySchedule**.

### **Варіант 24.**

На базі класу **Array** створити клас **Money** для роботи з грошовими сумами. Сума має бути представлена масивом, кожен елемент якого – десяткова цифра. Максимальна довжина масиву – 100 цифр, реальна довжина задається конструктором. Молодший індекс відповідає молодшій цифрі грошової суми. Молодші дві цифри – копійки.

### **Варіант 25.**

На базі класу **Array** створити клас **Polinom** для роботи з многочленами до 100-ої степені. Коефіцієнти мають бути представлені масивом з 100 елементів-коефіцієнтів. Молодша степінь має менший індекс (нульова степінь – нульовий індекс). Розмір масиву задається як аргумент конструктора ініціалізації. Реалізувати арифметичні операції та операції порівняння, обчислення значення поліному для заданого значення  $x$ , диференціювання, інтегрування.

## Варіант 26.

На базі класу `Array` створити клас `BitString` для роботи з бітовими рядками довжиною не більше ніж 100 біт. Бітовий рядок має бути представлений масивом типу `unsigned char`, кожен елемент якого приймає значення 0 або 1. Реальний розмір масиву задається як аргумент конструктора ініціалізації. Мають бути реалізовані всі традиційні операції для роботи з бітовими рядками: `and`, `or`, `xor`, `not`. Реалізувати зсув ліворуч та зсув праворуч на задану кількість бітів.

## Варіант 27.

На базі класу `Array` створити клас `Decimal` для роботи з без-знаковими цілими десятковими числами, використовуючи для представлення числа масив з 100 елементів типу `unsigned char`, кожен з яких є десятковою цифрою. Молодша цифра має менший індекс (одиниці – в нульовому елементі масиву). Реальний розмір масиву задається як аргумент конструктора ініціалізації. Реалізувати арифметичні операції, аналогічні вбудованим в C++ операціям для цілих чисел та операції порівняння.

## Варіант 28.

На базі класу `Array` створити клас `Decimal` для роботи з знаковими цілими десятковими числами, використовуючи для представлення числа масив з 100 елементів типу `unsigned char`, кожен з яких є десятковою цифрою. Молодша цифра має менший індекс (одиниці – в нульовому елементі масиву). Реальний розмір масиву задається як аргумент конструктора ініціалізації. Реалізувати арифметичні операції, аналогічні вбудованим в C++ операціям для цілих чисел та операції порівняння.

Знак представити окремим полем `sign`.

## Варіант 29.

На базі класу `Array` створити клас `Hex` для роботи з без-знаковими цілими шістнадцятковими числами, використовуючи для представлення числа масив з 100 елементів типу `unsigned char`, кожен з яких є шістнадцятковою цифрою. Молодша цифра має менший індекс. Реальний розмір масиву задається як аргумент конструктора ініціалізації. Реалізувати арифметичні операції, аналогічні вбудованим в C++ операціям для цілих чисел та операції порівняння.

## Варіант 30.

На базі класу `Array` створити клас `Octal` для роботи з без-знаковими цілими вісімковими числами, використовуючи для представлення числа масив з 100 елементів типу `unsigned`

`char`, кожен елемент якого є вісімковою цифрою. Молодша цифра має менший індекс (одиниці – в нульовому елементі масиву). Реальний розмір масиву задається як аргумент конструктора ініціалізації. Реалізувати арифметичні операції, аналогічні вбудованим в C++ для цілих чисел, та операції порівняння.

### **Варіант 31.**

На базі класу `Array` створити клас `Fraction` для роботи з дробовими десятковими числами. Кількість цифр в дробовій частині має задаватися в окремому полі і ініціалізуватися конструктором. Знак представити окремим полем `sign`.

### **Варіант 32.**

На базі класу `Array` створити клас `String` для роботи з літерними рядками, аналогічними рядкам `Turbo Pascal` (рядок представляється як масив символів із 256 байт, елементи з 1 по 255 представляють значущі символи, фактична довжина – в нульовому байті). Максимальний розмір рядка має задаватися. Обов'язково мають бути реалізовані: визначення довжини рядка, пошук підрядка у рядку, видалення підрядка з рядка, вставка підрядка у рядок, зчеплення двох рядків.

### **Варіант 33.**

На базі класу `Array` створити клас `Rational`, використовуючи два масиви з 100 елементів типу `unsigned char` для представлення чисельника і знаменника. Кожен елемент є десятковою цифрою. Молодша цифра має менший індекс (одиниці – в нульовому елементі масиву). Реальний розмір масиву задається як аргумент конструктора ініціалізації.

### **Варіант 34.**

На базі класу `Array` створити клас `Set` (множина) не більше ніж з 256 елементів-символів. Потрібно забезпечити включення елемента у множину, вилучення елемента із множини, об'єднання, перетин і різницю множин, обчислення кількості елементів в множині, перевірку наявності елемента у множині, перевірку входження однієї множини у іншу.

### **Варіант 35.**

На базі класу `Array` створити клас `Date`, використовуючи для представлення місяців масив структур. Структура має два поля: назва місяця (літерний рядок) та кількість днів в місяці. Індексом в масиві місяців є переліковий тип `month`. Реалізувати два варіанти класу: із звичайним масивом і статичним масивом місяців.



## Варіант 36.

На базі класу `Array` створити клас `Money`, використовуючи для представлення суми грошей масив структур. Структура має два поля: номінал купюри і кількість купюр цієї вартості. Номінали представити як переліковий тип `nominal`. Елемент масиву структур з меншим індексом містить менший номінал.

## Варіант 37.\*

Прайс-лист комп'ютерної фірми включає список моделей комп'ютерів, що продаються. Одна позиція списку (`Model`) містить марку комп'ютера, тип процесора, частоту роботи процесора, об'єм пам'яті, об'єм жорсткого диску, об'єм пам'яті відеокарти, ціну комп'ютера в умовних одиницях і кількість екземплярів, що є в наявності.

На базі класу `Array` створити клас `PriceList`, полями якого є дата його створення, номінал умовної одиниці в гривнях і список моделей комп'ютерів, що продаються. У списку не може бути двох моделей однакової марки. У класі `PriceList` реалізувати методи додавання, зміни і видалення запису про модель, метод пошуку інформації про модель по марці комп'ютера, за об'ємом пам'яті, диска і відеокарти (рівно або не менше заданого), а також метод підрахунку загальної суми.

Реалізувати методи об'єднання і перетину прайс-листів.

При реалізації має створюватися новий прайс-лист, а початкові прайс-листи не мають змінюватися. При об'єднанні новий прайс-лист має містити без повторень всі елементи, що містяться в обох прайс-листах - операндах. При перетині новий прайс-лист має складатися тільки з тих елементів, які є в обох прайс-листах - операндах. При відніманні новий прайс-лист має містити елементи першого прайс-листа - операнда, які відсутні в другому.

