

**ATOM-N 2022**

AUGUST 25-28,  
CONSTANTA,  
ROMANIA

ADVANCED TOPICS IN OPTOELECTRONICS, MICROELECTRONICS AND NANOTECHNOLOGIES

**SPIE.**

CONNECTING MINDS.  
ADVANCING LIGHT.

# ***Class template – Primary Approach for the Flexible Dynamic Memory Allocation in the Hybrid Modelling Software Development***

**Emil M. Oanță<sup>a)</sup>, Alexandru Pescaru<sup>b)</sup>**

**a) Constanța Maritime University, General Sciences Department**

**b) Constanța Maritime University, Department of Navigation**

**Acknowledgement:**

***This work is inspired by the results of the ID1223 - "Computer Aided Advanced Studies in Applied Elasticity from an Interdisciplinary Perspective" Scientific Research Project, under the supervision of the National University Research Council (CNCSIS), Romania, 2007-2010.***

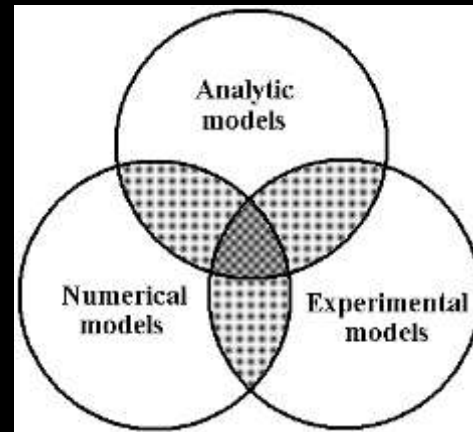
# Table of contents

- Introduction, example of a hybrid model
- Motivation
- Methodology
  - Level 1 – data structures for mathematical methods
  - Level 2 – optimization: access time minimization
  - Level 3 – library of header files; rapid development
- Results
  - Template classes' oriented development
  - Why not a general solution?
  - Some technical info and, of course, more questions
  - Type identification using a 'common use' compiler
  - T 'generic' type conversion
  - Operating the sample application
  - Some C++ Standard Template Library solutions
- Conclusion
- Acknowledgement
- References

# Introduction

## Actual conditions

- Complex phenomena;
- Huge volume of data;
- Multi-disciplinary approaches;
- Inter-domains influences;



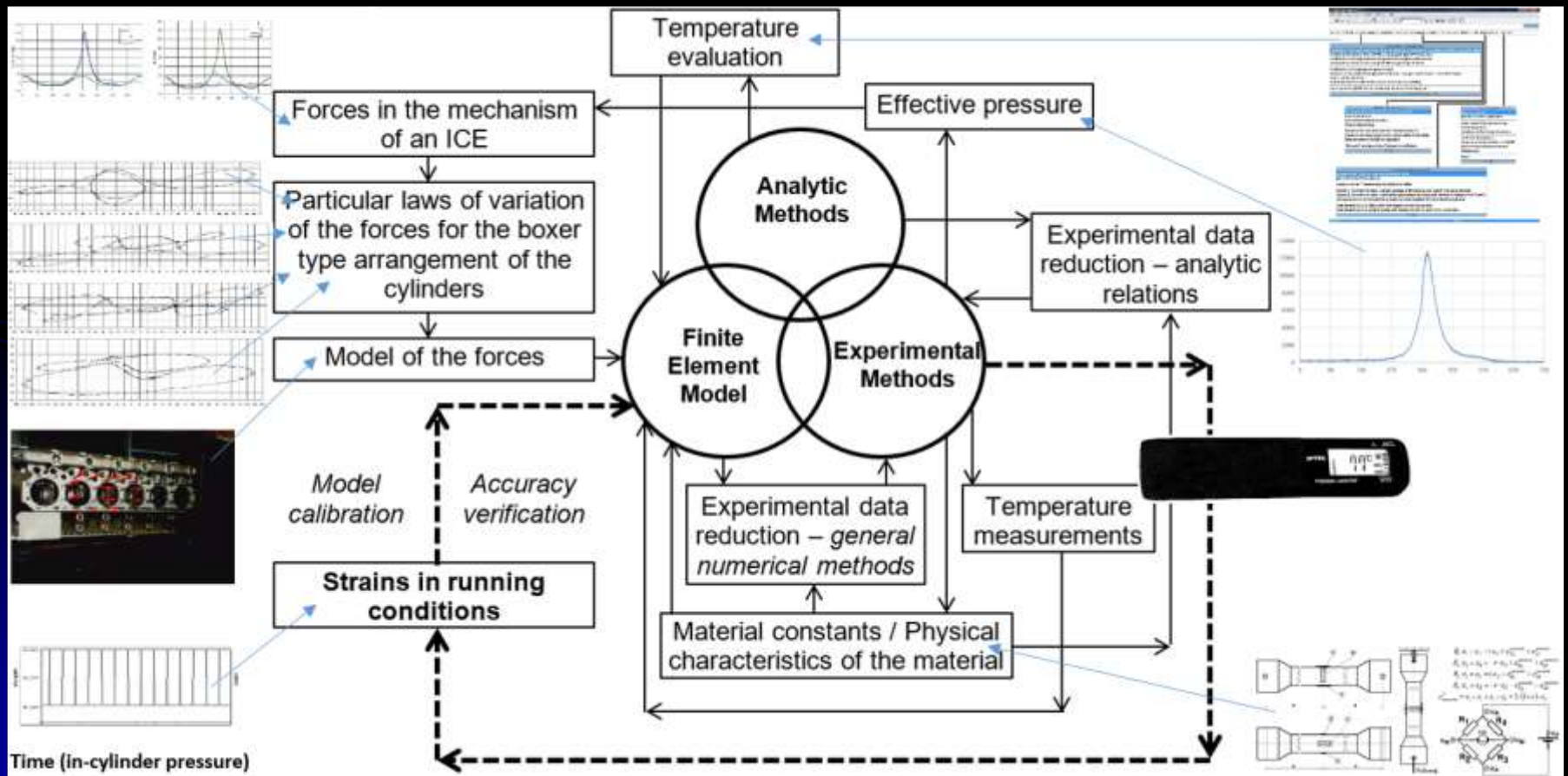
## New modelling paradigm → hybrid approaches

- Hybrid mode = 'building blocks' deeply connected;
- 'Building block' = study: theoretical model or experimental study;
- Theoretical approaches: analytical, numerical, semi-numerical etc.;
- **Paramount: connections between the 'building blocks';**
- All the 'building blocks' are implemented as **computer based solutions;**
- **Connections** = interfaces, i.e. **computer based solutions;**

# Example – Hybrid model

## An ICE's Structural model of the cylinder block:

- 12B165 – A navy vessel ICE;
- Experimental study → SAE2000, Detroit, USA;
- Running conditions;
- PhD, 'Cum laude', 88 letters / 105 reviewers;



# Motivation

**Key factor** → ***computer based solutions!***

## **Computers are used for distinct studies:**

- Analytical approaches → **original** software;
- General numerical methods → **original** & **commercial** software;
- Dedicated numerical methods (FEM) → **original** & **commercial** software;
- Semi-numerical methods → **original** & **commercial** software;
- Dedicated algorithms & solvers → **original** & **commercial** software;
- Decisional problems → **original** software.

