else if(!deadline && !T1.getErrorDate()) {

int a;

for(a=0;a<tokens.size();a++) { //recurring task

if(tokens[a]=="every") {

R1.setRecWord(tokens[a]);

if((a+2)<=tokens.size() && isDayValid(tokens[a+1])) {

if(tokens[a+2]=="until"){

//findYear(tokens,(a+2),T1);

// if(T1.getErrorDate())

// return T1;

if((a+6)==tokens.size()) {

R1.setWord(tokens[a+2]);

/\*if(isYearValid) {

for(int b=0;b<a;b++) {

desc = desc + originalTokens[b] + " ";

}

T1.setDescription(desc);

R1.setTaskDay(tokens[a+1]);

R1.setEndDate(tokens[a+3]);

R1.setEndMonth(tokens[a+4]);

R1.setRecurring(true);

break;

}\*/

try {

gregDate = stoi(tokens[a+3]);

}

catch (const std::invalid\_argument) {

R1.setRecurring(false);

break;

}

gregMonth = getMonthNumber(tokens[a+4]);

try {

gregYear = stoi(tokens[a+5]);

}

catch (const std::invalid\_argument) {

R1.setRecurring(false);

break;

}

try {

boost::gregorian::date d(gregYear, gregMonth, gregDate);

ptime now = microsec\_clock::local\_time();

boost::gregorian::date today = now.date();

if(d<today) {

R1.setRecurringError(true);

R1.setRecurring(false);

}

else {

recurring=true;

R1.setRecurring(recurring);

}

for(int b=0;b<a;b++) {

desc = desc + originalTokens[b] + " ";

}

T1.setDescription(desc);

R1.setTaskDay(tokens[a+1]);

R1.setEndDate(tokens[a+3]);

R1.setEndMonth(tokens[a+4]);

break;

}

catch (std::out\_of\_range) {

bool value=true;

cout << "error8" << endl;

T1.setErrorDate(value);

recurring=false;

}

}

}

/\* else if((a+3)==tokens.size() && ((tokens[a+2]).at(0)=='x' || (tokens[a+2]).at(0)=='X') && isdigit(tokens[a+2].at(1))) {

R1.setWord(tokens[a+2]);

for(int b=0;b<a;b++) {

desc = desc + originalTokens[b] + " ";

}

T1.setDescription(desc);

R1.setTaskDay(tokens[a+1]);

R1.setRecurring(true);

break;

}

else if(tokens[a+4]=="until" && tokens[a+2]=="by") {

findYear(tokens,(a+4),T1);

if(T1.getErrorDate())

return T1;

vector <string> byRecTime;

boost::split(byRecTime,tokens[a+3],boost::is\_any\_of(":"));

if(byRecTime.size()==2 && isHourValid(byRecTime[0]) && isMinuteValid(byRecTime[1])) {

T1.setHour(byRecTime[0]);

T1.setMinute(byRecTime[1]);

}

if((a+7)==tokens.size()) {

R1.setWord(tokens[a+4]);

if(isYearValid) {

R1.setRecurring(true);

for(int b=0;b<a;b++) {

desc = desc + originalTokens[b] + " ";

}

T1.setDescription(desc);

R1.setTaskDay(tokens[a+1]);

R1.setEndDate(tokens[a+5]);

R1.setEndMonth(tokens[a+6]);

break;

}

try {

gregDate = stoi(tokens[a+5]);

}

catch (const std::invalid\_argument) {

R1.setRecurringError(true);

break;

}

gregMonth = getMonthNumber(tokens[a+6]);

try {

boost::gregorian::date d(gregYear, gregMonth, gregDate);

ptime now = microsec\_clock::local\_time();

boost::gregorian::date today = now.date();

if(d<today) {

dateValid=false;

R1.setRecurringError(true);

R1.setRecurring(false);

}

else {

dateValid=true;

recurring=true;

R1.setRecurring(recurring);

}

for(int b=0;b<a;b++) {

desc = desc + originalTokens[b] + " ";

}

T1.setDescription(desc);

R1.setTaskDay(tokens[a+1]);

R1.setEndDate(tokens[a+5]);

R1.setEndMonth(tokens[a+6]);

break;

}

catch (std::out\_of\_range) {

bool value=true;

cout << "error9" << endl;

T1.setErrorDate(value);

recurring=false;

}

}

else {

bool value=true;

T1.setTimeError(value);

return T1;

}

}

else if((a+5)==tokens.size() && ((tokens[a+4]).at(0)=='x' || (tokens[a+4]).at(0)=='X') && isdigit(tokens[a+4].at(1))) {

vector <string> byRecTime;

boost::split(byRecTime,tokens[a+3],boost::is\_any\_of(":"));

if(byRecTime.size()==2 && isHourValid(byRecTime[0]) && isMinuteValid(byRecTime[1])) {

T1.setHour(byRecTime[0]);

T1.setMinute(byRecTime[1]);

}

else {

bool value=true;

T1.setTimeError(value);

return T1;

}

R1.setWord(tokens[a+4]);

for(int b=0;b<a;b++) {

desc = desc + originalTokens[b] + " ";

}

T1.setDescription(desc);

R1.setTaskDay(tokens[a+1]);

R1.setRecurring(true);

break;

}

else if(tokens[a+6]=="until" && tokens[a+2]=="from" && tokens[a+4]=="to") {

findYear(tokens,(a+6),T1);

if(T1.getErrorDate())

return T1;

vector <string> fromRecTime;

boost::split(fromRecTime,tokens[a+3],boost::is\_any\_of(":"));

vector <string> toRecTime;

boost::split(toRecTime,tokens[a+5],boost::is\_any\_of(":"));

if(fromRecTime.size()==2 && toRecTime.size()==2 && isHourValid(toRecTime[0]) && isHourValid(fromRecTime[0]) && isMinuteValid(toRecTime[1]) && isMinuteValid(fromRecTime[1])) {