Методи додавання і зміни приймають як вхідний параметр об'єкт класу `Model`. Метод пошуку повертає об'єкт класу `Model` як результат.

## Варіант 38.\*

Картка іноземного слова є структурою, що містить іноземне слово та його переклад. Для моделювання електронного словника іноземних слів слід на базі класу `Array` створити клас `Dictionary`. Цей клас має поле-назву словника і містить масив структур `WordCard`, що є картками іноземного слова. Назва словника задається при створенні нового словника, але має бути надана можливість його зміни під час роботи. Картки додаються в словник і видаляються з нього. Реалізувати пошук певного слова як окремий метод. Аргументом операції індексування має бути іноземне слово. У словнику не може бути карток-дублікатів.

Реалізувати операції об'єднання, перетину і обчислення різниці словників. При

реалізації має створюватися новий словник, а початкові словники не мають змінюватися. При об'єднанні новий словник має містити без повторень всі слова, що містяться в обох словниках-операндах. При перетині новий словник має складатися тільки з тих слів, які є в обох словниках-операндах. При обчисленні різниці новий словник має містити слова першого словника-операнда, які відсутні в другому.

### **Варіант 39.\***

Одне тестове питання є структурою `Task` з наступними полями: текст питання; п'ять варіантів відповіді; номер правильної відповіді; бали за правильну відповідь, що нараховуються. Для моделювання набору тестових питань на базі класу `Array` створити клас `TestContent`, що містить масив тестових питань. Реалізувати методи додавання і видалення тестових питань, а також метод доступу до тестового завдання за його порядковим номером у списку. У масиві не може бути питань, що повторюються.

Реалізувати операцію злиття двох тестових наборів, операцію перетину і операцію обчислення різниці.

При реалізації має створюватися новий тестовий набір, а початкові тестові набори не мають змінюватися. При об'єднанні новий тестовий набір має містити без повторень всі елементи, що містяться в обох тестових наборах - операндах. При перетині новий тестовий набір має складатися тільки з тих елементів, які є в обох тестових наборах - операндах. При обчисленні різниці новий тестовий набір має містити елементи першого тестового набору - операнді, які відсутні в другому.

Додатково реалізувати операцію генерації конкретного об'єкту `Test` (об'ємом не більш `K` питань) з об'єкту типу `TestContent`.

### **Варіант 40.\***

Картка персони містить прізвище і дату народження. На базі класу `Array` створити клас `ListPerson` для роботи з картотекою персоналій. Клас має містити масив карток персон. Реалізувати методи додавання і видалення карток персон, а також метод доступу до картки за прізвищем. Прізвища в масиві мають бути унікальні.

Реалізувати операції об'єднання двох картотек, операцію перетину і обчислення різниці.

При реалізації має створюватися нова картотека, а початкові картотеки не мають змінюватися. При об'єднанні нова картотека має містити без повторень всі елементи, що містяться в обох картотеках-операндах. При перетині нова картотека має складатися тільки з тих елементів, які є в обох картотеках-операндах. При обчисленні різниці нова картотека має містити елементи першої картотеки-операнді, які відсутні в другій.

Реалізувати метод, що видає за прізвищем знак зодіаку. Для цього в класі має бути оголошений масив структур **Zodiac** з полями: назва знаку зодіаку, дата початку і дата закінчення періоду. Індексом в масиві має бути переліковий тип **zodiac**. Реалізувати два варіанти класу: із звичайним масивом і статичним масивом **Zodiac**.

## Лабораторна робота № 3.6. Множинне успадкування

### Мета роботи

Освоїти використання множинного успадкування.

### Питання, які необхідно вивчити та пояснити на захисті

- 1) Поняття та призначення успадкування.
- 2) Поняття базового / похідного класу, клас-предок / клас-нащадок.
- 3) Загальний синтаксис директиви успадкування.
- 4) Види успадкування.
- 5) Ключі успадкування та директиви доступу.
- 6) Просте успадкування.
- 7) Відкрите успадкування – успадкування інтерфейсу.
- 8) Закрите успадкування – успадкування реалізації.
- 9) Множинне успадкування.

### Зразок виконання завдання

Подається лише умова завдання та текст програми.

### Варіант 0.

#### Умова завдання

Побудувати ієрархію класів відповідно до схеми успадкування, заданої у варіанті завдання.

Кожен клас має містити:

- власне поле, ім'я якого – це ім'я класу, записане в нижньому регістрі (малими літерами), – див. зразок виконання завдання;
- конструктор ініціалізації,
- деструктор,
- функцію `show()` для виведення інформації про тип об'єкта цього класу та про значення його полів, – див. зразок виконання завдання;

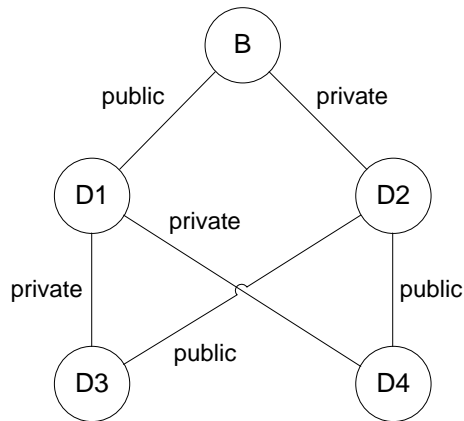
Функція `main()` має ілюструвати ієрархію успадкування – тобто, для всіх класів, вказаних у завданні:

- створити об'єкти,
- проініціалізувати всі поля створених об'єктів,

- вивести розмір кожного об'єкту,
- вивести результат функції `show()`.

Визначення кожного класу та реалізацію його методів слід розмістити в окремих модулях.

Нарисувати UML-діаграму класів та структурну схему (або UML-діаграму пакетів).