## **Computers are used to connect the studies:**

- Based on the Application Program Interfaces → **original** software;
- Interfaces (CSV, JSON, original) → **original** software;
- Data integration → **original** software;
- Knowledge acquisition (using knowledge based systems) → **original** software;



# Methodology - Stages

## Level 1 – data structures for mathematical methods

The diagram illustrates the conversion of a matrix into a list structure. On the left, a matrix with elements  $a_{11}, a_{12}, \dots, a_{1M}$  in the first row and  $a_{N1}, a_{N2}, \dots, a_{NM}$  in the last row is shown. An arrow points to a vertical list structure where the elements are arranged in columns:  $a_{11}, a_{12}, \dots, a_{1M}$  in the first column,  $a_{21}, a_{22}, \dots, a_{2M}$  in the second, and  $a_{N1}, a_{N2}, \dots, a_{NM}$  in the last. To the right, a code block defines these structures. A green box highlights the definition of a list element (lines 00007-00013), with a green arrow pointing from the list structure to this box. A red box highlights the definition of an array of lists (line 00015), with a red arrow pointing from the list structure to this box. A blue box highlights the definition of a random access file's record (lines 00023-00034), with a blue arrow pointing from the list structure to this box. The code block contains the following definitions:

```
00001 Type
00002 Kod_Tip_Lista = 0..2; { Kod_Tip_Lista=0/1/2 Linear doubly linked list }
00003 Matrix_Value = Extended; { Kod_Tip_Lista=1 ==> Circular doubly linked list }
00004 Numar_Elemente = LongInt; { Kod_Tip_Lista=2 ==> Linear singly linked list }
00005 PRec = ^Inreg;
00006 Inreg = Record
00007   Anterior : PRec; { Address of the previous element in the list }
00008   X : Matrix_Value; { No of links between I, J: ABS(I-J) or I+N-J }
00009   Posterior : PRec; { Address of the following element in the list }
00010   Case_Tip_Lista : Kod_Tip_Lista Of
00011     1 : ( Index : Numar_Elemente; { Index of the element }
00012           Total_Ind : Numar_Elemente; { Total number of elements in the list }
00013           )
00014   End;
00015 Vector_Liste = Array [1..Nr_Max_Lin] Of PRec;
00016 Matrix = Array [1..Maxim_Number_Of_Rows; 1..Maxim_Number_Of_Columns] Of Matrix_Value;
00017 R_Matrix = Array [1..Reduced_Dimension, 1..Reduced_Dimension] Of Matrix_Value;
00018 Vector = Array [1..Maxim_Number_Of_Elements] Of Matrix_Value;
00019 IMatrix = Array [1..Maxim_Number_Of_Elements] Of Numar_Elemente;
00020 St_Vector = Array [1..Maxim_Number_Of_Elements] Of ExtStr;
00021 Strl = String[1];
00022 MsgStr = PathStr; {String[11];}
00023 Matrix_Record = Record
00024   Control_01 : Strl; { 002 - 002 * 001 character }
00025   Control_02 : Strl; { 003 - 003 * 001 character }
00026   Control_03 : Strl; { 004 - 004 * 001 character }
00027   Control_04 : Strl; { 005 - 005 * 001 character }
00028   Total_Lines : Numar_Elemente; { 006 - 009 * 004 characters }
00029   Total_Columns : Numar_Elemente; { 010 - 013 * 004 characters }
00030   Current_Line : Numar_Elemente; { 014 - 017 * 004 characters }
00031   Current_Column : Numar_Elemente; { 018 - 021 * 004 characters }
00032   Current_Value : Matrix_Value; { 022 - 052 * 031 characters }
00033   Current_Message : MsgStr; { 053 - 131 * 079 characters }
00034 End;
00035 Matrix_File_Type = File Of Matrix_Record;
00036 Vect_Integer_Type = Array [1..Maxim] Of LongInt;
00037 Vect_Position = Array [1..Maxim] Of LongInt;
00038 Vect_Integer = Array [1..Maxim] Of Numar_Elemente;
00039
00040 Var
00041   Var_Tip_Lista : Kod_Tip_Lista;
```

The definition of the lists' element

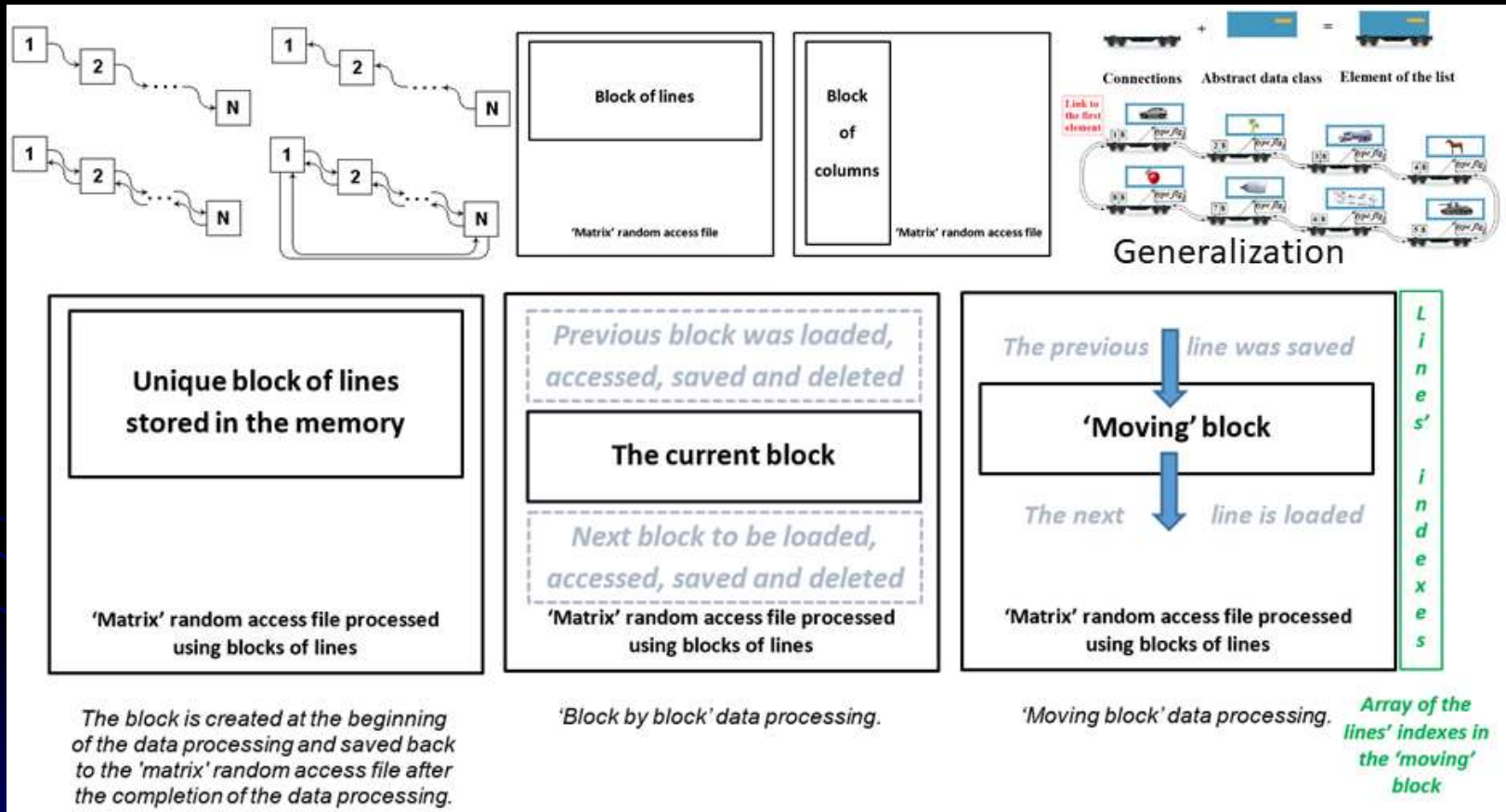
Array of lists

The definition of the random access file's record

- Matrix methods are ubiquitous in science;
- Large matrices → finer discretization → higher accuracy;

