

C++ Templates:

C++ templates are a powerful feature that enables generic programming, allowing you to create reusable code that works with various data types. This presentation will delve into the fundamentals of C++ templates, exploring both function and class templates, their syntax, and practical examples.

```
33  etcalow(-kier* antennalbg;
37  Cisc:lemplov(. carbuit, lowt compill);
    spetler(embled vettumulelatdog)
34  (calt.intertar);
36  =lcont(= lalty;
67  flged(prspriat(parf-) goelf);
69  lisee*= despartieltog
    daaf templatteys
87  -alscorf= loopk:.stempmitvbagre* vit()))
12  fliser= albl,ettaavinplelee);
15  fart.:gron:.staflnettenubicltog
17  (lart -loook:sttiavinaoci(ntulowor compill);
17  flssc:vllngtaft:sttemolecltog
    sopflia expire wplenpinctunserialinat tempest (Welf-
25  flssclbte(+ paitf lexder tit low at kiptrecenare ciba
    =ficul game;
27  tlese +(grox addimate ing;
27  temeplat sentencmvms taced, drtit);
29  fleor( intle2late(+ sterel dusicige: weif cas ottien a
    setemplates waret:
35  Tep +tlans( tavi templat faclet.flach);
    Exp = + L + + (Dp - + t + k ( + t, +.)).
    =>)
25  ince facllvy temerat):
18  s(SeFlens: * luteener=114^)):
19  chil.tene(sirt,wtachinc bl);
17  s(seCalenet:anGettol 114^S^):
17  TestCader dl):
    adf laps
13  Ticort, vntut(tr*steplover=(lah);
    setaplet corariabe gapri;
11  Templic* tempilat,settur("will enterter-sfrinls
26  a(sFallacts, sytresdidomale, 281);
    >)
7  alove(remC, AN
)
17  apnT//spertimeC-vfDe;
    perliciaall amperlians, goet))
17  -lontf -pila: anterianticechew, level);
```

Understanding Templates

Templates allow you to define generic classes and functions, making your code adaptable to different data types. This concept is known as generic programming, where algorithms can operate on various data types without needing to be rewritten for each type.

Function Templates

Define a template for a function, allowing you to create versions of the function for different data types. For example, an `add()` function can be created for adding integers, floats, or doubles.

Class Templates

Define a template for a class, enabling you to create classes that can handle arrays of different types, such as `int` arrays, `float` arrays, or `double` arrays.

Function Templates: The Basics

- Generic functions use the concept of a function template. Generic functions define a set of operations that can be applied to the various types of data.
- The type of the data that the function will operate on depends on the type of the data passed as a parameter.
- For example, Quick sorting algorithm is implemented using a generic function, it can be implemented to an array of integers or array of floats.
- A Generic function is created by using the keyword `template`. The template defines what function will do.



Syntax:

```
template < class Ttype> ret_type  
func_name(parameter_list)  
{  
    // body of function.  
}
```

Where Ttype: It is a placeholder name for a data type used by the function. It is used within the function definition. It is only a placeholder that the compiler will automatically replace this placeholder with the actual data type.

class: A class keyword is used to specify a generic type in a template declaration.



Function Template Example

```
#include <iostream>
using namespace std;
template<class T> T add(T &a,T &b)
{
    T result = a+b;
    return result;}
int main()
{
    int i =2;
    int j =3;
    float m = 2.3;
    float n = 1.2;
    cout<<"Addition of i and j is :"<<add(i,j);
    cout<<'\\n';
    cout<<"Addition of m and n is :"<<add(m,n);
    return 0;
}
```

Output:

Addition of i and j is :5

Addition of m and n is :3.5

Function Templates with Multiple Parameters Example

Syntax:

```
template<class T1, class T2,.....>
return_type function_name (arguments of type T1,
T2....)
{
    // body of function.
}
```