## Текст програми

**Зауваження.** При виконанні лабораторних завдань визначення кожного класу та реалізацію його методів слід розмістити в окремих модулях.

```

#include <iostream>
#include <windows.h>

using namespace std;

class B
{
 int b;

public:
 B()
 : b(0)
 { }

 B(int x)
 {
 b = x;
 }

 void show_B()
 {
 cout << "class B:" << endl;
 cout << "show_B()" << endl
 << "B::b = " << b << endl << endl;
 }
};

class D1 : public B
{
 int d1;

public:

```

```

D1(int x, int y)
 : B(y)
{
 d1 = x;
}

void show_D1()
{
 cout << "class D1:" << endl;
 show_B();
 cout << "show_D1()" << endl
 << "D1::d1 = " << d1 << endl << endl;
}
};

class D2 : private B {
 int d2;

public:
 D2(int x, int y)
 : B(y)
 {
 d2 = x;
 }

 void show_D2()
 {
 cout << "class D2:" << endl;
 show_B();
 cout << "show_D2()" << endl
 << "D2::d2 = " << d2 << endl << endl;
 }
};

class D3 : private D1, public D2
{
 int d3;

public:
 D3(int x, int y, int z, int i, int j)
 : D1(y, z), D2(i, j)
 {
 d3 = x;
 }

 void show_D3()
 {
 cout << "class D3:" << endl;
 show_D1();
 show_D2();
 cout << "show_D3()" << endl
 << "D3::d3 = " << d3 << endl << endl;
 }
};

class D4 : public D2, private D1
{
 int d4;

public:
 D4(int x, int y, int z, int i, int j)
 : D1(y, z), D2(i, j)

```

```

{
 d4 = x;
}

void show_D4()
{
 cout << "class D4:" << endl;
 show_D1();
 show_D2();
 cout << "show_D4()" << endl
 << "D4::d4 = " << d4 << endl << endl;
}
};

int main()
{
 SetConsoleCP(1251);
 SetConsoleOutputCP(1251);

 B o0(777);
 cout << "B o0(777);" << endl;
 cout << "sizeof(B) = " << sizeof(B) << endl;
 cout << endl << "Ієрархія класу B: " << endl;
 o0.show_B();

 D1 o1(111, 222);
 cout << "D1 o1(111, 222);" << endl;
 cout << "sizeof(D1) = " << sizeof(D1) << endl;
 cout << endl << "Ієрархія класу D1: " << endl;
 o1.show_D1();

 D2 o2(1000, 2000);
 cout << "D2 o2(1000, 2000);" << endl;
 cout << "sizeof(D2) = " << sizeof(D2) << endl;
 cout << endl << "Ієрархія класу D2: " << endl;
 o2.show_D2();

 D3 o3(100, 200, 300, 400, 500);
 cout << "D3 o3(100, 200, 300, 400, 500);" << endl;
 cout << "sizeof(D3) = " << sizeof(D3) << endl;
 cout << endl << "Ієрархія класу D3: " << endl;
 o3.show_D3();

 D4 o4(1, 2, 3, 4, 5);
 cout << "D4 o4(1, 2, 3, 4, 5);" << endl;
 cout << "sizeof(D4) = " << sizeof(D4) << endl;
 cout << endl << "Ієрархія класу D4: " << endl;
 o4.show_D4();

 return 0;
}

```

## Варіанти завдань

Побудувати ієрархію класів відповідно до схеми успадкування, заданої у варіанті завдання.

Кожен клас має містити:

- власне поле, ім'я якого – це ім'я класу, записане в нижньому регістрі (малими літерами), – див. зразок виконання завдання;
- конструктор ініціалізації,
- деструктор,
- функцію `show()` для виведення інформації про тип об'єкта цього класу та про значення його полів, – див. зразок виконання завдання;

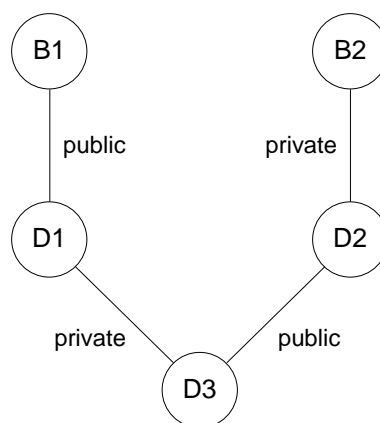
Функція `main()` має ілюструвати ієрархію успадкування – тобто, для всіх класів, вказаних у завданні:

- створити об'єкти,
- проініціалізувати всі поля створених об'єктів,
- вивести розмір кожного об'єкту,
- вивести результат функції `show()`.

Визначення кожного класу та реалізацію його методів слід розмістити в окремих модулях.

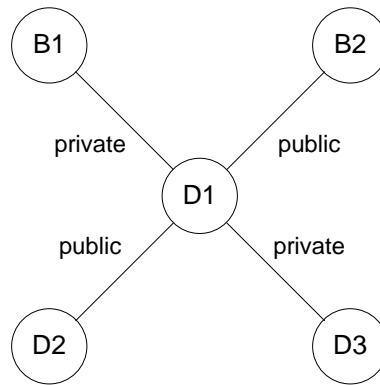
Нарисувати UML-діаграму класів та структурну схему (або UML-діаграму пакетів).

### Варіант 1.

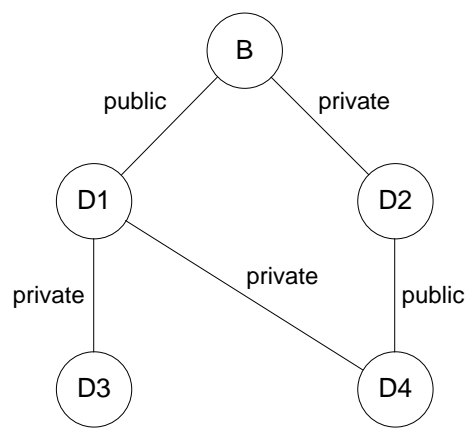




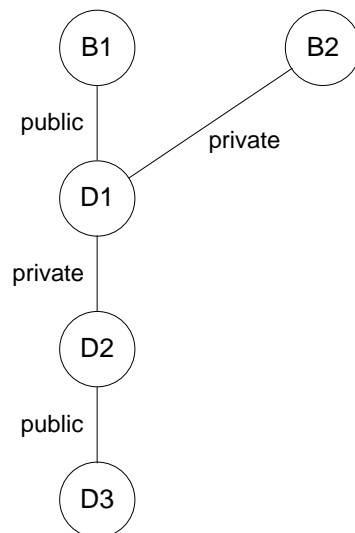
## Варіант 2.



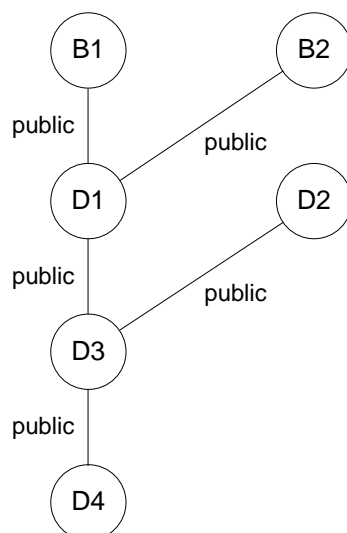
## Варіант 3.