# Methodology - Stages

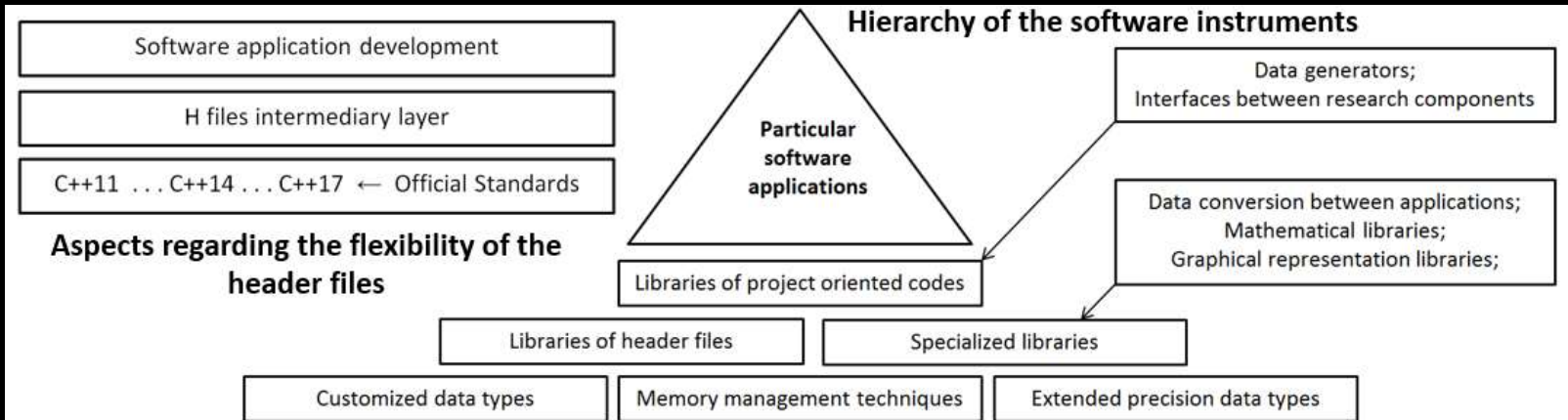
## Level 2 – optimization: access time minimization



- Data structures for dynamical memory allocation;
- Vector of doubly linked lists for matrices' block processing;

# Methodology - Stages

## Level 3 – library of header files; rapid development



*New: class oriented programming*  
**string\_funcs class**

```

Outline x Task List Build Targets
● string_funcs::setString(int) : void
● string_funcs::getString() : int
● string_funcs::setFString(float) : void
● string_funcs::getFString() : float
● string_funcs::setDString(double) : void
● string_funcs::getDString() : float
● string_funcs::str_pad(std::string, std::size_t, short int, std::string) : std::string
● string_funcs::str_trim(std::string) : std::string
● string_funcs::str_trimL(std::string) : std::string
● string_funcs::str_trim_all(std::string) : std::string
● string_funcs::str_extract_substr(std::string, std::size_t, std::size_t) : std::string
● string_funcs::str_int_to_string(int) : std::string
● string_funcs::str_string_to_int(std::string) : int
● string_funcs::str_replace_all_occurrences(std::string, std::string, std::string) : std::string
● string_funcs::str_to_uppercase(std::string) : std::string
● string_funcs::str_to_lowercase(std::string) : std::string
● string_funcs::str_ascii2string(std::size_t) : std::string
● string_funcs::str_char2ascii(std::string) : std::size_t
● string_funcs::strings_are_equal(std::string, std::string, std::size_t, std::size_t) : std::size_t
● string_funcs::read_from_file(std::string) : std::string
● string_funcs::write_to_file(std::string, std::string) : void
● string_funcs::file_exists(std::string, std::size_t) : std::size_t
● string_funcs::save_string_in_text_file(std::string, std::size_t, std::string) : void
    
```

A	B	C
1	ascii2string_char.h	get_external_exe_name_from_full_path.h
2	basic_types.h	integer2string.h
3	cancel.h	integer2string_and_pad.h
4	char_asterix_2_string.h	is_alphabetic.h
5	circular_segment_geometrical_characteristics.h	list_array_of_strings_in_a_text_file.h
6	create_folder.h	load_cfg.h
7	create_folder_using_OS_cmd.h	load_csv_file.h
8	create_parameters_computer_code.h	load_csv_of_strings.h
9	current_date2string.h	load_csv_of_strings_across_multiple_lines.h
10	current_time2string.h	load_parameters.h
11	display_CSV_ARRAY.h	load_parameters77.h
12	domains_basic_types.h	messagebox_info.h
13	domains_create_AutoCAD_script_files.h	messagebox_yes_no.h
14	domains_geometrical_characteristics_of_a_domain.h	no_of_occurrences.h
15	domains_report_info_4_a_set_of_domains.h	pause.h
16	extract_filename_without_extension.h	polygon_09_v.h
17	file_exists.h	polygon_csv_array.h
18	folder_exists.h	real2asciiStr.h
19	from_OS_path_2_system_command_path.h	real2integer.h
20	generate_generator.h	real2string.h
21	get_current_folder_separator.h	real2string_and
22	get_current_path.h	reals_are_equal.h
23	get_exe_path.h	real_abs.h
		removeCharacters.h
		removeSpaces.h
		replace_all_occurrences.h
		replace_extension_of_a_file.h
		save_string_in_text_file.h
		select_a_file.h
		select_multiple_files.h
		spline_CSV_ARRAY_give_values.h
		spline_derivative_give_values.h
		spline_give_values.h
		spline_integrals.h
		string2real.h
		string2index_type.h
		string2integer.h
		string2flag_type.h
		strings_are_equal.h
		string_char2ascii.h
		to_lowercase.h
		to_uppercase.h
		trim.h
		trim_right.h
		write_parameters.h

- header files are used as 'building blocks';
- reusability is paramount;
- OCTAVE may be connected with C++;
- API of the CAD/CAE commercial software;

Library of header files in C++



# Results

## Template classes' oriented development

- Classes: functions in *old* header files are included in *dedicated* classes;
- Template classes: types are 'generic' ← high reusability;