T1.setEndHour(toRecTime[0]);

T1.setEndMinute(toRecTime[1]);

T1.setStartHour(fromRecTime[0]);

T1.setStartMinute(fromRecTime[1]);

}

else {

bool value=true;

T1.setTimeError(value);

return T1;

}

if((a+9)==tokens.size()) {

R1.setWord(tokens[a+6]);

if(isYearValid) {

for(int b=0;b<a;b++) {

desc = desc + originalTokens[b] + " ";

}

T1.setDescription(desc);

R1.setTaskDay(tokens[a+1]);

R1.setEndDate(tokens[a+7]);

R1.setEndMonth(tokens[a+8]);

R1.setRecurring(true);

break;

}

try {

gregDate = stoi(tokens[a+7]);

}

catch (const std::invalid\_argument) {

R1.setRecurringError(true);

break;

}

gregMonth = getMonthNumber(tokens[a+8]);

try {

boost::gregorian::date d(gregYear, gregMonth, gregDate);

ptime now = microsec\_clock::local\_time();

boost::gregorian::date today = now.date();

if(d<today) {

R1.setRecurringError(true);

R1.setRecurring(false);

}

else {

recurring=true;

R1.setRecurring(recurring);

}

for(int b=0;b<a;b++) {

desc = desc + originalTokens[b] + " ";

}

T1.setDescription(desc);

R1.setTaskDay(tokens[a+1]);

R1.setEndDate(tokens[a+7]);

R1.setEndMonth(tokens[a+8]);

break;

}

catch (std::out\_of\_range) {

bool value=true;

cout << "error10" << endl;

T1.setErrorDate(value);

recurring=false;

}

}

}

else if((a+7)==tokens.size() &&((tokens[a+6]).at(0)=='x' || (tokens[a+6]).at(0)=='X') && isdigit(tokens[a+6].at(1))) {

vector <string> fromRecTime;

boost::split(fromRecTime,tokens[a+3],boost::is\_any\_of(":"));

vector <string> toRecTime;

boost::split(toRecTime,tokens[a+5],boost::is\_any\_of(":"));

if(fromRecTime.size()==2 && toRecTime.size()==2 && isHourValid(toRecTime[0]) && isHourValid(fromRecTime[0]) && isMinuteValid(toRecTime[1]) && isMinuteValid(fromRecTime[1])) {

T1.setEndHour(toRecTime[0]);

T1.setEndMinute(toRecTime[1]);

T1.setStartHour(fromRecTime[0]);

T1.setStartMinute(fromRecTime[1]);

}

else {

bool value=true;

T1.setTimeError(value);

return T1;

}

R1.setWord(tokens[a+6]);

for(int b=0;b<a;b++) {

desc = desc + originalTokens[b] + " ";

}

T1.setDescription(desc);

R1.setTaskDay(tokens[a+1]);

R1.setRecurring(true);

break;

}

else {

bool value=true;

R1.setRecurringError(value);

}

}

}

else if(tokens[a]=="from") { // add ..... from 23 march 2015 daily.....

R1.setRecWord(tokens[a]);

findRecDate(tokens,a,R1);

if(!R1.getRecurring() && T1.getErrorDate()) {

cout << "yeah " << endl;

return T1;

}

else if(R1.getRecurring()) {

if((a+3)<=tokens.size() && isPeriodValid(tokens[a+3])) {

R1.setPeriod(tokens[a+3]);

if(tokens[a+4]=="until"){

findYear(tokens,(a+4),T1);

if(T1.getErrorDate())

return T1;

if((a+7)==tokens.size()) {

R1.setWord(tokens[a+4]);

if(isYearValid) {

for(int b=0;b<a;b++) {

desc = desc + originalTokens[b] + " ";

}

T1.setDescription(desc);

R1.setEndDate(tokens[a+5]);

R1.setEndMonth(tokens[a+6]);

R1.setRecurring(true);

break;

}

try {

gregDate = stoi(tokens[a+5]);

}

catch (const std::invalid\_argument) {

recurring = false;

break;

}

gregMonth = getMonthNumber(tokens[a+6]);

try {

boost::gregorian::date d(gregYear, gregMonth, gregDate);

ptime now = microsec\_clock::local\_time();

boost::gregorian::date today = now.date();

if(d<today) {

R1.setRecurringError(true);

R1.setRecurring(false);

}

else {

recurring=true;

R1.setRecurring(recurring);

}

for(int b=0;b<a;b++) {

desc = desc + originalTokens[b] + " ";

}

T1.setDescription(desc);

R1.setEndDate(tokens[a+5]);

R1.setEndMonth(tokens[a+6]);

break;

}

catch (std::out\_of\_range) {

bool value=true;

cout << "error10" << endl;

T1.setErrorDate(value);

R1.setRecurring(false);

}

}

else {

R1.setRecurringError(true);

R1.setRecurring(false);

}

}

else if((a+5)==tokens.size() &&((tokens[a+4]).at(0)=='x' || (tokens[a+4]).at(0)=='X') && isdigit(tokens[a+4].at(1))) {

R1.setWord(tokens[a+4]);

for(int b=0;b<a;b++) {

desc = desc + originalTokens[b] + " ";

}

T1.setDescription(desc);

R1.setRecurring(true);

break;

}

}\*/

else {

R1.setRecurringError(true);

return T1;

}

}

}

}

if(a==tokens.size()) {

bool value=false;

R1.setRecurring(false);

}

}