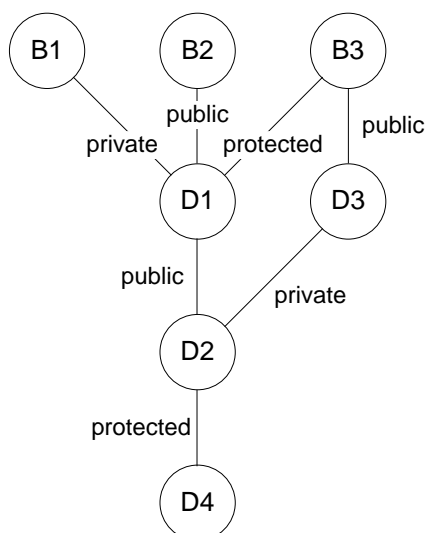
## Варіант 4.



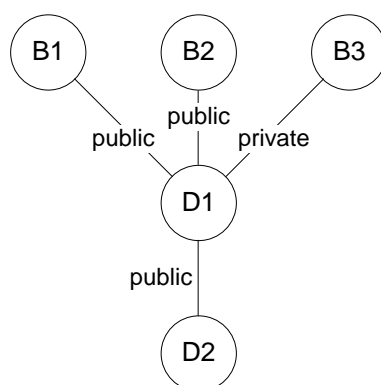
### Варіант 5.



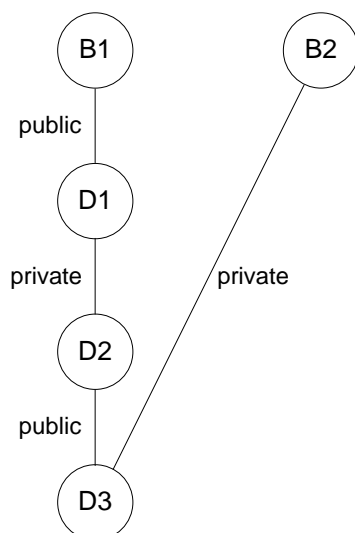
### Варіант 6.



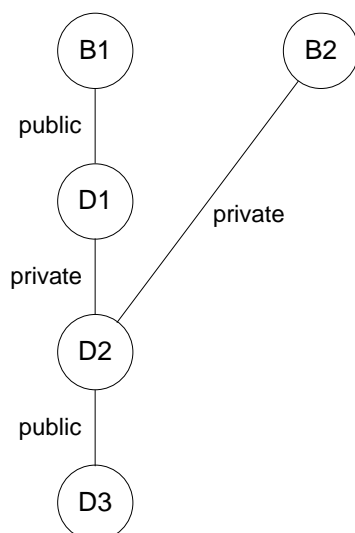
### Варіант 7.



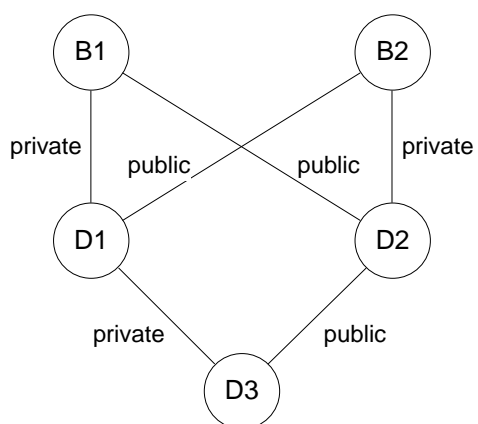
### Варіант 8.



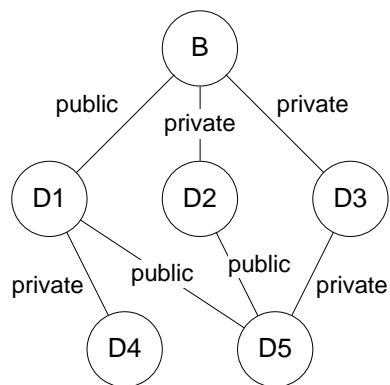
### Варіант 9.



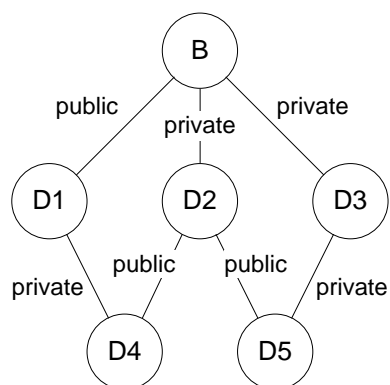
### Варіант 10.



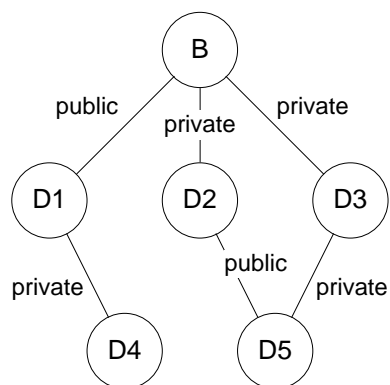
### Варіант 11.



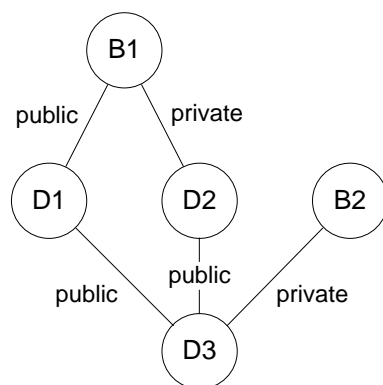
### Варіант 12.



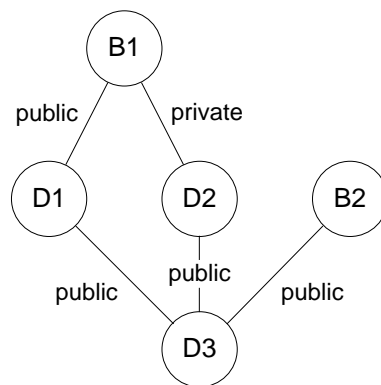
### Варіант 13.



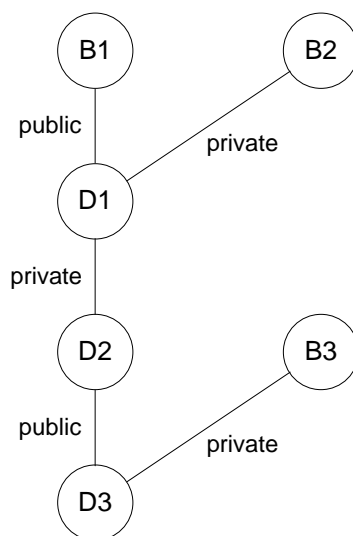
### Варіант 14.