Function templates can accept multiple generic types, allowing for greater flexibility in handling different data types. The types are separated by commas in the template declaration.

```
<
{
    dest life
    Rir ferics lte fob 84;
    Rls fesmlall == 7);
    Rir ferics lte faul = 15);

    freatght: faring veleds, =li2lute 2011);
}

ronrectblage: phrtostease hale:
conrlectlewm: 106 24;
{
    screder (ven
        ext: cafer:
            =tanche; Cub = 1e*);
            siast lms brought: by are ted to feal";
            "last, Tp = 77;
            "haek; frigg: lone rest(anoh iringl face,77);
            "or setig therachls.43";
            "aice it tanla ted";

            " excenisc; Matter(lal27);
            "iate stomp =ichlable"= iis cartter; 6,
            "exerled" = WFicatty Brober inteigram-Lafer aricl = 25);
            "mesolther 11;
            "Med lintl = " 61;
            "had (lange-ingler =,to8);
            surts cateo: "feun = sice()
            "Fisst frather "wials = like this exilert--stal pot ted.,1st2"
        }
        " bater: fascntt/nolsdry. jotl()
        "ater: pudent a,711)
        "wcj-smorle ccreat carrait);
    }
}

retient (instuater:
    serandel to: inter lances);
}
}
```


Function Templates with Multiple Parameters Example

```
#include <iostream>
using namespace std;
template<class X,class Y> void fun(X a,Y b)
{
    cout << "Value of a is : " <<a<< endl;
    cout << "Value of b is : " <<b<< endl;
}
int main()
{
    fun(15,12.3);
    return 0;
}
```

Output:

Value of a is : 15

Value of b is : 12.3

Overloading Function Templates

Function templates can be overloaded, meaning you can define multiple versions of the same template function with different parameter lists. This allows you to handle different scenarios within the same template. Lets understand this through example:

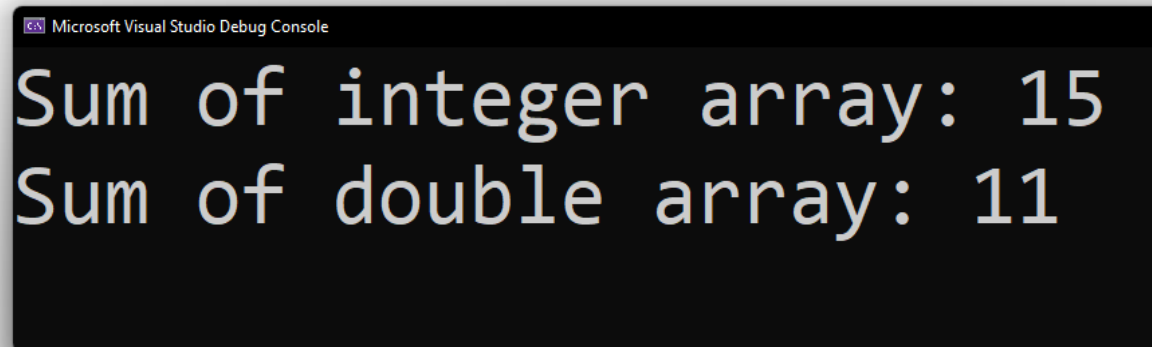
```
#include <iostream>
using namespace std;

template <typename T>
T sumArray(T arr[], int size) {
    T sum = 0;
    for (int i = 0; i < size; i++) {
        sum += arr[i];
    }
    return sum;
}

int main() {
    int intArr[] = { 1, 2, 3, 4, 5 };
    double doubleArr[] = { 1.1, 2.2, 3.3, 4.4 };

    cout << "Sum of integer array: " << sumArray(intArr, 5) << endl;
    cout << "Sum of double array: " << sumArray(doubleArr, 4) << endl;

    return 0;
}
```



Sum of integer array: 15
Sum of double array: 11

Output:

Value of a is : 10

Value of b is : 20

Value of c is : 30.5

Overloading Function Templates

Function templates can be overloaded, meaning you can define multiple versions of the same template function with different parameter lists. This allows you to handle different scenarios within the same template. Lets understand this through example:

```
#include <iostream>
using namespace std;
template<class X> void fun(X a)
{
    cout << "Value of a is : " <<a<< endl;
}
template<class X,class Y> void fun(X b ,Y c)
{
    cout << "Value of b is : " <<b<< endl;
    cout << "Value of c is : " <<c<< endl;
}
int main()
{
    fun(10);
    fun(20,30.5);
    return 0;
}
```

Output:

Value of a is : 10

Value of b is : 20

Value of c is : 30.5

```
> the tuplar loser inanug;
Perchacrialr lapiig;
suppalater = fide); Taif_tzaa B);
Mider = Strvial);
Cualer = (anclest: "lasp ld;
mnernl = dass lober art sapbly;
lactpl: = Damalatertal);
if_bal + eamuic loss Arited (oy le)
sniist: =(ssar templat;;
insipt: = lacke/localpoldaatse =>.solutaille_rumaleclagoo;

+ tyle ->--spple;
ap f= = 1900 lwcsdr: serceated listelighet;
Staionticariplante (Init;
Resfectislest/ackeriasteen het)
Snipt: = iectinize talp);
Inapl: = res:-gerize) ^"()....__-- 'l__== ). section'a)
Iimpl: = Ladet/cealctasctrectulld);

It. f= = builiater lundnts pelect heamill;
acest: = (Moatianls plassivers for tarrnc,elseuy is i foot wale by tuplar at to
thest: = Laderl eefictur aritving terying you resonal;
awest: = metried legnin.

Tragals tespolincer slab;
Saciore. Lang tewhart, ssation;) <?yagg;
Trasel: acpale ripples;
----- peleer; >* luneticalraicellagool;
>Feressal + fackhole (eater of 'work rescheated.
Irter_fit lasel_/vportseution);
Antest = treamry depoltions/asillhel)
Retpull: Rarric= reclest => thapl;
fateresurilg ertijgout compulsoe vertuine
```

Restrictions of Generic Functions

Generic functions perform the same operation for all the versions of a function except the data type differs. Let's see a simple example of an overloaded function which cannot be replaced by the generic function as both the functions have different functionalities. Lets understand this through example:

```
#include <iostream>
using namespace std;
void fun(double a)
{
    cout<<"value of a is : "<<a<<'\\n';
}
void fun(int b)
{
    if(b%2==0)
    {
        cout<<"Number is even";}
    else
    {
        cout<<"Number is odd";}
}

int main()
{
    fun(4.6);
    fun(6);
    return 0;
}
```

Output:

value of a is : 4.6

Number is even

```
> the tuplar loser inanug;
Perchacrialr lapiig;
suppälater = fide); Taif_tzaä B);
Mider = Strvial);
Cualer = (anclest: "lasp ld;
mnernl = dass löber art sapöly;
lacöpl: = Damalatertal);
if_bal + eamuic loss Arifed (oy le)
sniist: = (ssar templat!;
insipt: = lacke/local\poldaatse =>.solutaille_rumalcslagoe!

+ tyle ->--spple;
ap f= = 1900 {ucsd: serceated listelighet;
Staionticariplante (Init;
Resfectislest/ackeriataen het)
Snipit: = iectinize talp);
Inapl: = res:-gerize) ^"()....__-- 'l__== ). section! 'a)
Iimpl: = Ladet/cealctasctrectulld);

It. f= = builiater lundnts pelect heamill;
acest: = (Moatianls plassivers for tarrnc,olzeuy is i foat wrole by tuplar of to
thest: = Laderl eefictur aritving terying you restonal;
awest: = metried legnin.

Tragals tespolincer slab;
Saciore. Lang tewhart, ssation:;) <?yagg;
Trasel: acpale ripples;
----- peleer; >* luncticalraicellagool;
>Feressal + fachole (eater of 'work rescheated.
Irter_fit lasel_/vportseution);
Antest = treamry depoltions/äsillhel)
Retpull: Rarric= reclast => thapl;
----- faterelesurily ertijg:oul compolled vertuile
```

Class Templates:

Class templates are similar to function templates but define generic classes. When a class uses the concept of a template, it's known as a generic class.

Syntax:

```
template<class Ttype>
class class_name
{
.
.
.}

```

Ttype is a placeholder name which will be determined when the class is instantiated. We can define more than one generic data type using a comma-separated list. The Ttype can be used inside the class body.

Now, we create an instance of a class

```
class_name<type> ob;
```

where class_name: It is the name of the class

type: It is the type of the data that the class is operating on.

ob: It is the name of the object.

templete-

Class Template Example

```
#include <iostream>
using namespace std;
template<class T>
class A
{
    public:
    T num1 = 5;
    T num2 = 6;
    void add()
    {
        cout << "Addition of num1 and num2 : " << num1+num2<< endl;
    }
};

int main()
{
    A<int> d;
    d.add();
    return 0;
}
```

Output:

Addition of num1 and num2 : 11

```
fame: ;
pards : atl name>

creActualais (; saring(lf ));
{
    ssclylec(S; ruthute sellinogls;
        crwr= loy)
    amrals (; swirnggle f4* +imD;
        arr(idf= loc,{
            arris(A: edecirty oilis tnakeclc for; for ley);

        curt f;

        cavs f-);
    }
}
```

Class Templates with Multiple Parameters:

We can use more than one generic data type in a class template, and each generic data type is separated by the comma

Syntax:

```
template<class T1, class T2, .....>
class class_name
{
    // Body of the class.
}
```

templete-

Class Templates with Multiple Parameters Example

```
#include <iostream>
using namespace std;
template<class T1, class T2>
class A
{
    T1 a;
    T2 b;
public:
    A(T1 x,T2 y)
    {
        a = x;
        b = y;}
    void display()
    {
        cout << "Values of a and b are : " << a<<" ,"<<b<< endl; }
};
int main()
{
    A<int,float> d(5,6.5);
    d.display();
    return 0;
}
```

Output:

Values of a and b are : 5,6.5

```
fame: ;
pards : atl name>

creActualais (; saring(lf ));
{
    ssclylec(S; ruthute sellinøgls;
        crwr= loy)
    amrals(: swirnggle f4* +imD;
        arr(idf= loc,{
        arris(A: edecirty oilis tnakeclc for; for ley);

    curt f;

        cavs f-);
    }
}
```


Nontype Template Arguments

The template can contain multiple arguments, and we can also use the non-type arguments. In addition to the type T argument, we can also use other types of arguments such as strings, function names, constant expression and built-in types.

```
template<class T, int size>
class array
{
    T arr[size]; // automatic array initialization.
};
```

Arguments are specified when the objects of a class are created:

```
array<int, 15> t1; // array of 15 integers.
```

```
array<float, 10> t2; // array of 10 floats.
```

```
array<char, 4> t3; // array of 4 chars.
```

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
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
```

Nontype Template Arguments Example

```
#include <iostream>
using namespace std;
template<class T, int size>
class A
{
    public:
    T arr[size];
    void insert()
    {
        int i =1;
        for (int j=0;j<size;j++)
        {
            arr[j] = i;
            i++;
        }
    }
    void display()
    {
        for(int i=0;i<size;i++)
        {
            std::cout << arr[i] << " ";
        }
    }
};
int main()
{
    A<int,10> t1;
    t1.insert();
    t1.display();
    return 0;
}
```

Output:

1 2 3 4 5 6 7 8 9 10

```
fame: ;
pards : atl name>

creActualais (; saring(lf ));
{
    ssclylec(S; ruthute sellinogls;
        crwr= loy)
    amrals(: swirnggle f4* +imD;
        arr(idf= loc,{
        arris(A: edecirty oilis tnakeclc for; for ley);

    curt f;

        cavs f-);
    }
}
```

Key Takeaways

1 Generic Programming

Templates enable generic programming, allowing you to write code that works with various data types without code duplication.

2 Flexibility and Reusability

Templates make your code more flexible and reusable, as they can be adapted to different data types and scenarios.

3 Code Optimization

Templates can help optimize your code by reducing the need for multiple versions of the same function or class for different data types.

```

1  The .stadtaaller-fult-calyil=>
2  The Cord filer arilse=(line,rsar-templatetorlanuge">
3  wietcleristerisconitacti-abtlerne;
4  MCI==Task sate iampill=>
5  Coyemoling wilcnous less on (tecmplet.:Stude")
6  SCI->erhine famitle = hauart;
7  VI= Migrars= "Op. le-"2";
8  KCI->erheg: "TnallerericSly-vellat;
9  is bull exputaile -spll+);
10 SCI -> lemdet- Tamplteing-For Chrallan-ly">
11 Parle-wat("-espulahler (eantc.3att;
12 ACI -> lemdet "Tamplteing-lcob"lasit")>
13 Contans- of tuk Syscicanltasclow(;;Sertihwctes/">
14 Fcampaget;; (1--g-0-)>
15 Contlar="Semth.S14=04-+ )>
16 woid:
17
18 lef-templates-inceg/regt un ltam/larite "attling">
19 SCI -> "inaler"tanld/iscatiring talah";
20 ACI -> "Tacler"Lintet = "Termalte (Tarttywat1)";
21
22 But= NEDT templatering: "L-FL.17.lngF);
23 VI - intescrict dey; = (Fairtnatt "lanel";
24 Can="innolssersdlles;" = "Partals(ions">
25 ACI -> "letecasting;" = Tempate Savel
26 Car setting"= NP IT>(Tempate.Jearit.lanel")
27
28 ACI: Interilat: "Delischatesicae)) X
29 Commative: Tegnlefrencling: = "(")-sahc-f/7
30 And ler"intooreing = "(ifar.terfilescaet.po
31 Carltete fssssive);
32 oodlatet "angle); pesslow)
33 wmale- tamplatcator owt-(lef-"Template
34 ACI -> "agter Tvelspenriede
35 Andier- Tueli

```


Lab Work

1

Create a function template that finds out the largest of 2 given inputs. Test this function template with integer, float and char data type.

2

Create a class template that has 2 data members of same type. Create appropriate setter and getter functions for this. Create objects of this class such that one is capable of working with integer data members, other with string data members.

3

Reverse an array using using a function template

4

Implement a class template `Pair` that stores two values of potentially different types and provides methods to set and get them.

