```
load "Utility.m";

function DivisorsWithNorm(I, n)
// Input: Z_K-Ideal I; norm n in Z

// Output: List of divisors of I with norm n

  norm := Integers() ! Norm(I);

  if n eq 1 then return [I*I^(-1)]; end if;
  if not IsDivisibleBy(norm, n) then return []; end if;
  if norm eq n then return [I]; end if;

  Fact := Factorization(I);

  p1 := Fact[1][1];
  s1 := Fact[1][2];
  np := Integers() ! Norm(p1);

  Results := [];

  for j in [0..s1] do
    if IsDivisibleBy(n, np^j) then
      B := DivisorsWithNorm(I*p1^(-s1), Integers() ! (n / np^j));

      for J in B do
        Append(~Results, p1^j*J);
      end for;
    end if;
  end for;

  return Results;

end function;


function TotallyRealGenerator(I, K, Kpos)
// Input: Z_K-Ideal I; Field K; Field Kpos

// Output: Boolean that indicates success; totally real generator of I cap Kpos

  ZK := Integers(K);
  ZKpos := Integers(Kpos);

  Ipos:=ideal<ZKpos|1>;
  Split:=[];

  Fact := Factorization(I);

  for i in [1..#Fact] do
    if i in Split then continue; end if;

    pi:=Fact[i][1];
    si:=Fact[i][2];
```

```
    piConj := IdealConjugate(pi,K);

    p:=MinimalInteger(pi);

    pFact:=Factorization(ideal< ZKpos | p >);

    for qj in [fact[1] : fact in pFact] do
      if ideal<ZK | Generators(qj)> subset pi then
        a := qj;
        break;
      end if;
    end for;

    aZK := ideal<ZK|Generators(a)>;

    if aZK eq pi^2 then

      if not IsDivisibleBy(si, 2) then return false, _; end if;
      Ipos *:= a^(Integers() ! (si/2));

    elif aZK eq pi then

      Ipos *:= a^si;

    elif aZK eq pi*piConj then

      if Valuation(I, pi) ne Valuation(I, piConj) then return false, _; end if;
      Ipos *:= a^si;
      for j in [1..#Fact] do
        pj := Fact[j][1];
        if pj eq piConj then
          Append(~Split, j);
          break;
        end if;
      end for;
    end if;
  end for;

  return IsPrincipal(Ipos);

end function;


function EmbeddingMatrix(K, Kpos)
// Input: Field K; Field Kpos

// Output: Matrix M whose entries give the signs of the embeddings of the fundamental units; List U of all totally positive units in ZKpos modulo norms; List of generators of a subgroup of Z_Kpos^* of odd index

  ZKpos := Integers(Kpos);

  t := #Basis(ZKpos);

```

```
  G, mG := pFundamentalUnits(ZKpos,2);
  FundUnits := [mG(G.i) : i in [1..t]];

  M := ZeroMatrix(GF(2), t, t);

  for i in [1..t] do
    Embeds := RealEmbeddings(FundUnits[i]);
    for j in [1..t] do
      if Embeds[j] lt 0 then
        M[i][j] := 1;
      end if;
    end for;
  end for;

  U := [];
  for a in Kernel(M) do
    e := ZKpos ! &*[FundUnits[i]^(Integers() ! a[i]) : i in [1..t]];
    Append(~U, e);
  end for;

  ZRel := Integers(RelativeField(Kpos, K));

  Units := [];
  for u in U do
    for w in Units do
      if NormEquation(ZRel,ZRel ! (u/w)) then
        continue u;
      end if;
    end for;

    Append(~Units, u);
  end for;

  return M, Units, FundUnits;

end function;


function TotallyPositiveGenerators(alpha, K, Kpos, M, U, FundUnits)
// Input: alpha in ZKpos; Field K; Field Kpos; Embedding-Matrix M; List U of all totally-positive units in ZKpos modulo norms; List FundUnits of generators of a subgroup of Z_Kpos^* of odd index

// Output: Boolean that inducates success; List of all totally-positive generators of alpha*ZK modulo norms

  t := #Basis(Kpos);
  V := ZeroMatrix(GF(2), 1, t);

  Embeds := RealEmbeddings(alpha);
  for i in [1..t] do
    if Embeds[i] lt 0 then
      V[1][i] := 1;
```

```
    end if;
  end for;

  solvable, x := IsConsistent(M,V);
  if not solvable then
    return false, _;
  end if;

  g := Integers(Kpos) ! &*[FundUnits[i]^(Integers() ! x[1][i]) : i in [1..t]];

  return true, [alpha*g*u : u in U];

end function;


function ClassesModGalois(K)
// Input : Field K

// Output : List of representatives of the class group of Z_K modulo the action of the Galois-group of K/Q

  ZK := Integers(K);
  Cl, mCl := ClassGroup(ZK : Proof:="GRH");

  ClModGal:=[];
  for a in Cl do
    A:=mCl(a);
    for f in Automorphisms(K) do
      if Inverse(mCl)(ideal<ZK | [f(x) : x in Generators(A)]>) in ClModGal then
        continue a;
      end if;
    end for;
    Append(~ClModGal,a);
  end for;

  return [mCl(g) : g in ClModGal];

end function;



function LatFromIdeal(J, alpha, K)
// Input: ZK-Ideal J; Totally positive element alpha Kpos; Field K

// Output: Z-Lattice with elements J and inner product (x,y) := Tr(alpha*x*Conj(y))

  n := #Basis(K);
  z := PrimitiveElement(K);

  GeneratorMatrix := KMatrixSpace(Rationals(), #Generators(J)*n, n) ! 0;