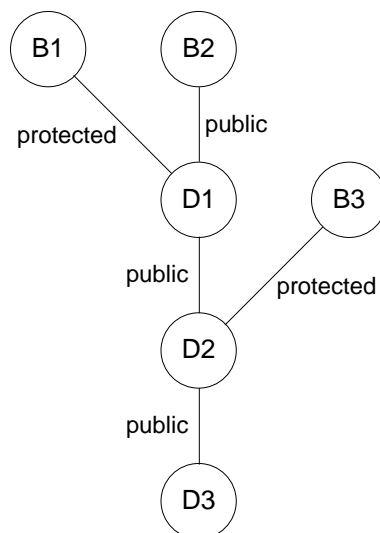
### Варіант 15.



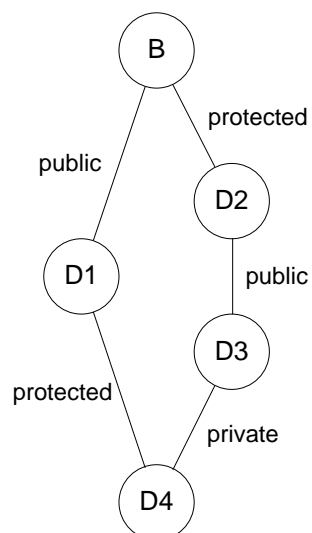
### Варіант 16.



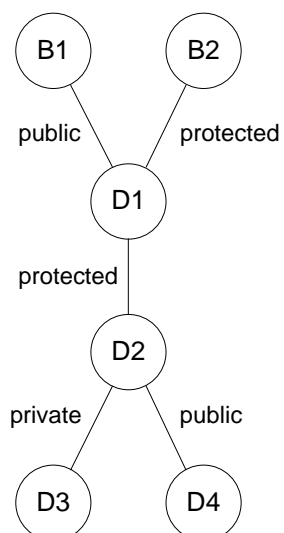
### Варіант 17.



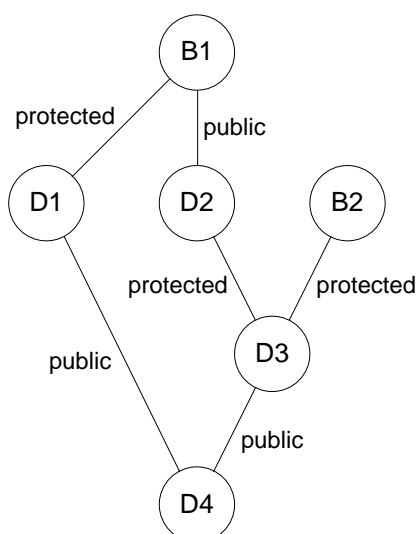
### Варіант 18.



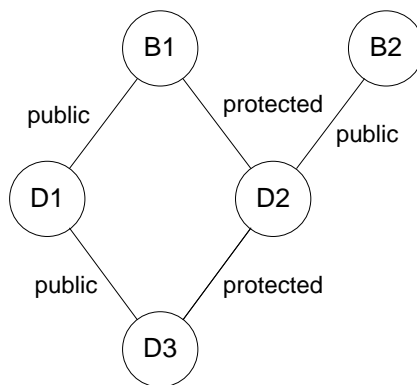
### Варіант 19.



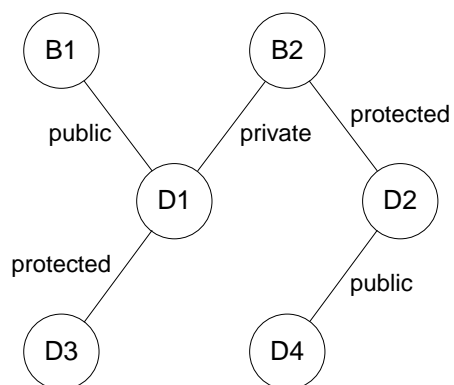
### Варіант 20.



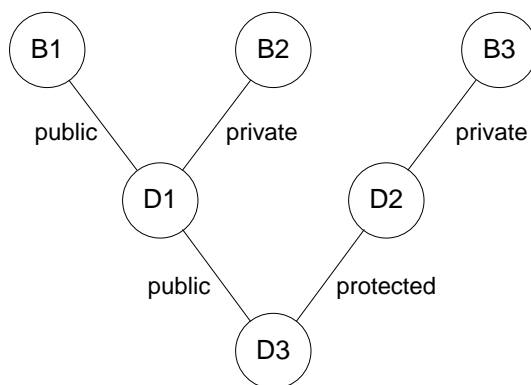
### Варіант 21.



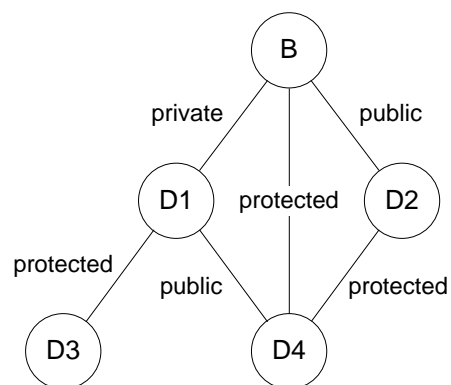
### Варіант 22.



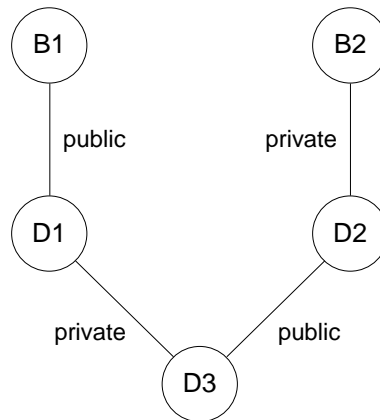
### Варіант 23.



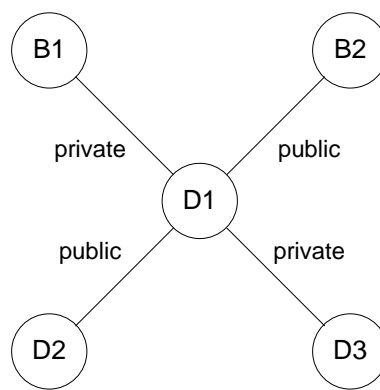
### Варіант 24.



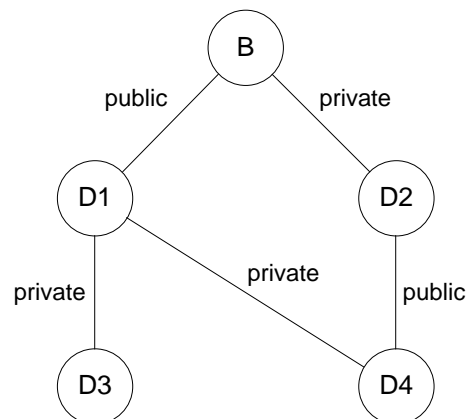
### Варіант 25.



