DATALOGGER PROGRAMMING AND SENSORS CONNECTION ADVANCED COURSE

Modulo 1 - CRBasic editor (teoria)

- Michele Mattioni, Simone Sabbatini, Tiziano Sorgi

IR0000032 – ITINERIS, Italian Integrated Environmental Research Infrastructures System (D.D. n. 130/2022 - CUP B53C22002150006) Funded by EU - Next Generation EU PNRR-Mission 4 "Education and Research" - Component 2: "From research to business" - Investment 3.1: "Fund for the realisation of an integrated system of research and innovation infrastructures"

Finanziato dall'Unione europea NextGenerationEU

Ministero dell'Università e della Ricerca

Italiadomani

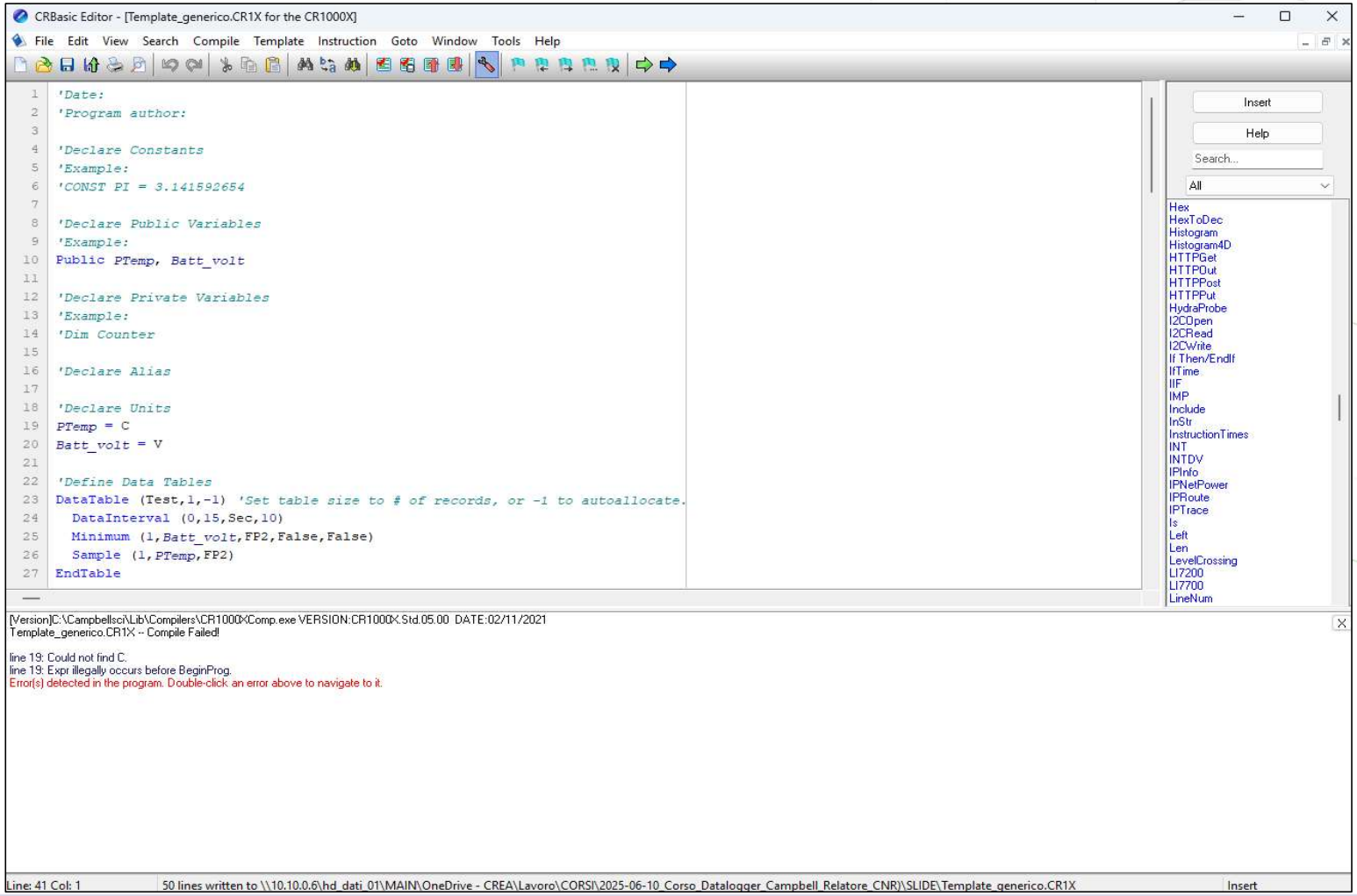
Ambiente di sviluppo CRBasic

L'IDE CRBasic è l'ambiente di sviluppo integrato per la programmazione dei datalogger Campbell Scientific. Permette di scrivere, testare, caricare e monitorare i programmi nel linguaggio CRBasic.

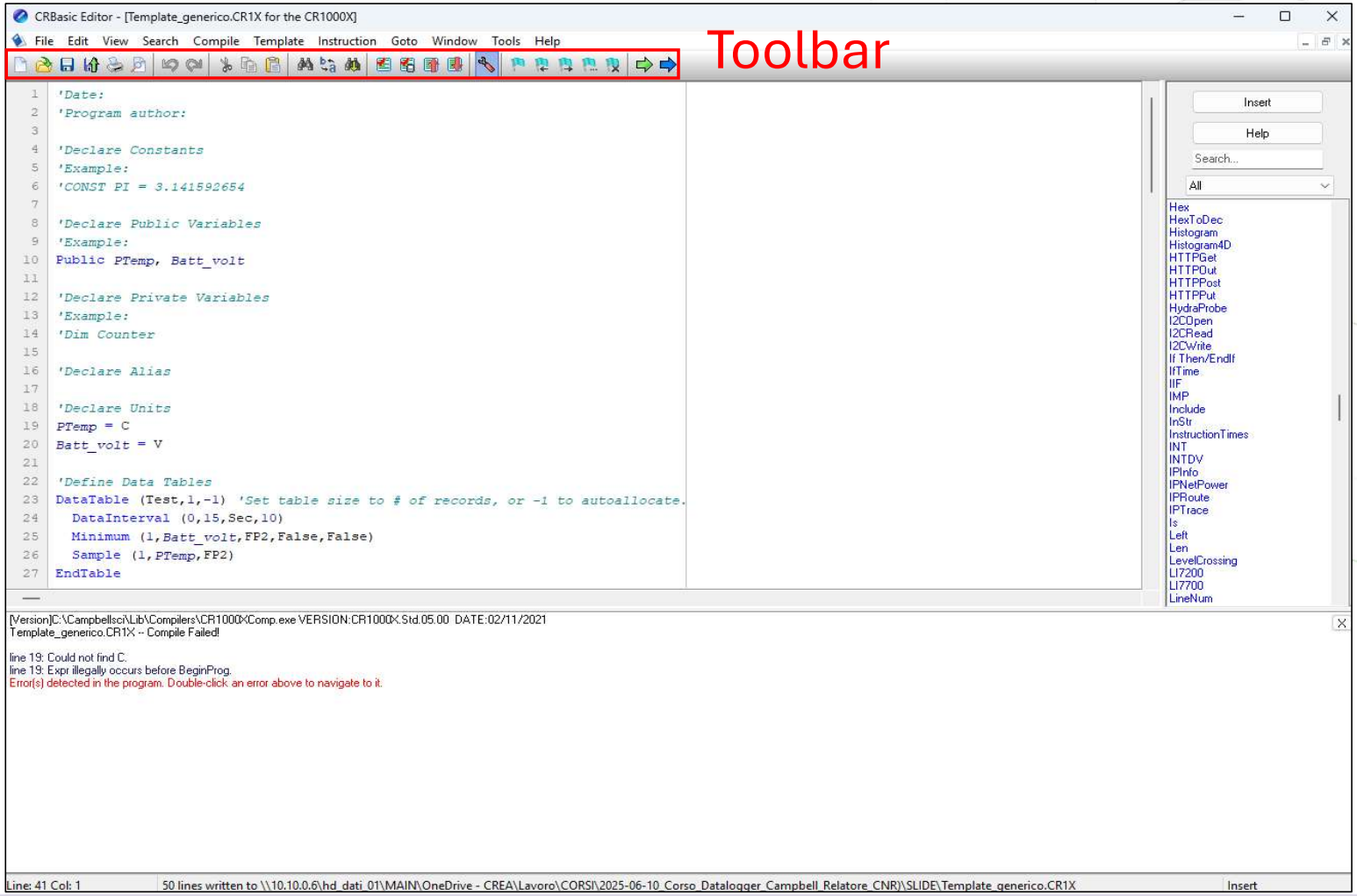
Componenti principali dell'IDE:

- Editor di codice: scrittura programma, indentazione automatica, ricerca.
- Verifica del codice: compilazione offline per il controllo sintattico.
- Output Window: mostra errori, warning e messaggi di debug.
- Toolbar: accesso rapido a funzioni di upload/download, salvataggio, compilazione.
- Template Programs: programmi base per sensori comuni.
- Controllo del datalogger: invio diretto del programma, monitoraggio variabili.

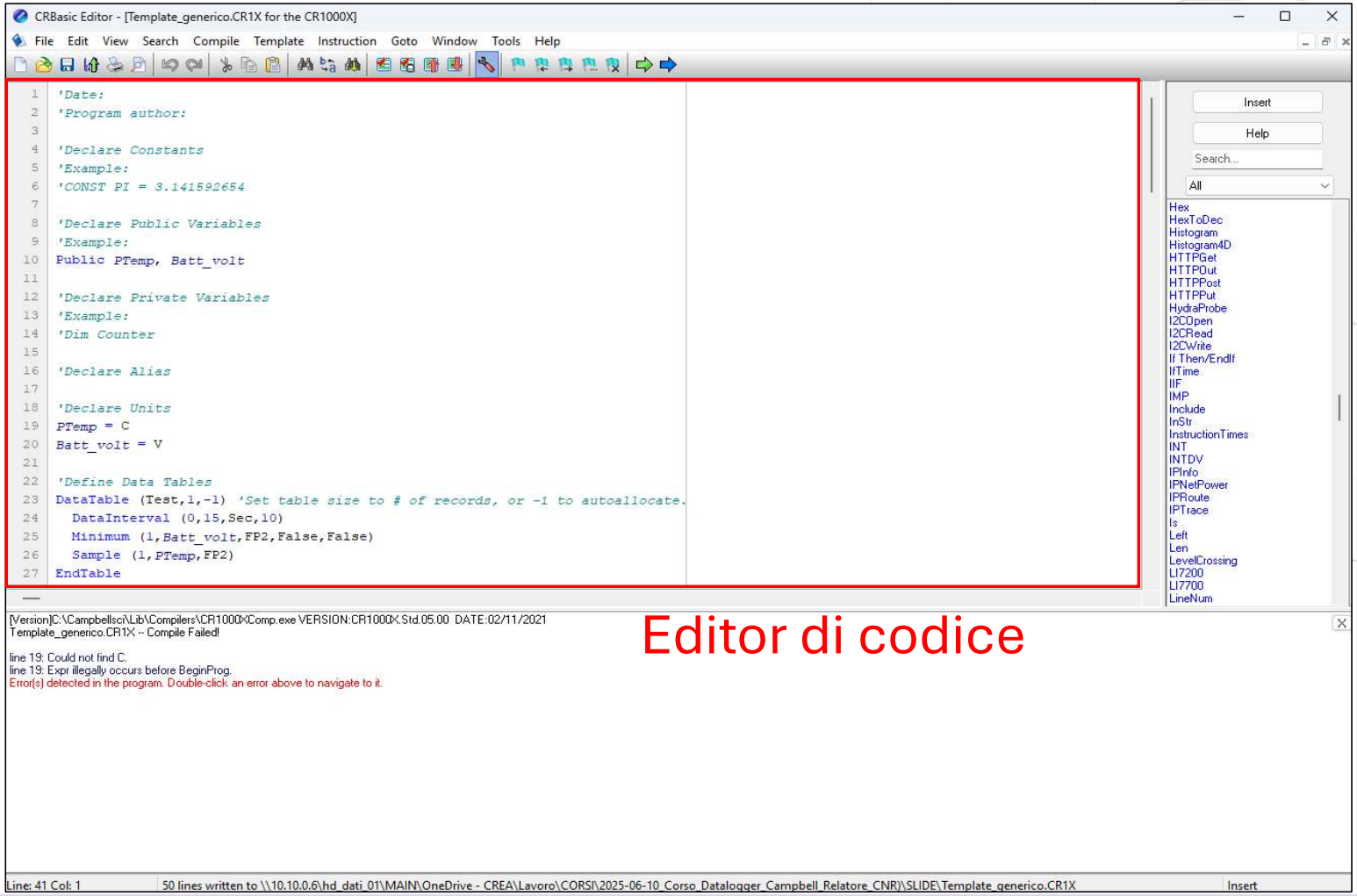
Ambiente di sviluppo CRBasic



Ambiente di sviluppo CRBasic

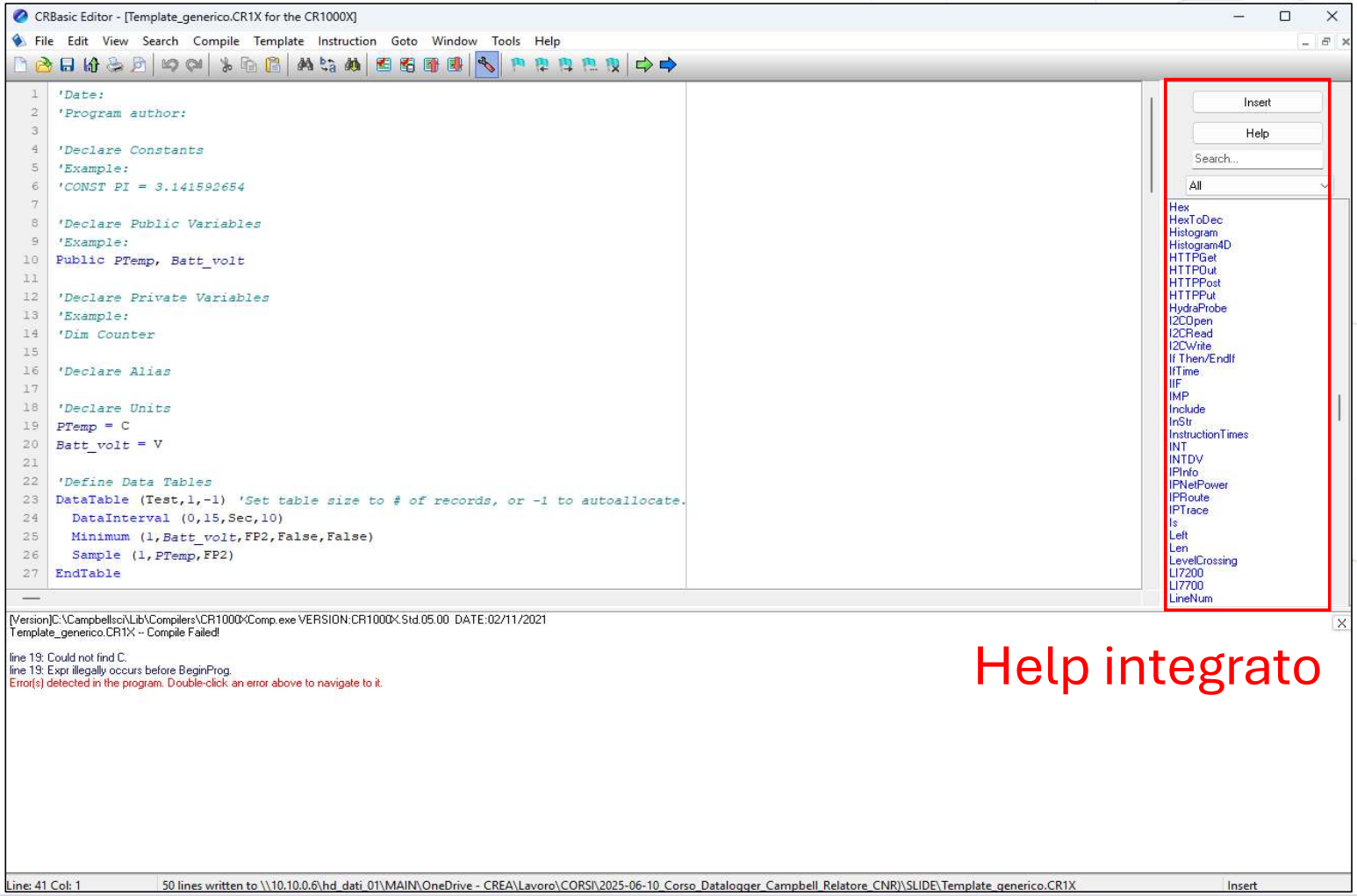


Ambiente di sviluppo CRBasic



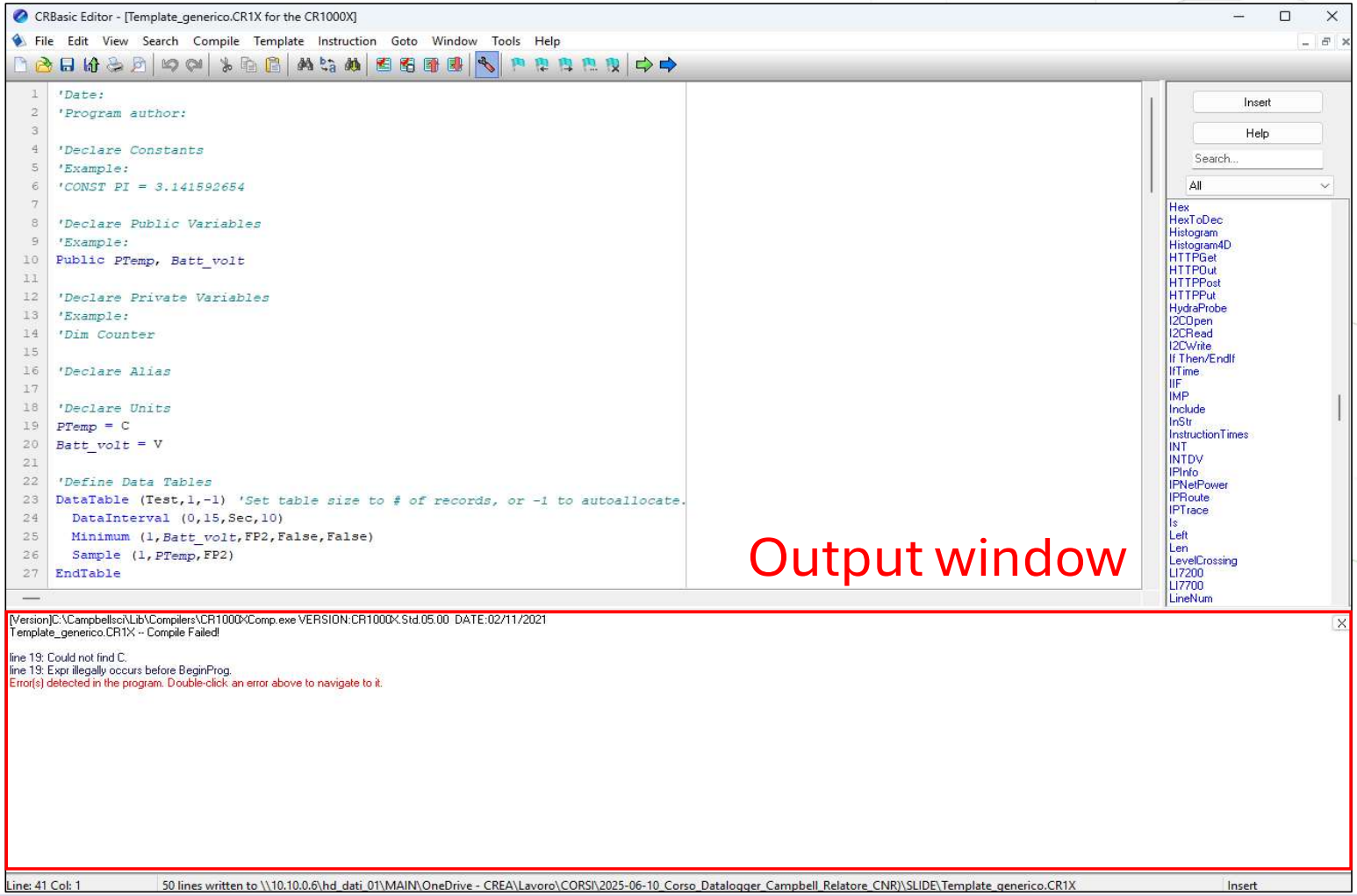
Editor di codice

Ambiente di sviluppo CRBasic



Help integrato

Ambiente di sviluppo CRBasic



Struttura di un programma CRBasic

Dichiarazioni variabili e costanti

Dichiarazioni alias e unità di misura

Definizione Data Table

Definizione Subroutine e Function

Main Program

