Άσκηση 1η: Πρόγραμμα Histogram

Σημαντική σημείωση: Το πρόγραμμα που δημιουργήσαμε δοκιμάστηκε στους compilers των εξής γνωστών IDE’s: CLion, VSCode και Atom, καθώς και σε μια πληθώρα από αξιόπιστα online compilers που παρέχει μηχανή αναζήτησης του Google. Το πρόγραμμα σε όλες αυτές τις περιπτώσεις έτρεξε κανονικά, χωρίς οποιοδήποτε σφάλμα και παρείχε ορθά αποτελέσματα. Ωστόσο, κατά την εκτέλεση του εκτελέσιμου αρχείου που δημιουργήθηκε από τον gnu compiler, το πρόγραμμα τερματίζει εφόσον λάβει τον αριθμό τον στοιχείων που θέλουμε να εισαχθούν στη συνδεδεμένη λίστα, για λόγο που παρόλη την προσπάθεια μας δεν μπορέσαμε να καθορίσουμε. Εικάζουμε, ωστόσο, ότι πιθανώς να έχει να κάνει είτε (α) με το ότι η γεννήτρια τυχαίων αριθμών επεξεργάζεται των χώρο της μνήμης cache με τρόπο που το λειτουργικό σύστημα δεν επιτρέπει είτε (β) με τις βιβλιοθήκες που χρησιμοποιούνται για την γεννήτρια τυχαίων αριθμών, τις οποίες ενδεχομένως ο gnu compiler να μην υποστηρίξει (είτε λόγω παλαιότερης/νεότερης έκδοσης, ή επειδή έχουν διαφθαρεί, ή επειδή απαιτούν άδειες πρόσβασης πχ στην cache που το λειτουργικό δεν επιτρέπει). Ωστόσο εμείς αποφασίσαμε συνειδητά να παρέχουμε μαζί και το εκτελέσιμο, για λόγους πληρότητας και τυπικότητας. Θα συστήναμε το τρέξιμο του προγράμματος σε έναν Online C++ Compiler όπως τους OnlineGDB, Programiz, C++ Shel και Replit, για μεγαλύτερη ευκολία, ως απόδειξη ότι το πρόβλημα εμφανίζεται μόνο με το τρέξιμο του εκτελέσιμου στον υπολογιστή και όχι οπουδήποτε αλλού. Να σημειωθεί ότι το πρόβλημα αυτό ΔΕΝ ΥΦΊΣΤΑΤΑΙ στο 2ο πρόγραμμα, αυτό με τους σωρούς μεγίστων και ελαχίστων, του οποίου το εκτελέσιμο τρέχει κανονικά.

Στις 3 πρώτες γραμμές κώδικα γίνεται η συμπερίληψη των κατάλληλων header files που θα χρειαστούμε, το Random και το functional για την λειτουργία της γεννήτριας τυχαίων αριθμών και το iostream για το standard Output και Input στο πρόγραμμα. Έπειτα δηλώνουμε ότι χρησιμοποιούμε το namespace std για μεγαλύτερη ευκολία και ταχύτητα στη συγγραφή του κώδικα.

Στις επόμενες 4 γραμμές ακολουθεί η δήλωση των κλάσεων για την Chain και για την Histogram, φυσικά ως templated κλάσεις, πριν από την υλοποίησή τους, ώστε να μην υπάρξει λάθος στο compile time, το οποίο θα έλεγε πως οι κλάσεις ChainNode και HistogramNode είναι φίλοι κλάσεων που δεν υπάρχουν, μιας και ο compiler δεν θα της είχε ακόμα προσπελάσει.