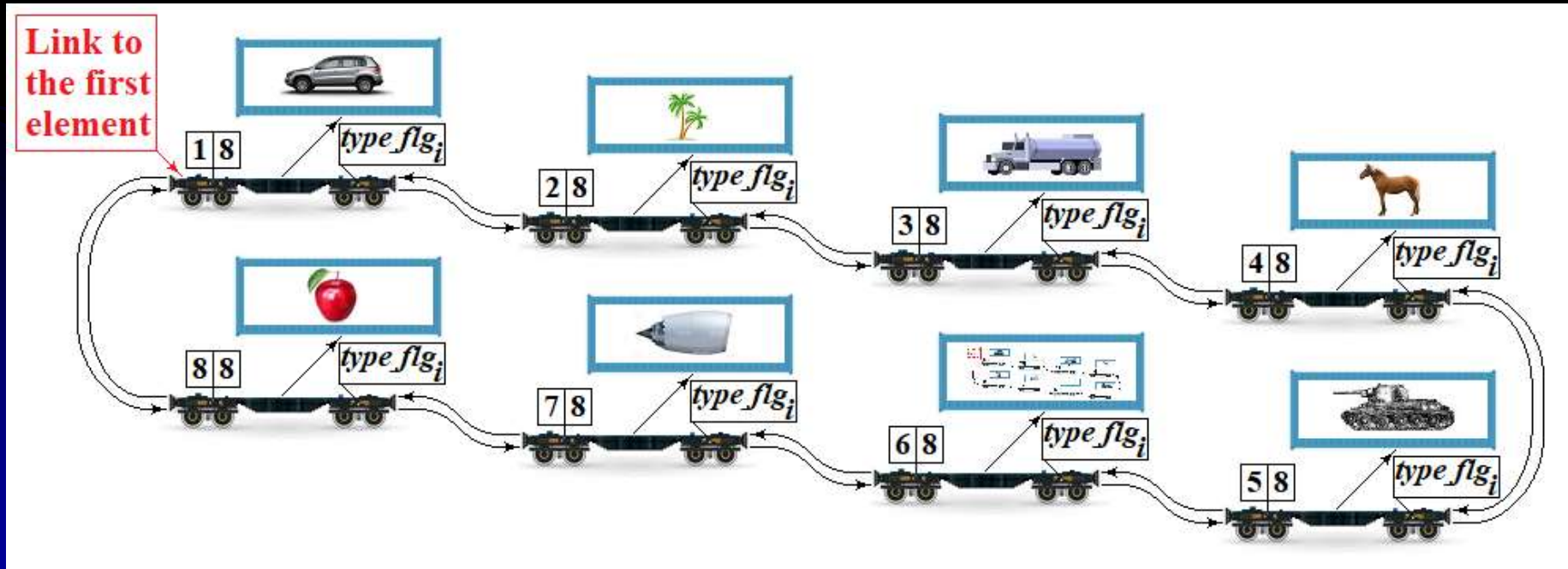
The screenshot shows the Eclipse IDE with a C++ project named 'OOP\_Test\_011'. The main editor displays the header file 'tmpmt\_cls\_CDLL.hpp'. The code defines a template class 'tmpmt\_cls\_CDLL' for a doubly linked list. A red circle highlights the line 'T i\_data;' in the 'node' struct, with a red arrow pointing to it from the text 'Same type for all the elements of the doubly linked list'. The code includes standard headers for file streams, strings, and typeinfo. The class has a private section for implicit construction and a public section for the 'node' struct and various methods. The 'node' struct contains pointers to previous and next nodes, a unique identifier 'node\_index', and a generic data field 'i\_data'. The class methods include initialization, insertion, deletion, and display functions. The IDE interface shows the Project Explorer on the left, the Outline view on the right, and the Build Console at the bottom.

```
22 #include <fstream>
23 #include <sstream>
24 #include <string>
25 #include <typeinfo>
26 //
27 template <class T>
28 class tmpmt_cls_CDLL {
29 private: // implicit, it may be not explicitly specified
30 // Structure of the node
31 struct node {
32 // Linking info
33 struct node* previous; // link to the previous element of the list
34 struct node* next; // link to the following element of the list
35 // Identification
36 int node_index; // unique identifier
37 // 'Payload' or 'useful data'
38 T i_data;
39 };
40 // Flag: if an element is found (=1) or not (=0)
41 int found_flag;
42 // Circular Doubly Linked List
43 node* CDL_list;
44 // Number of elements in the doubly linked list
45 int list_no_of_elems;
46 // Name/symbol of the current type; it is set using 'get_T_type_name()'
47 std::string T_type_name;
48 //
49 //
50 //
51 //
52 //
53 //
54 //
55 //
56 //
57 //
58 //
59 //
60 //
61 //
62 //
63 //
64 //
65 //
66 //
67 //
68 //
69 //
70 //
71 //
72 //
73 //
74 //
75 //
76 //
77 //
78 //
79 //
80 //
81 //
82 //
83 //
84 //
85 //
86 //
87 //
88 //
89 //
90 //
91 //
92 //
93 //
94 //
95 //
96 //
97 //
98 //
99 //
100 //
101 //
102 //
103 //
104 //
105 //
106 //
107 //
108 //
109 //
110 //
111 //
112 //
113 //
114 //
115 //
116 //
117 //
118 //
119 //
120 //
121 //
122 //
123 //
124 //
125 //
126 //
127 //
128 //
129 //
130 //
131 //
132 //
133 //
134 //
135 //
136 //
137 //
138 //
139 //
140 //
141 //
142 //
143 //
144 //
145 //
146 //
147 //
148 //
149 //
150 //
151 //
152 //
153 //
154 //
155 //
156 //
157 //
158 //
159 //
160 //
161 //
162 //
163 //
164 //
165 //
166 //
167 //
168 //
169 //
170 //
171 //
172 //
173 //
174 //
175 //
176 //
177 //
178 //
179 //
180 //
181 //
182 //
183 //
184 //
185 //
186 //
187 //
188 //
189 //
190 //
191 //
192 //
193 //
194 //
195 //
196 //
197 //
198 //
199 //
200 //
201 //
202 //
203 //
204 //
205 //
206 //
207 //
208 //
209 //
210 //
211 //
212 //
213 //
214 //
215 //
216 //
217 //
218 //
219 //
220 //
221 //
222 //
223 //
224 //
225 //
226 //
227 //
228 //
229 //
230 //
231 //
232 //
233 //
234 //
235 //
236 //
237 //
238 //
239 //
240 //
241 //
242 //
243 //
244 //
245 //
246 //
247 //
248 //
249 //
250 //
251 //
252 //
253 //
254 //
255 //
256 //
257 //
258 //
259 //
260 //
261 //
262 //
263 //
264 //
265 //
266 //
267 //
268 //
269 //
270 //
271 //
272 //
273 //
274 //
275 //
276 //
277 //
278 //
279 //
280 //
281 //
282 //
283 //
284 //
285 //
286 //
287 //
288 //
289 //
290 //
291 //
292 //
293 //
294 //
295 //
296 //
297 //
298 //
299 //
300 //
301 //
302 //
303 //
304 //
305 //
306 //
307 //
308 //
309 //
310 //
311 //
312 //
313 //
314 //
315 //
316 //
317 //
318 //
319 //
320 //
321 //
322 //
323 //
324 //
325 //
326 //
327 //
328 //
329 //
330 //
331 //
332 //
333 //
334 //
335 //
336 //
337 //
338 //
339 //
340 //
341 //
342 //
343 //
344 //
345 //
346 //
347 //
348 //
349 //
350 //
351 //
352 //
353 //
354 //
355 //
356 //
357 //
358 //
359 //
360 //
361 //
362 //
363 //
364 //
365 //
366 //
367 //
368 //
369 //
370 //
371 //
372 //
373 //
374 //
375 //
376 //
377 //
378 //
379 //
380 //
381 //
382 //
383 //
384 //
385 //
386 //
387 //
388 //
389 //
390 //
391 //
392 //
393 //
394 //
395 //
396 //
397 //
398 //
399 //
400 //
401 //
402 //
403 //
404 //
405 //
406 //
407 //
408 //
409 //
410 //
411 //
412 //
413 //
414 //
415 //
416 //
417 //
418 //
419 //
420 //
421 //
422 //
423 //
424 //
425 //
426 //
427 //
428 //
429 //
430 //
431 //
432 //
433 //
434 //
435 //
436 //
437 //
438 //
439 //
440 //
441 //
442 //
443 //
444 //
445 //
446 //
447 //
448 //
449 //
450 //
451 //
452 //
453 //
454 //
455 //
456 //
457 //
458 //
459 //
460 //
461 //
462 //
463 //
464 //
465 //
466 //
467 //
468 //
469 //
470 //
471 //
472 //
473 //
474 //
475 //
476 //
477 //
478 //
479 //
480 //
481 //
482 //
483 //
484 //
485 //
486 //
487 //
488 //
489 //
490 //
491 //
492 //
493 //
494 //
495 //
496 //
497 //
498 //
499 //
500 //
501 //
502 //
503 //
504 //
505 //
506 //
507 //
508 //
509 //
510 //
511 //
512 //
513 //
514 //
515 //
516 //
517 //
518 //
519 //
520 //
521 //
522 //
523 //
524 //
525 //
526 //
527 //
528 //
529 //
530 //
531 //
532 //
533 //
534 //
535 //
536 //
537 //
538 //
539 //
540 //
541 //
542 //
543 //
544 //
545 //
546 //
547 //
548 //
549 //
550 //
551 //
552 //
553 //
554 //
555 //
556 //
557 //
558 //
559 //
560 //
561 //
562 //
563 //
564 //
565 //
566 //
567 //
568 //
569 //
570 //
571 //
572 //
573 //
574 //
575 //
576 //
577 //
578 //
579 //
580 //
581 //
582 //
583 //
584 //
585 //
586 //
587 //
588 //
589 //
590 //
591 //
592 //
593 //
594 //
595 //
596 //
597 //
598 //
599 //
600 //
601 //
602 //
603 //
604 //
605 //
606 //
607 //
608 //
609 //
610 //
611 //
612 //
613 //
614 //
615 //
616 //
617 //
618 //
619 //
620 //
621 //
622 //
623 //
624 //
625 //
626 //
627 //
628 //
629 //
630 //
631 //
632 //
633 //
634 //
635 //
636 //
637 //
638 //
639 //
640 //
641 //