Main Scan e Slow Sequence

Struttura di un programma - Dichiarazione variabili

- In un programma, le variabili rappresentano uno degli strumenti fondamentali per l'elaborazione e la gestione dei dati. Attraverso le variabili è possibile conservare e manipolare qualsiasi tipo di informazione rilevante per il funzionamento del datalogger: misure acquisite dai sensori, stati logici, risultati di calcoli intermedi, contatori, flag di controllo o dati da registrare.
- Il ruolo principale delle variabili è quello di immagazzinare temporaneamente i valori durante l'esecuzione del programma, in modo che possano essere utilizzati per prendere decisioni, fare confronti, eseguire operazioni matematiche o semplicemente essere salvati nelle tabelle dati per l'analisi successiva.
- Alcune variabili vengono dichiarate per essere accessibili da tutto il programma, e possono anche essere lette o scritte da software esterni come LoggerNet o PC400, ad esempio per configurare i parametri operativi di un sensore o monitorare lo stato del sistema.
- Altre variabili invece hanno un ambito più ristretto e vengono utilizzate solo per funzioni interne, come contare quante volte una certa condizione si verifica o accumulare dei valori per poi calcolare una media.

Struttura di un programma – Tipi di Dichiarazione

«Public»

- **Accessibili globalmente** (tutte le sequenze e subroutine)
- **Visibili e modificabili** anche da LoggerNet o strumenti esterni
- **Persistono nel tempo**
- Ideali per: **output e configurazioni**

«Dim»

- **Accessibili globalmente**, ma **non visibili esternamente**
- **Persistono nel tempo**
- Usate per: **calcoli interni, accumuli, variabili temporanee**

Struttura di un programma – Tipi di Dichiarazione

«Alias»

- **Puntatori o etichette** per accedere a specifici elementi di un array o campo DataTable
- Non occupano memoria, ma migliorano la leggibilità

«Const»

- Costanti definite **una volta sola**
- Non modificabili durante l’esecuzione
- Utili per: valori fissi di configurazione

Struttura di un programma – Tipi di Dichiarazione

«Var»

- Usata **solo all'interno di Function/Subroutine**
- **Locale** alla funzione
- Viene creata e distrutta ad ogni chiamata

«Array»

- Tutti i tipi sopra (Public, Dim, ecc.) possono essere **array** (1D o 2D)

Struttura di un programma – Tipi di Variabile

Tipi di dati disponibili per le variabili:

- **Float:** Numero con virgola mobile a precisione singola
- **Long:** Intero a 4 byte (±2,1 miliardi)
- **Boolean:** Variabile logica (True/False)
- **String:** Sequenza di caratteri (max 127 caratteri)
- **Integer:** Intero a 2 byte (±32.768), usato raramente
- **Double:** Numero con virgola mobile a precisione doppia

Float è il tipo più comunemente usato, specialmente per letture da sensori.