Στις επόμενες 8 γραμμές υλοποιείται η κλάση των κόμβων για την Chain, με private δεδομένα ένα pointer προς τον επόμενο κόμβο (\*linkL) και τα δεδομένα του κάθε κόμβου (dataL). Η κλάση αυτή είναι φίλη (friend) με τις κλάσεις Chain και Histogram, έτσι ώστε τα private αυτά δεδομένα που αναφέραμε να μπορούν να προσπελαστούν και στις 2 αυτές άλλες κλάσεις. Αντίστοιχα, στις επόμενες 9 γραμμές υλοποιείται η κλάση των κόμβων της Histogram, με τη διαφορά ότι έχει ακόμα ένα private δεδομένο, το countH, όπου αποθηκεύουμε πόσες φορές εμφανίζεται το κάθε στοιχείο στο Histogram. Επίσης δεν χρειάζεται να οριστεί ως φίλος της η κλάση Chain, αφού τα δεδομένα του HistogramNode δεν μας χρειάζονται στην Chain.

Στις 100 περίπου επόμενες γραμμές κώδικα, ακολουθεί η υλοποίηση της templated κλάσης Histogram.

Στο public κομμάτι της, υλοποιούνται αρχικά ο Constructor της με τον Pointer firstH που δείχνει στο 1ο στοιχείο της αλυσίδας να αρχικοποιήται με 0 και έπειτα ο Destructor της, για την δυναμική αποδέσμευση μνήμης. Ορίζεται νέος κόμβος \*next και με τη βοήθεια του \*first που δείχνει κάθε φορά στο πρώτο στοιχείο της αλυσίδας, διαγράφουμε έναν προς έναν τους κόμβους της, έως ότου το \*first να είναι Null, δηλαδή να μην υπάρχουν άλλα στοιχεία στην αλυσίδα.

Στις επόμενες 40 και πλέον γραμμές ακολουθεί ίσως το σημαντικότερο κομμάτι του προγράμματος, η μέθοδος Insert της κλάσης Histogram, όπου κατασκευάζεται, από τα στοιχεία της L το αντικείμενο της Histogram, όπως περιγράφεται στις προδιαγραφές του προγράμματος. Αναλυτικότερα, ορίζουμε δύο νέα pointers κατάλληλου τύπου, ένα να δείχνει στο 1ο στοιχείο του συγκεκριμένου αντικειμένου Chain L που επεξεργαζόμαστε και το άλλο να δείχνει στο 1ο στοιχείο του αντικειμένου του Histogram. Με το αρχικό while loop, που τρέχει όσο το currentL δείχνει σε στοιχείο της L και όχι Null, τσεκάρουμε για το κάθε στοιχείο της αν υπάρχει στο Histogram. Μέσα σε αυτό το loop, ερωτόμαστε αν currentH είναι Null, δηλαδή αν το αντικείμενο του Histogram είναι κενό, χωρίς στοιχείο. Τότε, αυτό σημαίνει ότι ετοιμαζόμαστε να εισάγουμε το 1ο στοιχείο του Histogram, άρα το countH του θα είναι 1 και το firstH (αλλά και το current) θα δείχνει σε αυτό. Δίνουμε στο πεδίο dataH του την τιμή του αντίστοιχου πεδίου της L. Διαφορετικά, αν δεν πρόκειται για το 1ο στοιχείο του Histogram, τότε αρχικά διατρέχουμε ολόκληρο το αντικείμενο του Histogram, ως έχει δημιουργηθεί μέχρι τώρα και αν βρούμε το τρέχων στοιχείο της L, τότε αυξάνουμε τον μετρητή του στοιχείου του Histogram όπου το βρήκαμε, ενώ αν δεν το βρούμε αυξάνουμε τον μετρητή i. Έπειτα καλούμε τη μέθοδο Length, την οποία αναλύουμε παρακάτω, για το τρέχων αντικείμενο του Histogram και με όσα στοιχεία αυτό έχει μέσα του στην παρούσα επανάληψη κάθε φορά και το αποτέλεσμά της αποθηκεύουμε στη μεταβλητή size. Συγκρίνουμε τότε το size με τη μεταβλητή I, που λειτουργεί ως counter, όπως αναφέραμε. Αν είναι ίσα τότε αυτό σημαίνει ότι το τρέχων στοιχείο της L δεν υπάρχει στο Histogram και για αυτό τώρα θα το εισάγουμε σε αυτό. Ορίζουμε αρχικά νέο pointer y τύπου HistogramNode<T> και θέτουμε στο dataH του την τιμή του dataL, στο countH το 1, αφού το στοιχείο κάνει την 1η του εμφάνιση και επιστρέφουμε το currentH στο firstH για το επόμενο loop. Στο επόμενο, λοιπόν, while loop όσο το counter c είναι μικρότερο του Histogram με τα στοιχεία που έχει τώρα (size) τότε προχωράμε μέσω του linkH στο επόμενό του στοιχείο, μέχρις ότου να είναι ίσα, δηλαδή να βρισκόμαστε στο τελευταίο στοιχείο του Histogram, μετά από το οποίο θα εισάγουμε το νέο στοιχείο y. Εφόσον όλες αυτές οι διαδικασίες έχουν εκτελεστεί, προχωράμε στο επόμενο στοιχείο της αλυσίδας L, μέχρι να μην υπάρχουν άλλα. Στο τέλος της μεθόδου επιστρέφουμε pointer προς αυτό το αντικείμενο, όπως έχει δημιουργηθεί πλέον μέσα από την παραπάνω διαδικασία. Αυτό, φυσικά, συμφωνεί με τον τύπο που έχουμε δώσει στη μέθοδο αυτή κατά τη δήλωση της, ο οποίος είναι Histogram<T>&.