642 //
643 //
644 //
645 //
646 //
647 //
648 //
649 //
650 //
651 //
652 //
653 //
654 //
655 //
656 //
657 //
658 //
659 //
660 //
661 //
662 //
663 //
664 //
665 //
666 //
667 //
668 //
669 //
670 //
671 //
672 //
673 //
674 //
675 //
676 //
677 //
678 //
679 //
680 //
681 //
682 //
683 //
684 //
685 //
686 //
687 //
688 //
689 //
690 //
691 //
692 //
693 //
694 //
695 //
696 //
697 //
698 //
699 //
700 //
701 //
702 //
703 //
704 //
705 //
706 //
707 //
708 //
709 //
710 //
711 //
712 //
713 //
714 //
715 //
716 //
717 //
718 //
719 //
720 //
721 //
722 //
723 //
724 //
725 //
726 //
727 //
728 //
729 //
730 //
731 //
732 //
733 //
734 //
735 //
736 //
737 //
738 //
739 //
740 //
741 //
742 //
743 //
744 //
745 //
746 //
747 //
748 //
749 //
750 //
751 //
752 //
753 //
754 //
755 //
756 //
757 //
758 //
759 //
760 //
761 //
762 //
763 //
764 //
765 //
766 //
767 //
768 //
769 //
770 //
771 //
772 //
773 //
774 //
775 //
776 //
777 //
778 //
779 //
780 //
781 //
782 //
783 //
784 //
785 //
786 //
787 //
788 //
789 //
790 //
791 //
792 //
793 //
794 //
795 //
796 //
797 //
798 //
799 //
800 //
801 //
802 //
803 //
804 //
805 //
806 //
807 //
808 //
809 //
810 //
811 //
812 //
813 //
814 //
815 //
816 //
817 //
818 //
819 //
820 //
821 //
822 //
823 //
824 //
825 //
826 //
827 //
828 //
829 //
830 //
831 //
832 //
833 //
834 //
835 //
836 //
837 //
838 //
839 //
840 //
841 //
842 //
843 //
844 //
845 //
846 //
847 //
848 //
849 //
850 //
851 //
852 //
853 //
854 //
855 //
856 //
857 //
858 //
859 //
860 //
861 //
862 //
863 //
864 //
865 //
866 //
867 //
868 //
869 //
870 //
871 //
872 //
873 //
874 //
875 //
876 //
877 //
878 //
879 //
880 //
881 //
882 //
883 //
884 //
885 //
886 //
887 //
888 //
889 //
890 //
891 //
892 //
893 //
894 //
895 //
896 //
897 //
898 //
899 //
900 //
901 //
902 //
903 //
904 //
905 //
906 //
907 //
908 //
909 //
910 //
911 //
912 //
913 //
914 //
915 //
916 //
917 //
918 //
919 //
920 //
921 //
922 //
923 //
924 //
925 //
926 //
927 //
928 //
929 //
930 //
931 //
932 //
933 //
934 //
935 //
936 //
937 //
938 //
939 //
940 //
941 //
942 //
943 //
944 //
945 //
946 //
947 //
948 //
949 //
950 //
951 //
952 //
953 //
954 //
955 //
956 //
957 //
958 //
959 //
960 //
961 //
962 //
963 //
964 //
965 //
966 //
967 //
968 //
969 //
970 //
971 //
972 //
973 //
974 //
975 //
976 //
977 //
978 //
979 //
980 //
981 //
982 //
983 //
984 //
985 //
986 //
987 //
988 //
989 //
990 //
991 //
992 //
993 //
994 //
995 //
996 //
997 //
998 //
999 //
1000 //
1001 //
1002 //
1003 //
1004 //
1005 //
1006 //
1007 //
1008 //
1009 //
1010 //
1011 //
1012 //
1013 //
1014 //
1015 //
1016 //
1017 //
1018 //
1019 //
1020 //
1021 //
1022 //
1023 //
1024 //
1025 //
1026 //
1027 //
1028 //
1029 //
1030 //
1031 //
1032 //
1033 //
1034 //
1035 //
1036 //
1037 //
1038 //
1039 //
1040 //
1041 //
1042 //
1043 //
1044 //
1045 //
1046 //
1047 //
1048 //
1049 //
1050 //
1051 //
1052 //
1053 //
1054 //
1055 //
1056 //
1057 //
1058 //
1059 //
1060 //
1061 //
1062 //
1063 //
1064 //
1065 //
1066 //
1067 //
1068 //
1069 //
1070 //
1071 //
1072 //
1073 //
1074 //
1075 //
1076 //
1077 //
1078 //
1079 //
1080 //
1081 //
1082 //
1083 //
1084 //
1085 //
1086 //
1087 //
1088 //
1089 //
1090 //
1091 //
1092 //
1093 //
1094 //
1095 //
1096 //
1097 //
1098 //
1099 //
1100 //
1101 //
1102 //
1103 //
1104 //
1105 //
1106 //
1107 //
1108 //
1109 //
1110 //
1111 //
1112 //
1113 //
1114 //
1115 //
1116 //
1117 //
1118 //
1119 //
1120 //
1121 //
1122 //
1123 //
1124 //
1125 //
1126 //
1127 //
1128 //
1129 //
1130 //
1131 //
1132 //
1133 //
1134 //
1135 //
1136 //
1137 //
1138 //
1139 //
1140 //
1141 //
1142 //
1143 //
1144 //
1145 //
1146 //
1147 //
1148 //
1149 //
1150 //
1151 //
1152 //
1153 //
1154 //
1155 //
1156 //
1157 //
1158 //
1159 //
1160 //
1161 //
1162 //
1163 //
1164 //
1165 //
1166 //
1167 //
1168 //
1169 //
1170 //
1171 //
1172 //
1173 //
1174 //
1175 //
1176 //
1177 //
1178 //
1179 //
1180 //
1181 //
1182 //
1183 //
1184 //
1185 //
1186 //
1187 //
1188 //
1189 //
1190 //
1191 //
1192 //
1193 //
1194 //
1195 //
1196 //
1197 //
1198 //
1199 //
1200 //
1201 //
1202 //
1203 //
1204 //
1205 //
1206 //
1207 //
1208 //
1209 //
1210 //
1211 //
1212 //
1213 //
1214 //
1215 //
1216 //
1217 //
1218 //
1219 //
1220 //
1221 //
1222 //
1223 //
1224 //
1225 //
1226 //
1227 //
1228 //
1229 //
1230 //
1231 //
1232 //
1233 //
1234 //
1235 //
1236 //
1237 //
1238 //
1239 //
1240 //
1241 //
1242 //
1243 //
1244 //
1245 //
1246 //
1247 //
1248 //
1249 //
1250 //
1251 //
1252 //
1253 //
1254 //
1255 //
1256 //
1257 //
1258 //
1259 //
1260 //
1261 //
1262 //
1263 //
1264 //
1265 //
1266 //
1267 //
1268 //
1269 //
1270 //
1271 //
1272 //
1273 //
1274 //
1275 //
1276 //
1277 //
1278 //
1279 //
1280 //
1281 //
1282 //
1283 //
1284 //
1285 //
1286 //
1287 //
1288 //
1289 //
1290 //
1291 //
1292 //
1293 //
1294 //
1295 //
1296 //
1297 //
1298 //
1299 //
1300 //
1301 //
1302 //
1303 //
1304 //
1305 //
1306 //
1307 //
1308 //
1309 //
1310 //
1311 //
1312 //
1313 //
1314 //
1315 //
1316 //
1317 //
1318 //
1319 //
1320 //
1321 //
1322 //
1323 //
1324 //
1325 //
1326 //
1327 //
1328 //
1329 //
1330 //
1331 //
1332 //
1333 //
1334 //
1335 //
1336 //
1337 //
1338 //
1339 //
1340 //
1341 //
1342 //
1343 //
1344 //
1345 //
1346 //
1347 //
1348 //
1349 //
1350 //
1351 //
1352 //
1353 //
1354 //
1355 //
1356 //
1357 //
1358 //
1359 //
1360 //
1361 //
1362 //
1363 //
1364 //
1365 //
1366 //
1367 //
1368 //
1369 //
1370 //
1371 //
1372 //
1373 //
1374 //
1375 //
1376 //
1377 //
1378 //
1379 //
1380 //
1381 //
1382 //
1383 //
1384 //
1385 //
1386 //
1387 //
1388 //
1389 //
1390 //
1391 //
1392 //
1393 //
1394 //
1395 //
1396 //
1397 //
1398 //
1399 //
1400 //
1401 //
1402 //
1403 //
1404 //
1405 //
1406 //
1407 //
1408 //
1409 //
1410 //
1411 //
1412 //
1413 //
1414 //
1415 //
1416 //
1417 //
1418 //
1419 //
1420 //
1421 //
1422 //
1423 //
1424 //