### Варіант 26.

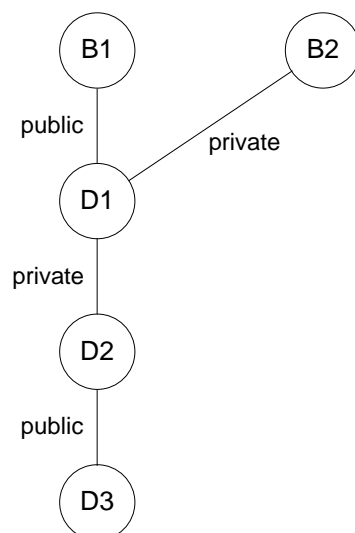


### Варіант 27.

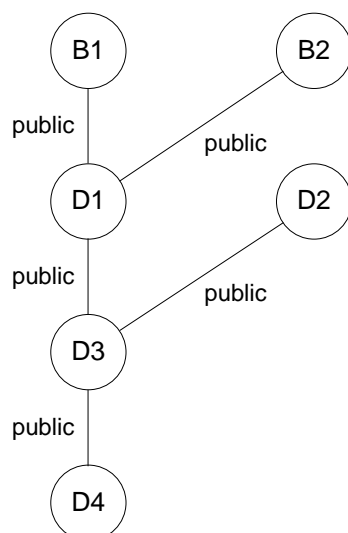




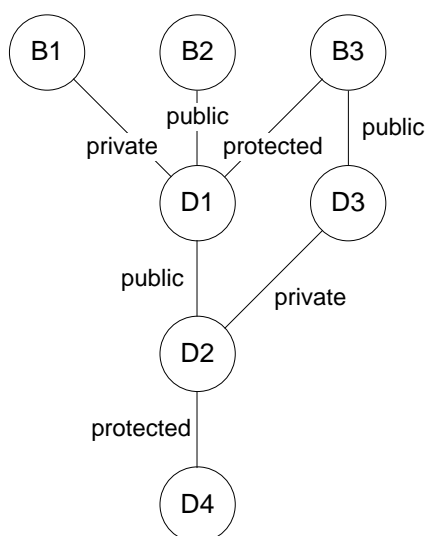
### Варіант 28.



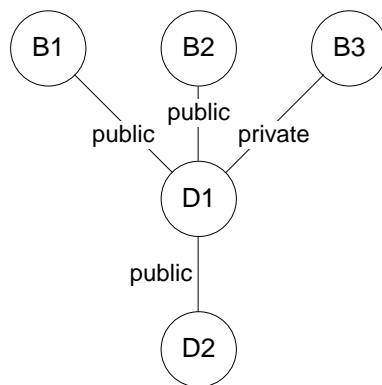
### Варіант 29.



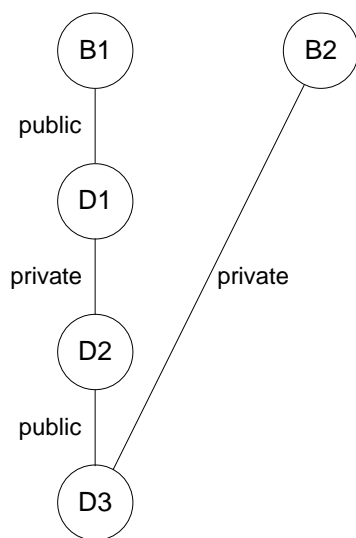
### Варіант 30.



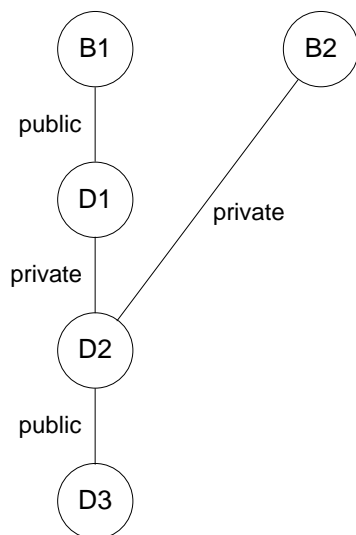
### Варіант 31.



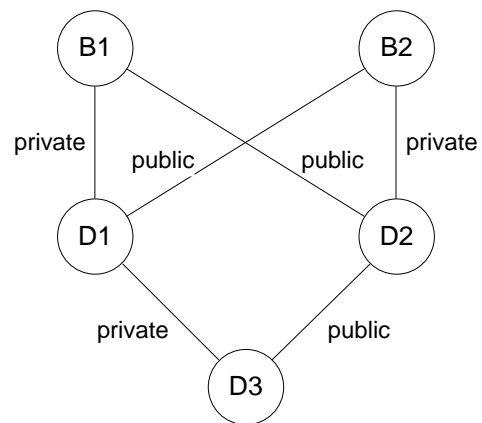
### Варіант 32.



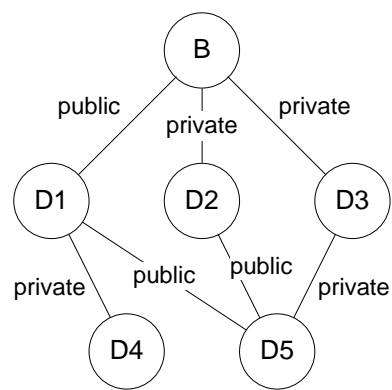
### Варіант 33.



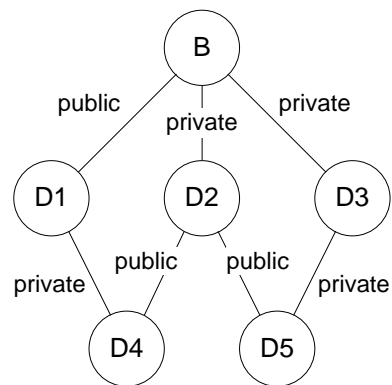
### Варіант 34.



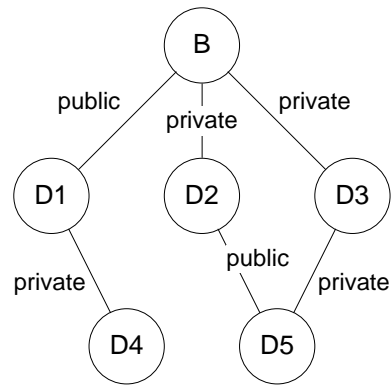
### Варіант 35.