Στις επόμενες 9 γραμμές του κώδικα έχει υλοποιηθεί η μέθοδος Length(), τύπου int αφού επιστρέφει το μήκος του Histogram και const αφού απλά προσπελαύνει τα στοιχεία του, δίχως να τα επεξεργάζεται. Με τη βοήθεια του δείκτη current που δείχνει την πρώτη φορά στο 1ο στοιχείο του Histogram, θα προσθέτει 1 στον μετρητή len για κάθε του στοιχείο, μέχρις ότου να είναι πλέον Null. Η τελική τιμή του len είναι και αυτή που θα επιστραφεί.

Έπειτα έχει υλοποιηθεί η συνάρτηση ταξινόμησης του Histogram, με βασική της ιδέα να είναι ο αλγόριθμος της Bubble Sort, προσαρμοσμένος στην υλοποίηση με pointers και templates των μονά συνδεδεμένων λιστών. Η μέθοδος αυτή καλείται εφόσον έχουν εισαχθεί στο Histogram τα στοιχεία της λίστας L και καλούμαστε να τα ταξινομήσουμε. Δημιουργούνται αρχικά 2 temporary pointers τύπου HistogramNode<T> που θα χρησιμοποιηθούν εσωτερικά στις αντιμεταθέσεις των στοιχείων του Histogram. Τα for loops έχουν ως εξής: Στο 1ο loop το temp2 ξεκινάει από το 1ο στοιχείο του Histogram και επαναλαμβάνει όσο δείχνει σε στοιχείο του και όχι Null. Διασχίζει τα στοιχεία ένα-ένα. Στο 2ο for loop το temp2 ξεκινάει από την τρέχουσα τιμή του temp1, ενώ συνεχίζει όσο δεν είναι Null, δηλαδή όσο δείχνει σε στοιχείο. Σε κάθε επανάληψη ακολουθεί το linkH, προς το επόμενο στοιχείο από αυτό στο οποίο δείχνει. Έπειτα με τη δομή επιλογής ελέγχουμε αν το πεδίο dataH του αριστερού στοιχείου (αριστερού σχετικά με τα δύο στοιχεία που συγκρίνουμε) είναι μικρότερο από το στοιχείο που βρίσκεται δεξιά του και αν ναι τότε ακολουθεί ο γνωστός αλγόριθμος αντιμετάθεσης δύο στοιχείων, μία φορά για το πεδίο των δεδομένων τους (data) και μία φορά για τα πεδία των μετρητών τους (counts). Τέλος, επιστρέφει το τρέχων αντικείμενο ταξινομημένο στην main με pointer, όπως φαίνεται και στον τύπο της, στη δήλωσή της, που είναι Histogram<T>&.

