Using **auto**to declare dataypes can save lot of time during programming contests.  
  
When a variable is defined as

**auto**

, compiler determines its type during compile-time.For example,

1. **auto** a = 1; *// a will become 'int'*
2. **auto** b = 1LL; *// b will become 'long long'*
3. **auto** c = 1.0; *// c will become 'double'*
4. **auto** d = "variable"; *// d will become 'string'*

This is extremely useful when dealing with iterators. For iterating over

map<int,vector<int>> Map

, instead of writing this,

1. **for**(map<int,vector<int>>:: iterator it = Map.begin(); it != Map.end(); it++ ){}

it is possible to write this.

1. **for**(**auto** it = Map.begin(); it != Map.end(); it++ ){}

There is an even shorter way to write it.

1. **for**(**auto** &it : Map){}

These type of loops are called **range based loops**, previously which was supported by java only. Note that,

it

 is not an iterator in range based loops. So for accessing elements,

it.

 should be used instead of

it->

.  
  
N.B. These are **C++11** features, which is supported by all major online judges. Will not work for earlier versions.

1. Instead of using

scanf()

 and

printf()

 use

ios::sync\_with\_stdio(**false**);

 and

cout/cin

. Don`t use

scanf()

 and

printf()

 after writing

ios::sync\_with\_stdio(**false**);

, you will get incorrect output.  
Note :

ios::sync\_with\_stdio(**false**);

 doesn`t speed up I/O, it just disables the synchronization between C++ streams and C streams.  
2. Instead of writing

vector<**long** **long**>::const\_iterator cit;

, write

**auto** cit=[NameOfVector].cbegin()

.  
3. Never use

std::list<>

. They are only useful for large data types.  
4. Use

*#include* <bits/stdc++.h>

 instead of writing each header file one-by-one. This only affects compile time. And compile time has nothing to do with competitive programming. Only running time matters!  
5. Use range-based for loops instead of writing full for loop for C++ Containers for accessing all values.  
6. Tired of remembering return type of functions. Use [auto](http://en.cppreference.com/w/cpp/language/auto).  
7. Learn different functions available in

<algorithm>

 library of C++. They are quite useful in places.  
8. Instead of using

make\_pair()

 function, use

std::initializer\_list<>

.eg.

1. *#include* <bits/stdc++.h>
2. **using** **namespace** std;
3. int main() {
4. vector<pair<int,int>> ar;
5. *//use*
6. ar.push\_back({1,2});
7. ar.push\_back({4,5});
8. *//instead of*
9. ar.push\_back(make\_pair(1,2));
10. ar.push\_back(make\_pair(3,4));
11. *//though both compiled successfully*
12. **for**(**const** **auto**& x : ar)
13. cout<<x.first<<' '<<x.second<<'\n';
14. **return** 0;
15. }

Both are same but 1st one takes less time to code!  
9. Minimize the use of

endl

 manipulator with output streams in C++. It not only put '\n' but also does unnecessary flushing of stream. Instead do

cout<<'\n';

 and write

cout<<flush;

 at last.

**2.) Use of tie and tuple**

Following code will illustrate the usage of tie and tuple

1. tuple<int,int,int> t1 = make\_tuple(1,2,3)
2. tuple<**char**,int,**double**,int,**long** **long** int >t2;
3. t2=make\_tuple( 'a' , 2 , 2.3 , 1 , 10000 );
4. cout<<get<2>t1<< " "<< get<0> t2;*// will print 3 and 'a'*
5. *// Usage of tie*
6. int a;
7. int b;
8. **char** c;
9. tie ( a, b , c ) = make\_tuple(4,1,'a'); *// this is equilvalent to a=4; b=1 ; c='a'*
10. tie ( a , b ) = make\_tuple(b,a) *// swapping b and a*
11. int c=10;
12. int d=5;
13. int e=15;
14. tie(c,d,e) = make\_tuple( c+d+e, c+2\*d , 10\*c ); *// Now c will be equal to 30, //d will be equal to 20 and e will be equal to 100.*

**3.) C++ LAMBDAS**  
Just like python , c++11 also has functionality of lambdas. Lambdas are the functions that have body but doesn't have name.Simply , they are nameless functions.  
Syntax of lambdas are:  [](type1 name1,type2 name2){some computation related to variables name1 and name2}  
for example;

1. *//To sort elements of vector v we can use :*
2. **bool** func(int a,int b){
3. **return** a>b;
4. }
5. sort(v.begin(),v.end(),func);
6. *// or simply we can use Lambdas*
7. sort(v.begin(),v.end(),[](int a,int b){ **return** a>b });

**4.) Use of emplace\_back and move for vectors**

In  c++11 , emplace\_back work just like push\_back in case of vectors . It is  beneficial to use emplace\_back instead of push\_back because emplace\_back just adds the value at the end of the vector whereas push\_back stores values in temporary variable and then adds the value at the end of vector.Thus push\_back is little slower than emplace\_back.

1. vector<int>v;
2. v.push\_back(1);
3. v.emplace\_back(2);

While working with STL , you can use 'move' to just move vector ,  not to copy it all.

1. vector<int> u ={1,2,3,4};
2. *// or use can write*
3. *//u.push\_back(1);*
4. *//u.push\_back(2);*
5. *//u.push\_back(3);*
6. *//u.push\_back(4)...and so on*
7. vector<int> w =move(u);

Use of move can make code memory efficient , For example;

1. vector<int> func(int n){
2. vector<int>v;
3. **for**(int i=0;i<n;i++){
4. v.push\_back(i);
5. }
6. **return** v;
7. }
8. *// or alternatively you can use*
9. vector<int> func(int n){
10. **for**(int i=0;i<n;i++){
11. v.push\_back(i);
12. }
13. **return** move(v);
14. }
15. *// Latter one will save o(n) memory.*

**5) Get rid of includes**  
Use

1. *#include*<bits/stdc++.h>

This library includes all the libraries used in everyday coding like iostream,stdio.h,algorithm,set,vector..and many more.You don't need to include any of these.Thus only one**#include** instead to 10-15 **#include** 's to write a code.  
**6) Conditional operators**

1. **if**(x==1)
2. cout<<"Yes\n";
3. **else**
4. cout<<"No\n";
5. *// above code can be written in short form as*
6. cout<<(x==1?"Yes":"No);
7. *// one more example*
8. **if**(x%2==0)
9. x=5;
10. **else**
11. x=6;
12. *// it can be written shortly as*
13. x=(x%2?6:5);

**7) Some Special  functions**  
1.) \_\_gcd(A,B) will return greatest common divisor of A and B. No need to use Euclidean Algorithm for calculating greatest common divisor.  
  
2.) swap(A,B) will swap values of A and B  
  
3.)use of**sort function:**  
 You can use inbuilt sort function present in **#include<algorithm>**instead of writing your own sorting algorithm   
for example;

1. int a[]={5,2,3,4,1};
2. sort(a,a+n); *// now a will contain {1,2,3,4,5}*
3. vector<int>a={5,2,3,4,1};
4. sort(a.begin(),a.end());*// now a will contain {1,2,3,4,5}*

4.) \_\_builtin\_popcount(R) will return all the set bits in binary representation of R  
  
5.)\_\_builtin\_ffs(R) will give 1+least significant 1 bit in binary representation of R  
  
6.) \_\_builtin\_ctz(R) will give number of trailing zeros in binary representation of R.  
  
7.)\_\_builtin\_clz(R) will give 32 - total number of bits in binary representation of R if R is**unsigned int** and 64- total number of bits in binary representation of R if R is **unsigned long long int**