1425 //
1426 //
1427 //
1428 //
1429 //
1430 //
1431 //
1432 //
1433 //
1434 //
1435 //
1436 //
1437 //
1438 //
1439 //
1440 //
1441 //
1442 //
1443 //
1444 //
1445 //
1446 //
1447 //
1448 //
1449 //
1450 //
1451 //
1452 //
1453 //
1454 //
1455 //
1456 //
1457 //
1458 //
1459 //
1460 //
1461 //
1462 //
1463 //
1464 //
1465 //
1466 //
1467 //
1468 //
1469 //
1470 //
1471 //
1472 //
1473 //
1474 //
1475 //
1476 //
1477 //
1478 //
1479 //
1480 //
1481 //
1482 //
1483 //
1484 //
1485 //
1486 //
1487 //
1488 //
1489 //
1490 //
1491 //
1492 //
1493 //
1494 //
1495 //
1496 //
1497 //
1498 //
1499 //
1500 //
1501 //
1502 //
1503 //
1504 //
1505 //
1506 //
1507 //
1508 //
1509 //
1510 //
1511 //
1512 //
1513 //
1514 //
1515 //
1516 //
1517 //
1518 //
1519 //
1520 //
1521 //
1522 //
1523 //
1524 //
1525 //
1526 //
1527 //
1528 //
1529 //
1530 //
1531 //
1532 //
1533 //
1534 //
1535 //
1536 //
1537 //
1538 //
1539 //
1540 //
1541 //
1542 //
1543 //
1544 //
1545 //
1546 //
1547 //
1548 //
1549 //
1550 //
1551 //
1552 //
1553 //
1554 //
1555 //
1556 //
1557 //
1558 //
1559 //
1560 //
1561 //
1562 //
1563 //
1564 //
1565 //
1566 //
1567 //
1568 //
1569 //
1570 //
1571 //
1572 //
1573 //
1574 //
1575 //
1576 //
1577 //
1578 //
1579 //
1580 //
1581 //
1582 //
1583 //
1584 //
1585 //
1586 //
1587 //
1588 //
1589 //
1590 //
1591 //
1592 //
1593 //
1594 //
1595 //
1596 //
1597 //
1598 //
1599 //
1600 //
1601 //
1602 //
1603 //
1604 //
1605 //
1606 //
1607 //
1608 //
1609 //
1610 //
1611 //
1612 //
1613 //
1614 //
1615 //
1616 //
1617 //
1618 //
1619 //
1620 //
1621 //
1622 //
1623 //
1624 //
1625 //
1626 //
1627 //
1628 //
1629 //
1630 //
1631 //
1632 //
1633 //
1634 //
1635 //
1636 //
1637 //
1638 //
1639 //
1640 //
1641 //
1642 //
1643 //
1644 //
1645 //
1646 //
1647 //
1648 //
1649 //
1650 //
1651 //
1652 //
1653 //
1654 //
1655 //
1656 //
1657 //
1658 //
1659 //
1660 //
1661 //
1662 //
1663 //
1664 //
1665 //
1666 //
1667 //
1668 //
1669 //
1670 //
1671 //
1672 //
1673 //
1674 //
1675 //
1676 //
1677 //
1678 //
1679 //
1680 //
1681 //
1682 //
1683 //
1684 //
1685 //
1686 //
1687 //
1688 //
1689 //
1690 //
1691 //
1692 //
1693 //
1694 //
1695 //
1696 //
1697 //
1698 //
1699 //
1700 //
1701 //
1702 //
1703 //
1704 //
1705 //
1706 //
1707 //
1708 //
1709 //
1710 //
1711 //
1712 //
1713 //
1714 //
1715 //
1716 //
1717 //
1718 //
1719 //
1720 //
1721 //
1722 //
1723 //
1724 //
1725 //
1726 //
1727 //
1728 //
1729 //
1730 //
1731 //
1732 //
1733 //
1734 //
1735 //
1736 //
1737 //
1738 //
1739 //
1740 //
1741 //
1742 //
1743 //
1744 //
1745 //
1746 //
1747 //
1748 //
1749 //
1750 //
1751 //
1752 //
1753 //
1754 //
1755 //
1756 //
1757 //
1758 //
1759 //
1760 //
1761 //
1762 //
1763 //
1764 //
1765 //
1766 //
1767 //
1768 //
1769 //
1770 //
1771 //
1772 //
1773 //
1774 //
1775 //
1776 //
1777 //
1778 //
1779 //
1780 //
1781 //
1782 //
1783 //
1784 //
1785 //
1786 //
1787 //
1788 //
1789 //
1790 //
1791 //
1792 //
1793 //
1794 //
1795 //
1796 //
1797 //
1798 //
1799 //
1800 //
1801 //
1802 //
1803 //
1804 //
1805 //
1806 //
1807 //
1808 //
1809 //
1810 //
1811 //
1812 //
1813 //
1814 //
1815 //
1816 //
1817 //
1818 //
1819 //
1820 //
1821 //
1822 //
1823 //
1824 //
1825 //
1826 //
1827 //
1828 //
1829 //
1830 //
1831 //
1832 //
1833 //
1834 //
1835 //
1836 //
1837 //
1838 //
1839 //
1840 //
1841 //
1842 //
1843 //
1844 //
1845 //
1846 //
1847 //
1848 //
1849 //
1850 //
1851 //
1852 //
1853 //
1854 //
1855 //
1856 //
1857 //
1858 //
1859 //
1860 //
1861 //
1862 //
1863 //
1864 //
1865 //
1866 //
1867 //
1868 //
1869 //
1870 //
1871 //
1872 //
1873 //
1874 //
1875 //
1876 //
1877 //
1878 //
1879 //
1880 //
1881 //
1882 //
1883 //
1884 //
1885 //
1886 //
1887 //
1888 //
1889 //
1890 //
1891 //
1892 //
1893 //
1894 //
1895 //
1896 //
1897 //
1898 //
1899 //
1900 //
1901 //
1902 //
1903 //
1904 //
1905 //
1906 //
1907 //
1908 //
1909 //
1910 //
1911 //
1912 //
1913 //
1914 //
1915 //
1916 //
1917 //
1918 //
1919 //
1920 //
1921 //
1922 //
1923 //
1924 //
1925 //
1926 //
1927 //
1928 //
1929 //
1930 //
1931 //
1932 //
1933 //
1934 //
1935 //
1936 //
1937 //
1938 //
1939 //
1940 //
1941 //
1942 //
1943 //
1944 //
1945 //
1946 //
1947 //
1948 //
1949 //
1950 //
1951 //
1952 //
1953 //
1954 //
1955 //
1956 //
1957 //
1958 //
1959 //
1960 //
1961 //
1962 //
1963 //
1964 //
1965 //
1966 //
1967 //
1968 //
1969 //
1970 //
1971 //
1972 //
1973 //
1974 //
1975 //
1976 //
1977 //
1978 //
1979 //
1980 //
1981 //
1982 //
1983 //
1984 //
1985 //
1986 //
1987 //
1988 //
1989 //
1990 //
1991 //
1992 //
1993 //
1994 //
1995 //
1996 //
1997 //
1998 //
1999 //
2000 //
2001 //
2002 //
2003 //
2004 //
2005 //
2006 //
2007 //
2008 //
2009 //
2010 //
2011 //
2012 //
2013 //
2014 //
2015 //
2016 //
2017 //
2018 //
2019 //
2020 //
2021 //
2022 //
2023 //
2024 //
2025 //
2026 //
2027 //
2028 //
2029 //
2030 //
2031 //
2032 //
2033 //
2034 //
2035 //
2036 //
2037 //
2038 //
2039 //
2040 //
2041 //
2042 //
2043 //
2044 //
2045 //
2046 //
2047 //
2048 //
2049 //
2050 //
2051 //
2052 //
2053 //
2054 //
2055 //
2056 //
2057 //
2058 //
2059 //
2060 //
2061 //
2062 //
2063 //
2064 //
2065 //
2066 //
2067 //
2068 //
2069 //
2070 //
2071 //
2072 //
2073 //
2074 //
2075 //
2076 //
2077 //
2078 //
2079 //
2080 //
2081 //
2082 //
2083 //
2084 //
2085 //
2086 //
2087 //
2088 //
2089 //
2090 //
2091 //
2092 //
2093 //
2094 //
2095 //
2096 //
2097 //
2098 //
2099 //
2100 //
2101 //
2102 //
2103 //
2104 //
2105 //
2106 //
2107 //
2108 //
2109 //
2110 //
2111 //
2112 //
2113 //
2114 //
2115 //
2116 //
2117 //
2118 //
2119 //
2120 //
2121 //
2122 //
2123 //
2124 //
2125 //
2126 //
2127 //
2128 //
2129 //
2130 //
2131 //
2132 //
2133 //
2134 //
2135 //
2136 //
2137 //
2138 //
2139 //
2140 //
2141 //
2142 //
2143 //
2144 //
2145 //
2146 //
2147 //
2148 //
2149 //
2150 //
2151 //
2152 //
2153 //
2154 //
2155 //
2156 //
2157 //
2158 //
2159 //
2160 //
2161 //
2162 //
216
```