Στις τελευταίες γραμμές κώδικα της κλάσης έχουμε υλοποιήσει την μέθοδο εξόδου του Histogram στην οθόνη, ως μια void μέθοδο, μιας και δεν επιστρέφει κάτι στην main και const γιατί δεν επεξεργάζεται το αντικείμενο, πέρα από το να το εκτυπώσει. Ορίζεται λοιπόν δείκτης current τύπου HistogramNode<T> και αρχικοποιήται με το περιεχόμενο του δείκτη firstH, που δείχνει στο 1ο στοιχείο του Histogram. Όσο το current δεν είναι Null η μέθοδος εκτυπώνει τα πεδία dataH και countH καθενός στοιχείου από το Histogram, πηγαίνοντας κάθε φορά στο επόμενο, μέσω του δείκτη linkH. Λίγο πριν τη λήξη της κλάσης έχουμε το private μέρος της κλάσης, το οποίο περιέχει τους δείκτες HistogramNode<T> \*firstH που επεξεργάζεται αντικείμενα της Histogram και τον ChainNode<T> \*firstL που επεξεργάζεται την L. Ο 2ος μας είναι και αυτός απαραίτητος, γιατί, όπως αναφέραμε και πριν, η συνδεδεμένη λίστα L πρέπει να υποβληθεί σε επεξεργασία στην μέθοδο της τρέχουσας κλάσης, Insert.

Έπειτα από την Histogram και για 50 περίπου γραμμές ακολουθεί η υλοποίηση της κλάσης Chain, που είναι αρκετά απλούστερη σε επίπεδο μεθόδων από την Histogram. Αρχικά λοιπόν δηλώνεται πως είναι φίλη με την Histogram, ώστε η Histogram να έχει πρόσβαση στα private κομμάτια της Chain. Μετά από αυτό υλοποιούνται το Constructor και το Destructor της κλάσης, με τρόπο ίδιο με αυτόν της Histogram, ώστε να γίνεται η δημιουργία των αλυσίδων-αντικειμένων της κλάσης και έπειτα η δυναμική αποδέσμευση χώρου με την καταστροφή τους. Το βασικό κομμάτι της κλάσης είναι η υλοποίηση της συνάρτησης Insert, για την εισαγωγή νέων στοιχείων σε μια συνδεδεμένη λίστα, δηλαδή ένα αντικείμενό της. Δέχεται, λοιπόν, σαν ορίσματα έναν ακέραιο k που χρησιμεύει ως τη θέση έπειτα από την οποία θα εισαχθεί το στοιχείο και ένα templated σταθερό x, που είναι το dataL πεδίο του νέου στοιχείου της λίστας, το οποίο δεν επεξεργαζόμαστε, απλά αναθέτουμε σε κάποια άλλη μεταβλητή. Έτσι, αν το K είναι μικρότερο του μηδενός κάνουμε throw ένα μήνυμα λάθους, που μας πληροφορεί ότι το k είναι εκτός των ορίων της λίστας. Εάν δεν συμβεί αυτό, τότε ορίζουμε νέο templated pointer κόμβο τύπου ChainNode, των αρχικοποιούμε με το πρώτο στοιχείο της λίστας firstL και με αυτόν διατρέχουμε τη λίστα μέχρις ότου να βρισκόμαστε στη ζητούμενη θέση. Αν αυτή ξεπερνάει τον μέγεθος της λίστας τότε κάνουμε throw ένα μήνυμα λάθους, που μας πληροφορεί ότι το k βρίσκεται εκτός ορίων της λίστας. Αν είναι μικρότερη ή ίση του μεγέθους της λίστας, τότε ορίζουμε τον κόμβο (y) που θα εκπροσωπήσει τον καινούργιο. An το κ-οστό στοιχείο υπάρχει, τότε εισάγουμε απλά στην επόμενη θέση το νέο στοιχείο και έπειτα προχωράμε στο επόμενο. Διαφορετικά το εισάγουμε, ουσιαστικά, ως 1ο στοιχείο, ολόκληρης της Chain και προχωράμε στο επόμενο. Στο τέλος της κλάσης έχουμε την μέθοδο εκτύπωσης της λίστας, ορισμένη και εδώ όπως και στην Histogram ως const για εξοικονόμηση χρόνου και υπολογιστικών πόρων, μιας και το αντικείμενο δεν δέχεται επεξεργασία, απλά εκτυπώνεται. Με τον ορισμό νέου pointer κόμβου τύπου ChainNode<T> που δείχνει στο 1ο στοιχείο του αντικειμένου και την διαδοχική προσπέλαση των στοιχείων του με την βοήθεια αυτού, τα elements της αλυσίδας L μπαίνουν στο ρεύμα cout του standard namespace και εκτυπώνονται στην οθόνη. Φυσικά, υπάρχει και το σύντομο private κομμάτι της κλάσης που περιέχει μονάχα τη δήλωση του pointer firstL τύπου ChainNode<T>, ο οποίος θα δείχνει, κατά την εκτέλεση του προγράμματος στο 1ο στοιχείο της μονά συνδεδεμένης λίστας L.