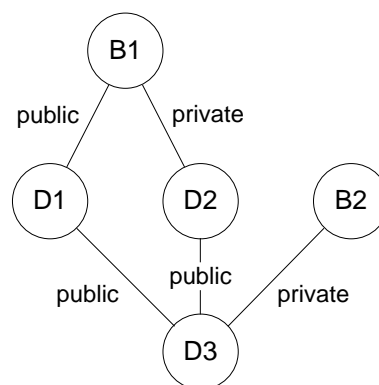
### Варіант 36.



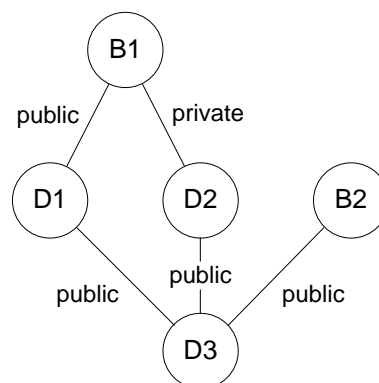
### Варіант 37.



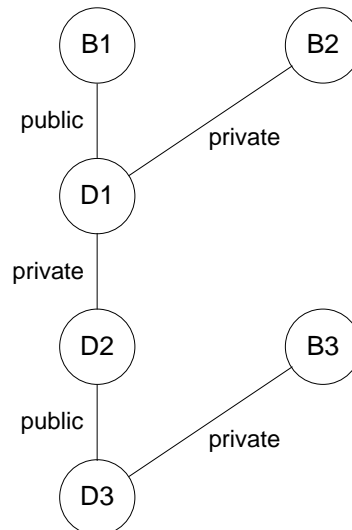
### Варіант 38.



### Варіант 39.



#### Варіант 40.



#### Варіант 41.

Створити ієрархію типів – файл для зчитування, файл для запису та файл для зчитування і запису. Класи мають мати конструктори, включаючи конструктор копіювання, віртуальні деструктори, перевантажені функції виведення в потік і введення з потоку.

#### Варіант 42.

Створити ієрархію типів, що описує програмне забезпечення та операційну систему. Визначити клас Windows NT як програмне забезпечення та операційну систему. Класи мають мати конструктори, включаючи конструктор копіювання, віртуальні деструктори, перевантажені функції виведення в потік і введення з потоку.

#### Варіант 43.

Створити ієрархію типів – автомобіль, пасажирський транспорт і автобус. Класи мають мати конструктори, включаючи конструктор копіювання, віртуальні деструктори, перевантажені функції виведення в потік і введення з потоку.

# Питання та завдання для контролю знань

## Просте успадкування

1. Які дві ролі виконує успадкування?
2. Які види успадкування можливі в C++?
3. Чим відрізняється модифікатор доступу `protected` від модифікаторів `private` і `public`?
4. Чим відкрите успадкування відрізняється від закритого і захищеного?
5. Чи може структура успадкувати від класу? А клас від структури?
6. Який тип успадкування від структури реалізується за умовчанням? А від класу?
7. У яких випадках в класі-нащадку недоступні елементи базового класу?
8. Які функції не успадковуються?
9. Сформулюйте правила написання конструкторів у похідному класі.
10. Який порядок виклику конструкторів? А деструкторів?
11. Якщо ім'я нового поля збігається з іменем успадкованого, то яким чином вирішити конфлікт імен?
12. Яким чином в конструкторі-нащадку викликати конструктор базового класу?
13. Що відбувається, якщо ім'я методу-нащадку збігається з іменем базового методу?
14. Яким чином в операції присвоєння класу-нащадка викликати операцію присвоєння базового класу?
15. Чи може вкладений клас успадковувати від зовнішнього? А зовнішній від вкладеного?
16. Сформулюйте принцип підстановки.
17. Коли виконується приведення типів, яке понижує? А приведення типів, яке підвищує?
18. Поясніть, що таке «зрізка», або «розщеплювання».
19. У яких випадках в класах використовується оголошення `using`?
20. Який вид успадкування «ближче» до композиції: відкрите чи закрите?
21. Дано визначення класів і об'єктів:

```
class C1 {
};

class C2: public C1 {
};

C1 a;
C2 b;

b = a;
```

Вказати і пояснити помилку.

## Множинне успадкування

22. Скільки класів можна використовувати в якості базових?
23. Чи виконується при множинному успадкуванні принцип підстановки?
24. Чи можна успадковувати закрито від одного класу, а відкрито від іншого?
25. В чому полягають проблеми при множинному успадкуванні?
26. Поясніть зміст віртуального успадкування.
27. Як формулюється принцип домінування?
28. Які функції відповідають за ініціалізацію віртуального базового класу?
29. Чи можна виконати приведення типу, яке понижує, за допомогою операції `dynamic_cast<>` ? А приведення, яке підвищує чи перехресне?
30. Чим відрізняється робота операції `dynamic_cast<>` при перетворенні вказівників і посилань?
31. Яку виняткову ситуацію генерує операція `dynamic_cast<>` і в яких випадках?
32. Дано визначення класів і об'єктів:

```
class A
{
 public:
 int a;
};

class B: public A
{
 public:
 int b;
};

class C: public A
{
 public:
 int c;
};

class D: public B, public C
{
 public:
 int d;
};

A oA;
B oB;
C oC;
D oD;
```

Нарисувати UML-діаграму класів.

Записати всі поля об'єктів oA, oB, oC, oD.

Якими будуть розміри об'єктів oA, oB, oC, oD ? (дані типу int займають 4 байти)

Виправити код – усунути дублювання полів при множинному успадкуванні (саме множинне успадкування слід залишити).

33. Дано визначення класів і об'єктів:

```
class A
{
 public:
 int a;
};

class B: public A
{
 public:
 int b;
};

class C: public A, public B
{
 public:
 int c;
};

class D: public A, public B, public C
{
 public:
 int d;
};

A oA;
B oB;
C oC;
D oD;
```

Нарисувати UML-діаграму класів.

Записати всі поля об'єктів oA, oB, oC, oD.

Якими будуть розміри об'єктів oA, oB, oC, oD ? (дані типу int займають 4 байти)

Виправити код – усунути дублювання полів при множинному успадкуванні (саме множинне успадкування слід залишити).

34. Дано визначення класів і об'єктів:

```
class A
{
 public:
 int a;
};

class B: public A
{
 public:
 int b;
};
```



```

class C: public B
{
 public:
 int c;
};

class D: public A, public B, public C
{
 public:
 int d;
};

A oA;
B oB;
C oC;
D oD;

```

Нарисувати UML-діаграму класів.

Записати всі поля об'єктів oA, oB, oC, oD.

Якими будуть розміри об'єктів oA, oB, oC, oD ? (дані типу int займають 4 байти)

Виправити код – усунути дублювання полів при множинному успадкуванні (саме множинне успадкування слід залишити).