# Results

## Why not a general solution?

- General solution: each element may have a distinct type;
- Template solution is not so general ← however why is it preferable?;
- SWOT analysis?
- Example 1: coordinates ← geometry, design, FEM, calculus domain;
- Example 2: identifiers ← graph theory, element definition (FEM);



# Results

## Some technical info and, of course, more questions

- Functions in header files and fields of the classes;
- Input/output variables of T type ← how to handle 'generic' types;

```
22 #include <fstream>
23 #include <sstream>
24 #include <string>
25 #include <typeinfo>
26 //
27 template <class T>
28 class tmpit_cls_CDLL {
29 private: // implicit, it may be not explicitly specified
30 // Structure of the node
31 struct node {
32 // Linking info
33 struct node* previous; // link to the previous element of the list
34 struct node* next; // link to the following element of the list
35 // Identification
36 int node_index; // unique identifier
37 // 'Payload' or 'useful data'
38 // In future versions this will be replaced by a general data
39 T i_data;
40 };
41 // Flag: if an element is found (=1) or not (=0)
42 int found_flag;
43 // Circular Doubly Linked List
44 node* CDL_list;
45 // Number of elements in the doubly linked list
46 int list_no_of_elems;
47 // Name/symbol of the current type; it is set using 'get_T_type_name()'
48 std::string T_type_name;
49 //
```

Functions in the class

Return is T

Same type for all the elements of the doubly linked list



# Results

## Type identification using a 'common use' compiler

- Function (programming 'trick') used to identify a 'generic' type . . .

```
44 // Get the type of T
45 template <class T>
46 std::string tmp_t_cls_CDLL<T>::get_T_type_name() {
47     // Out: out_str_type
48     std::string out_str_type;
49     // Get the type name in an explicit way
50     std::string tmp_str_type = typeid(this->CDL_list->i_data).name();
51     std::cout<<" - - - - - -> Current symbol is '"<<tmp_str_type<<"!\n";
52     out_str_type="UNKNOWN!";
53     if (!tmp_str_type.compare("NSt7__cxx112basic_stringIcSt11char_traitsIcESaIcEEE")) {
54         out_str_type="string";
55     }
56     if (!tmp_str_type.compare("b")) { out_str_type="bool"; }
57     if (!tmp_str_type.compare("c")) { out_str_type="char"; }
58     if (!tmp_str_type.compare("i")) { out_str_type="int"; }
59     if (!tmp_str_type.compare("j")) { out_str_type="size_t"; }
60     if (!tmp_str_type.compare("l")) { out_str_type="long"; }
61     if (!tmp_str_type.compare("m")) { out_str_type="unsigned long"; }
62     if (!tmp_str_type.compare("x")) { out_str_type="long long"; }
63     if (!tmp_str_type.compare("y")) { out_str_type="unsigned long long"; }
64     if (!tmp_str_type.compare("f")) { out_str_type="float"; }
65     if (!tmp_str_type.compare("d")) { out_str_type="double"; }
66     if (!tmp_str_type.compare("e")) { out_str_type="long double"; }
67     std::cout<<" - - - - - -> Current type is '"<<out_str_type<<"!\n";
68     return(out_str_type);
69 }
70 //
```

# Results

## T 'generic' type conversion

- Function (programming trick) used to convert a 'generic' type . . .
- Idea: convert  $T$  generic type to string & from string;
- Handling strings is paramount!
- Strings connect the data to the input/output (?CSV?) text files;

```
70 //
71 template <class T>
72 T tmp_cls_CDLL<T>::convert_to_T (std::string str_data) {
73     T out_T;
74     std::stringstream ss(str_data);
75     if (!this->T_type_name.compare("string")) {
76         str_data=lib_str_funcs.str_replace_all_occurrences(str_data, " ", str_replacementChars);
77     }
78     ss << str_data;
79     ss >> out_T;
80     return(out_T);
81 } // End of 'convert_to_T'
82 //
```



# Results

## Operating the sample application

- Operations → test the generation of various  $T$  types of doubly lnkd lists;
- Operations used to operate the doubly lnkd lists ← common in all apps;

```
MENU: SELECT THE TYPE OF THE DOUBLY CONNECTED LIST:
1. 'int' is the type of the 'payload'; input file: "inp_int.txt";
2. 'float' is the type of the 'payload'; input file: "inp_float.txt";
3. 'string' is the type of the 'payload'; input file: "inp_str.txt";
4. Get out of the application.
The selected option [1..4] is <-1>: 3 ← string

Menu - operations with the current doubly linked circular list:
1. insert a new element AFTER a given node of the list;
2. delete an element AFTER a given node of the list;
3. display the list from #1 upto nr_elems, on distinct lines;
4. display the list from #1 upto nr_elems, on the same line;
5. display the list starting from the current element, circular, on distinct lines;
6. display the list starting from the current element, circular, on the same line;
7. display the element located in position 'i';
8. find the element which contains 'payload';
9. modify an element of the list;
10. interactive creation of the doubly linked list;
11. load from the file the doubly linked list;
12. save in a file the doubly linked list;
13. delete the doubly linked list;
14. get out of this menu.

----> Tip: The first operation is the creation of the list, options 10 or 11!! <----

The selected option [1..14] is <-1>: 11

You selected option #11!

Input the name of the file from which the elements of the list are loaded: inp_str_CSV.csv

'display_list' -> display the current list
node 1 contains the info 'Melvin KAMINSKY,EGOT,"Brooklyn, New York City, USA";
node 2 contains the info 'Alfred Joseph HITCHCOCK,Several Awards,"Leytonstone, Essex, England";
node 3 contains the info 'George Orson WELLES,Several Awards,"Kenosha, Wisconsin, USA";
node 4 contains the info 'Michael CACOYANIS,Several Awards,"Limassol, British Cyprus";
node 5 contains the info 'Vasilis GEORGIADIS,Golden Globe Awards,"Dardanelles, Ottoman Empire";
```

```
Command Prompt - "C:\Program Files\Eclipse Software\CDLL\mp1t\class\Debug>21/04/2022> 9:35:09.809"OOP_Test_011 - CDLL
C:\Workspace-Eclipse\OOP_Test_011 - CDLL\mp1t\class\Debug>21/04/2022> 9:35:09.809"OOP_Test_011 - CDLL

MENU DE ALEGERE A TIPURII DE lista circulara dublu inlantuita:
1. 'int' este tipul 'payload'-ului; fisier de date: "inp_int.txt";
2. 'float' este tipul 'payload'-ului; fisier de date: "inp_float.txt";
3. 'string' este tipul 'payload'-ului; fisier de date: "inp_str.txt";
4. Iesire din aplicatie.
Optiunea aleasa [1..4] este <-1>: 1
Ati ales optiunea 1! ← int

Menu operare lista circulara dublu inlantuita:
1. inserare nou element DUPA un nod din lista;
2. stergere element DUPA un nod din lista;
3. afisare lista de la 1 la nr_elems, pe linii distincte;
4. afisare lista de la 1 la nr_elems, pe aceeaasi linie;
5. afisare lista de la elementul curent, circular, pe linii distincte;
6. afisare lista de la elementul curent, circular, pe aceeaasi linie;
7. afisare element din pozitia i;
8. gaseste element care contine 'payload';
9. modificare element din lista;
10. creare interactiva a listei circulare dublu inlantuita;
11. citire din fisier lista circulara dublu inlantuita;
12. scriere in fisier lista circulara dublu inlantuita;
13. stergere lista circulara dublu inlantuita;
14. iesire din program.

----> Tip: Prima operatie este cea de generare a listei, optiunile 10 sau 11!! <----

Optiunea aleasa [1..14] este <-1>: 11

Ati ales optiunea 11!

Introduceti numele fisierului din care se citesc elementele listei: inp_int.txt
Citire 'payload' din fisierul "inp_int.txt":
1. "111" ==> 111;
2. "222" ==> 222;
3. "333" ==> 333;
4. "444" ==> 444;
5. "555" ==> 555;
```