Στις τελευταίες 30 γραμμές του προγράμματος υλοποιείται η main και γίνονται οι βασικές διαδικασίες του προγράμματος, μέσα σε μια δομή try-catch για εντοπισμό τυχών runtime σφαλμάτων, ως εξής: Πρώτα γίνεται η ενδεδειγμένη υλοποίηση της γεννήτριας τυχαίων αριθμών και έπειτα ζητάται από τον χρήστη το μέγεθος της Chain, μέσα σε έλεγχο εγκυρότητας τιμών, ώστε να βεβαιωθούμε ότι ο αριθμός που θα δοθεί θα είναι θετικός. Έπειτα, βάση αυτών, γίνεται instantiation ενός object pointer τύπου Chain<int> και μέσα σε αυτό αποθηκεύονται τα κατάλληλα στοιχεία, βάση της random\_element() γεννήτριας, τα οποία δίνονται ως το 2ο όρισμα της μεθόδου Insert(), ενώ το 1ο όρισμα είναι το στοιχείο της συνδεδεμένης λίστας L, έπειτα από το οποίο θα εισαχθούν. Στη συνέχεια ακολουθούν το instantiation του αντίστοιχου object pointer τύπου Histogram<int>, η εισαγωγή των στοιχείων μέσα σε αυτό από τα στοιχεία του Object τύπου Chain, η εμφάνιση αυτών πριν την ταξινόμηση, η ταξινόμησή τους, η εμφάνιση της αρχικής αλυσίδας L και τελικά η εμφάνισή της Histogram, ταξινομημένης πλέον. Τέλος γίνεται δυναμική αποδέσμευση μνήμης. Αξίζει να αναφερθεί ότι η κλήση των μεθόδων που αφορούν την Chain και την Histogram γίνονται με τη χρήση του ¨->¨ , μιας και πρόκειται για pointer objects Εάν εντοπιστεί κάποιο runtime λάθος, τότε εμφανίζεται ένα μήνυμα λάθους, το οποίο έχει αποθηκευτεί από κάποιο από τα προηγούμενα throws που υπάρχουν στο πρόγραμμα και το οποίο εμφανίζεται μέσω του pointer msg (ο οποίος δεν αλλάζει μέσα στο catch block και για αυτό έχει δηλωθεί ως const), με το cerr. Αξίζει επίσης να σημειωθεί ότι όταν αρχικοποιούμε ένα αντικείμενο αντί για τον τύπο του χρησιμοποιούμε το keyword «auto», οπότε και ο τύπος του αντικειμένου καθορίζεται αποκλειστικά από το δεξί μέρος της εντολής δήλωσης και αρχικοποίησής του (το initializer του) και έτσι δεν χρειάζεται να ανησυχούμε για αλλαγή τύπου δεδομένων ή αναπάντεχα λάθη κατά την πληκτρολόγηση, κάνοντας το πρόγραμμά μας πιο εύκολο στη συντήρηση και τη διόρθωση.

Παναγιώτης Λειβαδάρος Π21085 και Νικόλαος Παπανικολάου Π21130