Struttura di un programma – Esempi di dichiarazione

```
Const PI = 3.141592654
Public PT as Float, Volt as Float, AT(5) as Float
Dim Counter

Alias PT = TempPannel
Alias Volt = Battery
Alias AT(1) = AirTemp_1, AT(2) = AirTemp_2, AT(3) = AirTemp_3
Alias AT(4) = AirTemp_4, AT(5) = AirTemp_5

Units PT = °C, AT(1) = °C, AT(2) = °C, AT(3) = °C, AT(4) = °C, AT(5) = °C
Units Volt = V

Sub routine
    Var i as long
EndSub
```

Struttura di un programma – Data Table

- Una DataTable è la struttura utilizzata per registrare e memorizzare i dati acquisiti o calcolati dal datalogger. Ogni datalogger è progettato per raccogliere dati in tempo reale, ma per poterli storicizzare e rendere disponibili al download o all'invio remoto, è necessario che vengano salvati in una o più tabelle. Queste tabelle, appunto chiamate DataTable, sono definite all'interno del programma in modo esplicito, e permettono di configurare in dettaglio quali variabili salvare, a quale intervallo, con quale formato, e con quali modalità di archiviazione.
- La prima parte è la sua dichiarazione, che specifica il nome della tabella, la condizione di attivazione (in genere True se sempre attiva) e la profondità del buffer (cioè quanti record può contenere prima di sovrascrivere i più vecchi).
- Questa dichiarazione segue la sintassi:
  DataTable(Meteo, True, -1)

Struttura di un programma – Data Table

- Dove "Meteo" è il nome della tabella, "True" indica che è sempre attiva, e "-1" significa che verrà utilizzata la massima profondità possibile compatibile con la memoria disponibile del datalogger.
- All'interno della DataTable vengono poi inseriti i comandi che definiscono cosa salvare, che possono essere molto semplici (come Sample, per salvare direttamente una variabile), oppure più elaborati (come Average, Max, Min, Tot, Histogram, ecc.), che permettono di calcolare e archiviare medie, minimi, massimi, sommatorie e altre statistiche in modo automatico.
- Ecco un esempio tipico:

```
DataTable(Meteo, True, -1)
        Sample(1, Temp, FP2)
        Average(1, RH, FP2, False)
        Totalize(1, Rain, FP2, False)
EndTable
```

Struttura di un programma – Data Table

- In questo esempio vengono salvati: la temperatura come campione singolo, l'umidità relativa come media, e la pioggia come sommatoria, tutti nel formato a virgola mobile a 2 byte (FP2) che permette di memorizzare numeri decimali fino alla quarta cifra, altrimenti si possono usare 4 byte (IEEE4) in modo da arrivare a 7 cifre decimali.
- Per rendere effettivo il salvataggio, ogni DataTable deve essere associata a un'istruzione CallTable, che va inserita all'interno di una Scan o SlowSequence. Questa istruzione è quella che comanda il salvataggio effettivo del record secondo la logica specificata
- Inoltre, è possibile controllare la frequenza di salvataggio usando gruppi temporali o condizioni logiche, ad esempio salvare solo ogni 10 minuti o solo se una certa soglia è superata. Questo offre un grande livello di controllo sulla quantità e qualità dei dati archiviati.

Struttura di un programma – Data Table

- Un’altra parte importante delle DataTable è la gestione del formato e del file. Quando si imposta la tabella per l’esportazione su file (con il comando TableFile()), è possibile decidere il nome del file, il formato (TOB1, TOA5, ecc.), la frequenza di scrittura su file e se sovrascrivere o meno i dati esistenti. I Datalogger scrivono i dati su 2 tipi di memorie, quella interna identificata come ‘USR’ (dimensione dipendente dal modello) ed esterna ‘CRD’ (se il Datalogger ha il lettore MicroSD integrato)
- In sintesi, le DataTable sono l’elemento che collega la fase di acquisizione a quella di archiviazione e trasmissione dei dati. Sono completamente configurabili, possono contenere qualunque tipo di elaborazione, e rappresentano una delle parti più potenti e versatili del linguaggio CRBasic. Una buona comprensione delle DataTable è essenziale per strutturare correttamente un programma di acquisizione dati efficiente e robusto.