# Results

## Some Standard Template Library C++ solutions

- Solution used to store random values, testing the degree of randomness;
- 'map' = associative container; 'ulli' stands for 'unsigned long long int';

```
39 // Definition of the map<map::key_type,map::mapped_type>; map<unique_key,payload_data>
40 // The largest size payload data type in C++ (beside GMP)
41 std::map<unsigned long long int, unsigned long long int> map_ulli;
42 std::map<unsigned long long int, unsigned long long int>::iterator iter_map_ulli; // iterator
```

- Solution used to store strings which symbolize points' coordinates;
- Useful to compute the coefficients within the polynomial regression;

```
38 //
39 // Definition of the map<map::key_type,map::mapped_type>; map<unique_key,payload_data>
40 // The largest size payload data type in C++ (beside GMP)
41 // This is a one-dimensional array of strings 'on steroids' --> map_1dim_string
42 typedef std::map<unsigned long long int, std::string> map_1dim_string;
43 typedef std::map<unsigned long long int, std::string>::iterator iter_map_1dim_string; // iterator
44 // This is a two-dimensional array of strings 'on steroids' --> map_2dim_string
45 // Remark:
46 // a. with respect to a matrix, it may be considered either a vector of lines
47 // or a vector of columns, depending on the way the map_2dim_string is
48 // initialised, i.e. loaded with the elements of the matrix, either
49 // along the lines i.e. line after line, or along columns, i.e. column
50 // after column;
51 // b. calculi in GMP are performed locally, by converting the string element
52 // of the matrix in the appropriate mpl_t type and then performing the
53 // current calculus
54 typedef std::map<unsigned long long int, std::map<unsigned long long int, std::string>> map_2dim_string;
55 typedef std::map<unsigned long long int, std::map<unsigned long long int, std::string>>::iterator iter_map_2dim_string; // iterator
56 //
57 map_2dim_string map_2dim_string_csv_content; // The matrix of strings defined as a 2dim map
58
```

# Conclusion

## Usefulness

- Modeling complex phenomena requires advanced concepts & instruments;
- Data integration and knowledge acquisition require computer based original software;
- Hybrid research approaches use composite models with deeply integrated modules;
- Facile and rapid development of software components is paramount;
- Various software libraries: solvers, interfaces, RNGs, data persistency etc.;
- “Failing to prepare, you are preparing to fail” → software development strategy;
- Strategy is valuable, being confirmed over the past 38 years in many R&D projects;

## Progresses in the last 3 years

- Header files approach → class based programming, i.e. (atomization → integration);
- Class templates based programming = generic programming;
- Standard Template Library in C++: containers, iterators, algorithms, functions;
- Collections in Java (not presented in this paper);

## Accomplishments

- Modern know how in the development of new software components for hybrid models
- Updated libraries and new libraries based on the aforementioned progresses;
- New computer based models in data science;

# Acknowledgement

The ideas presented in this scientific paper are inspired by the results of the ID1223 - “**Computer Aided Advanced Studies in Applied Elasticity from an Interdisciplinary Perspective**” Scientific Research Project, under the supervision of the National University Research Council (CNCSIS), Romania, 2007-2010<sup>7</sup> and by the follow-up scientific research project “**Mathematical Models for Inter-Domain Approaches with Applications in Engineering and Economy**”, under the supervision of the National Authority for Scientific Research, Romania, 2010-2012<sup>8</sup>.

# References

- [1] Oanta, E., Nicolescu, B., “A Versatile PC-Based Method for the Processing of the Large Matrices”, Proc. DETC99/CIE ‘1999 ASME Design Engineering Technical Conference’, September 12-15, 1999, Las Vegas, Nevada, ISBN 9780791819722, Vol 2, paper 9059, pp 457-464, <https://doi.org/10.1115/DETC99/CIE-9059> (1999).
- [2] Oanta, E., “Hybrid modeling in mechanical engineering”, Habilitation thesis, September 21 2018, Doctoral School of Mechanical Engineering, Constanta Maritime University, <https://cmu-edu.eu/blog/2018/09/18/sustinere-teza-de-abilitare-conf-univ-dr-ing-emil-m-oanta-21-09-2018/>, Constanta (2018).
- [3] Oanta, E., Pescaru, A., Lazaroiu, G., “General data structure for the dynamic memory allocation in the development of the computer based models in engineering”, Proc. SPIE 10977, Advanced Topics in Optoelectronics, Microelectronics, and Nanotechnologies IX, eISSN: 1996-756X, ISSN: 0277-786X, ISBN: 978-1-5106-2614-0; 1097721 (31 December 2018); doi: 10.1117/12.2324291; <https://doi.org/10.1117/12.2324291> (2018).
- [4] \* \* \*, Standard Template Library, [https://en.wikipedia.org/wiki/Standard\\_Template\\_Library](https://en.wikipedia.org/wiki/Standard_Template_Library) (18 July 2022).
- [5] Sabau, A., “Transient Regimes Analysis for a Diesel Engine”, Book Series: Advanced Materials Research, Volume 837, Page 471-476, DOI 10.4028/www.scientific.net/AMR.837.471, 2014, ISSN 1022-6680, ISBN 978-3-03785-929-2 (2014).
- [6] Sabau, A., “Analysis of flow in fluidized bed of particles”, Book Series: IOP Conference Series-Materials Science and Engineering, Volume 916, DOI 10.1088/1757-899X/916/1/012097, 2020, ISSN 1022-6680, ISBN 978-3-03785-929-2 (2020).
- [7] Oanță, E., & all, “Computer Aided Advanced Studies in Applied Elasticity from an Interdisciplinary Perspective”, Research Project ID1223, under the supervision of the National University Research Council (CNCSIS), Romania, 2007-2010.
- [8] Oanta, E., Panait, C., Lepadatu, L., Tamas, R., Constantinescu, M. et. all., Mathematical Models for Inter-Domain Approaches with Applications in Engineering and Economy, MIEC2010 - Bilateral Romania-Moldavia Scientific Research Project, 2010-2012, under the supervision of the National Authority for Scientific Research, Romania.



# Questions?



***“Failing to prepare, you are preparing to fail” Benjamin Franklin***