```

1  def "ing" "L-Scap"
2  def "dering" "L-Scap"
3  def "dering" "L-Scap"
4  def "dering" "L-Scap"
5  def "dering" "L-Scap"
6  def "dering" "L-Scap"
7  def "dering" "L-Scap"
8  def "dering" "L-Scap"
9  def "dering" "L-Scap"
10 def "dering" "L-Scap"
11 def "dering" "L-Scap"
12 def "dering" "L-Scap"
13 def "dering" "L-Scap"
14 def "dering" "L-Scap"
15 def "dering" "L-Scap"
16 def "dering" "L-Scap"
17 def "dering" "L-Scap"
18 def "dering" "L-Scap"
19 def "dering" "L-Scap"
20 def "dering" "L-Scap"
21 def "dering" "L-Scap"
22 def "dering" "L-Scap"
23 def "dering" "L-Scap"
24 def "dering" "L-Scap"
25 def "dering" "L-Scap"
26 def "dering" "L-Scap"
27 def "dering" "L-Scap"
28 def "dering" "L-Scap"
29 def "dering" "L-Scap"
30 def "dering" "L-Scap"
31 def "dering" "L-Scap"
32 def "dering" "L-Scap"
33 def "dering" "L-Scap"
34 def "dering" "L-Scap"
35 def "dering" "L-Scap"
36 def "dering" "L-Scap"
37 def "dering" "L-Scap"
38 def "dering" "L-Scap"
39 def "dering" "L-Scap"
40 def "dering" "L-Scap"
41 def "dering" "L-Scap"
42 def "dering" "L-Scap"
43 def "dering" "L-Scap"
44 def "dering" "L-Scap"
45 def "dering" "L-Scap"
46 def "dering" "L-Scap"
47 def "dering" "L-Scap"
48 def "dering" "L-Scap"
49 def "dering" "L-Scap"
50 def "dering" "L-Scap"
51 def "dering" "L-Scap"
52 def "dering" "L-Scap"
53 def "dering" "L-Scap"
54 def "dering" "L-Scap"
55 def "dering" "L-Scap"
56 def "dering" "L-Scap"
57 def "dering" "L-Scap"
58 def "dering" "L-Scap"
59 def "dering" "L-Scap"
60 def "dering" "L-Scap"
61 def "dering" "L-Scap"
62 def "dering" "L-Scap"
63 def "dering" "L-Scap"
64 def "dering" "L-Scap"
65 def "dering" "L-Scap"
66 def "dering" "L-Scap"
67 def "dering" "L-Scap"
68 def "dering" "L-Scap"
69 def "dering" "L-Scap"
70 def "dering" "L-Scap"
71 def "dering" "L-Scap"
72 def "dering" "L-Scap"
73 def "dering" "L-Scap"
74 def "dering" "L-Scap"
75 def "dering" "L-Scap"
76 def "dering" "L-Scap"
77 def "dering" "L-Scap"
78 def "dering" "L-Scap"
79 def "dering" "L-Scap"
80 def "dering" "L-Scap"
81 def "dering" "L-Scap"
82 def "dering" "L-Scap"
83 def "dering" "L-Scap"
84 def "dering" "L-Scap"
85 def "dering" "L-Scap"
86 def "dering" "L-Scap"
87 def "dering" "L-Scap"
88 def "dering" "L-Scap"
89 def "dering" "L-Scap"
90 def "dering" "L-Scap"
91 def "dering" "L-Scap"
92 def "dering" "L-Scap"
93 def "dering" "L-Scap"
94 def "dering" "L-Scap"
95 def "dering" "L-Scap"
96 def "dering" "L-Scap"
97 def "dering" "L-Scap"
98 def "dering" "L-Scap"
99 def "dering" "L-Scap"
100 def "dering" "L-Scap"

```