```

```
  for i in [1..#Generators(J)] do
    g := K ! (Generators(J)[i]);

    for j in [1..n] do
      GeneratorMatrix[(i-1)*n + j] := Vector(Rationals(), n, Eltseq(g*z^(j-1)));
    end for;
  end for;


  BaseVecs := Basis(Lattice(GeneratorMatrix));

  ZBase := [];
  for i in [1..n] do
    b := K ! 0;
    for j in [1..n] do
      b +:= BaseVecs[i][j]*z^(j-1);
    end for;
    Append(~ZBase, b);
  end for;

  InnProd := KMatrixSpace(Rationals(), n, n) ! 0;
  for i in [1..n] do
    for j in [1..n] do
      InnProd[i][j] := Trace(K ! (alpha * z^(i-j)));
    end for;
  end for;

  L := LatticeWithBasis(KMatrixSpace(Rationals(), n, n) ! Matrix(BaseVecs), InnProd);
  L := LatticeWithGram(LLLGram(GramMatrix(L)));

  return L;

end function;


function IdealLattices(d, K, Kpos, A, M, U, FundUnits, Reduce)
// Input: d in N; Field K; Field Kpos; Class Group of K mod Galois-Group A; Embedding-Matrix M; List of totally-positive units U; List FundUnits of generators of a subgroup of Z_Kpos^* of odd index; Boolean Reduce that indicates, whether the list shall be reduced by isometry.

// Output: List of all even ideal-lattices over K of square-free level and determinant d

  ZK := Integers(K);
  InvDiff := Different(ZK)^(-1);

  l := &*(PrimeDivisors(d));

  B := DivisorsWithNorm(ideal<ZK|l>, d);

  Results := [];

```

```
  for I in A do
    for b in B do
      J := (I*IdealConjugate(I,K))^(-1)*InvDiff*b;

      x, alphaPrime := TotallyRealGenerator(J, K, Kpos);

      if x then
        y, TotPos := TotallyPositiveGenerators(alphaPrime, K, Kpos, M, U, FundUnits);
        if y then
          for alpha in TotPos do
            L := LatFromIdeal(I, alpha, K);
            if IsEven(L) then
              Append(~Results, L);
            end if;
          end for;
        end if;
      end if;
    end for;
  end for;

  if Reduce then Results := ReduceByIsometry(Results); end if;

  return Results;

end function;


function ModIdLat(l, n)
// Input: square-free l in N; n in N

// Output: List of all l-modular lattices of dimension n that are ideal lattices over some cyclotomic field reduced by isometry

  det := l^(Integers() ! (n/2));

  Lattices := [];

  for m in [m : m in EulerPhiInverse(n) | m mod 4 ne 2] do
    K<z> := CyclotomicField(m);
    Kpos := sub<K | z + z^(-1)>;

    A := ClassesModGalois(K);
    M, U, FundUnits := EmbeddingMatrix(K, Kpos);
    Lattices cat:= IdealLattices(det, K, Kpos, A, M, U, FundUnits, false);
  end for;

  Lattices := ReduceByIsometry(Lattices);

  PrintFileMagma(Sprintf("IdealLattices/%o-Modular/%o-Dimensional", l, n), Lattices : Overwrite := true);

  return Lattices;

```

```
end function;

procedure MainLoop()
  for n := 2 to 36 by 2 do
    for l in [1,2,3,5,6,7,11,14,15,23] do
      printf "dim = %o, l = %o\n", n, l;
      Results := ModIdLat(l, n);
      ModList := [L : L in Results | IsModular(L, l)];
      StrongModList := [L : L in Results | IsStronglyModular(L,l)];
      PrintFileMagma(Sprintf("SubidealLattices/%o-Modular/%o-Dimensional", l, n), Results : Overwrite := true);
      PrintFileMagma(Sprintf("SubidealLattices/%o-Modular/%o-DimensionalModular", l, n), ModList : Overwrite := true);
      PrintFileMagma(Sprintf("SubidealLattices/%o-Modular/%o-DimensionalStronglyModular", l, n), StrongModList : Overwrite := true);

      if #Results gt 0 then
        printf "\n\n----------n = %o, l = %o: %o lattices found! %o of them are modular and %o are strongly modular----------\n\n", n, l, #Results, #ModList, #StrongModList;
      end if;
    end for;
  end for;
end procedure;
```