35. Дано визначення класів:

```

class A
{
 public:
 int a;
};

class B: public A
{
 public:
 int b;
};

class C: public A, public B
{
 public:
 int c;
};

class D: public B, public C
{
 public:
 int d;
};

A oA;
B oB;
C oC;
D oD;

```

Нарисувати UML-діаграму класів.

Записати всі поля об'єктів oA, oB, oC, oD.

Якими будуть розміри об'єктів oA, oB, oC, oD ? (дані типу int займають 4 байти)

Виправити код – усунути дублювання полів при множинному успадкуванні (саме множинне успадкування слід залишити).

# Предметний покажчик

## Б

Базовий клас, 22, 35  
Батьківський клас, 22, 35

## Г

Генералізація, 22, 35

## Д

Делегування, 31, 61  
Деструктор при успадкуванні, 25  
Директиви доступу, 24, 40  
    private, 24, 40  
    protected, 24, 40  
    public, 24, 40  
Дочірній клас, 22, 35

## К

Клас-нащадок, 22, 35  
Клас-предок, 22, 35  
Ключ успадкування, 24, 40, 41  
    доступність успадкованих елементів, 41  
Конструктори при успадкуванні, 25

## О

Об'єднання, 69

## П

Патерн Adapter  
    адаптер класу, 73  
    адаптер об'єктів, 63  
Перетворення типів  
    const cast, 80  
    dynamic cast, 82  
        вгору, 82  
        вниз, 83  
    перетворення посилань, 84  
    перехресне, 85  
    reinterpret cast, 81  
    static cast, 81, 87  
Похідний клас, 22, 35  
Принцип підстановки, 28, 56  
    зрізування, 59  
    зрізування та розщеплення, 29  
    розщеплення, 59

## С

Структури, 69

## У

Успадкування, 22, 35  
    виклик методів, 51  
    відкрите, 60  
    віртуальне, 74  
        фінальний клас, 78  
    вкладені класи, 28, 55  
    власні методи, 51  
    глибина, 23, 36  
    деструктор, 27, 46, 48  
    дружні функції, 30, 57, 59  
    закрите, 30, 33, 60, 62  
        принцип підстановки, 33  
    ієрархія, 23, 36  
    інтерфейсу, 60  
    кількість рівнів, 23, 36  
    конструктори, 27, 45, 48  
    методи, 25, 27, 48  
        виклик методу, 27  
        власні методи, 27  
        однойменні, 28, 51  
        приховування базового методу, 28, 51  
        перевантаження, 27, 51  
    множинне, 33, 69  
        домінування принцип, 77  
        неоднозначність, 70  
        даних, 70  
        методів, 71  
        ромбовидне успадкування, 72  
        усунення неоднозначності даних, 74  
    операції базового класу, 53  
    операція присвоєння, 27, 29, 48, 57  
        базового класу, 27, 49  
    перевантаження функцій, 51  
    позначення на UML-діаграмах класів, 22, 35  
    поля, 25, 47  
        однойменні, 26  
    поняття, 35  
    проекткування ієрархії класів, 37  
    просте відкрите, 24, 30, 45  
    просте успадкування, 23, 40  
    реалізації, 60, 62  
    статичні елементи класу, 55  
    статичні методи, 28, 55  
    статичні поля, 28, 54  
    структури, 33  
Успадкування інтерфейсу, 30  
Успадкування реалізації, 30, 33  
Успадкування та композиція, 32, 62

# Література

## Основна

1. Павловская Т.А. С/С++. Программирование на языке высокого уровня СПб.: Питер, 2007. – 461 с.
2. Павловская Т.А., Щупак Ю.А. С/С++. Объектно-ориентированное программирование: Практикум СПб.: Питер, 2005. – 265 с.
3. Дейтел Х.М., Дейтел П.Дж. Как программировать на С++ М.: Бином-Пресс, 2005. – 1248 с.
4. Уэллин С. Как не надо программировать на С++ СПб.: Питер, 2004. – 240 с.
5. Хортон А. Visual C++ 2005: Базовый курс М.: Вильямс, 2007. – 1152 с.
6. Солтер Н.А., Клеппер С.Дж. С++ для профессионалов. М.: Вильямс, 2006. – 912 с.
7. Лафоре Р. Объектно-ориентированное программирование в С++ СПб.: Питер, 2006. – 928 с.
8. Лаптев В.В. С++. Объектно-ориентированное программирование СПб.: Питер, 2008. – 464 с.
9. Лаптев В.В., Морозов А.В., Бокова А.В. С++. Объектно-ориентированное программирование. Задачи и упражнения СПб.: Питер, 2008. – 464 с.

## Додаткова

10. Прата С. Язык программирования С++. Лекции и упражнения СПб.: ДиаСофт, 2003. – 1104 с.
11. Мейн М., Савитч У. Структуры данных и другие объекты в С++ М.: Вильямс, 2002. – 832 с.
12. Саттер Г. Решение сложных задач на С++ М.: Вильямс, 2003. – 400 с.
13. Чепмен Д. Освой самостоятельно Visual C++ .NET за 21 день М.: Вильямс, 2002. – 720 с.
14. Мартынов Н.Н. Программирование для Windows на С/С++ М.: Бином-Пресс, 2004. – 528 с.
15. Паппас К., Мюррей У. Эффективная работа: Visual C++ .NET СПб.: Питер, 2002. – 816 с. М.: Вильямс, 2001. – 832 с.
16. Грэхем И. Объектно-ориентированные методы. Принципы и практика М.: Вильямс, 2004. – 880 с.
17. Элиенс А. Принципы объектно-ориентированной разработки программ М.: Вильямс, 2002. – 496 с.

18. Ларман К. Применение UML и шаблонов проектирования М.: Вильямс, 2002. – 624 с.
19. Шилэт Г. Полный справочник по С. 4-е издание. М.-СПб.-К: Вильямс, 2002.
20. Прата С. Язык программирования С. Лекции и упражнения. М.: ДиаСофтЮП, 2002.
21. Александреску А. Современное проектирование на С++ М.: Вильямс, 2002.
22. Браунси К. Основные концепции структур данных и реализация в С++ М.: Вильямс, 2002.
23. Подбельский В.В. Язык СИ++. Учебное пособие. М.: Финансы и статистика, 2003.
24. Павловская Т.А., Щупак Ю.А. С/С++. Программирование на языке высокого уровня. СПб.: Питер, 2002.
25. Савитч У. Язык С++. Объектно-ориентированного программирования. М.-СПб.-К.: Вильямс, 2001.
26. Страуструп Б. Дизайн и эволюция С++: Пер. с англ. – М.: ДМК Пресс; СПб.: Питер, 2006.