Struttura di un programma – Data Table

Istruzioni più importanti per un DataTable:

- DataInterval(0, 5, Min, 10): Impone che i dati siano salvati ogni 5 minuti, anche se il Scan è più frequente (es. ogni 10 secondi). 10 è un'opzione avanzata per “allineamento temporale” (può rimanere così nei casi semplici).
- DataEvent(RawBefore, AllarmeIn, AllarmeOut, RawAfter): Salva solo se AllarmeIn cambia da 0 a 1 (evento attivo) e finisce di salvare se AllarmeOut cambia da 0 a 1. RawBefore e RawAfter sono variabili numeriche che restituiranno rispettivamente le ultime righe prima della condizione di Evento e le prime righe dopo la condizione di Evento.
- TableFile (FileName, Opt, MaxF, NumRecs/TimeIntoInterv, Interv, Units, OutSt, LastFName): ‘FileName’ nome del file da esportare, ‘Opt’ definisce il formato del file (15 per il formato ASCII TOA5), ‘MaxF’ numero massimo di file che verranno scritti, se il numero massimo viene raggiunto verranno cancellati i più vecchi, ‘NumRecs/TimeIntoInterv’ (se ‘Interv’ = 0) immettere il numero massimo di Record che verranno scritti sul file, (se ‘Interv’ <> 0) immettere l'ora nell'intervallo (o offset) in cui un file deve essere scritto, ‘Interv’ se <> da 0 i file verranno scritti in base a un intervallo di tempo, ‘Units’ definisce l’unità di tempo su cui si basa l’intervallo di tempo, ‘OutSt’ è una variabile Booleana che assume il valore ‘vero’ quando un file viene scritto ed infine ‘LastFName’ che conserva il nome dell’ultimo file scritto.

Struttura di un programma – Esempio di Data Table

```
' Dichiarazione delle variabili
Public Temp As Float
Public AllarmeIn As Boolean, AllarmeOut As Boolean
Public RawBefore as Long, RawAfter as Long

' DataTable con trigger condizionale, evento, intervallo e salvataggio su file
DataTable(CondizionataEventi, Temp > 30, -1)
  DataInterval(0, 5, Min, 10)                                    ' Forza la registrazione ogni 5 minuti
  TableFile (FileName, Opt, MaxF, NumRecs/TimeIntoInterv, Interv, Units, OutSt, LastFName) ' Scrittura su file
  DataEvent(RawBefore, AllarmeIn, AllarmeOut, RawAfter) ' Salva record solo su stato AllarmeIn e AllarmeOut
  Sample(1, Temp, FP2)                                           ' Salva la temperatura
  Sample(1, StatoAllarme, Boolean1)                              ' Salva lo stato logico
EndTable
```

Struttura di un programma – Formato file di output

- Per quanto riguarda l'istruzione 'TableFile', a seconda del valore che ho sul parametro 'Opt', posso esportare i dati in un determinato formato di file, secondo la seguente tabella (valida per il datalogger CR1000X). Header, Timestamp e Record# inseriranno rispettivamente nel file le intestazioni delle colonne, il Timestamp ed il numero del record. Per le 'Opt' 7, 15, 19 e 35 queste 3 informazioni non saranno scritte.

0 TOB1, Header, TimeStamp, Record#	12 TOA5, TimeStamp, Record#
1 TOB1, Header, TimeStamp	13 TOA5, TimeStamp
2 TOB1, Header, Record#	14 TOA5, Record#
3 TOB1, Header	15 TOA5
4 TOB1, TimeStamp, Record#	16 CSIXML, TimeStamp, Record#
5 TOB1, TimeStamp	17 CSIXML, TimeStamp
6 TOB1, Record#	18 CSIXML, Record#
7 TOB1	19 CSIXML
8 TOA5, Header, TimeStamp, Record#	32 CSIJSON, TimeStamp, Record#
9 TOA5, Header, TimeStamp	33 CSIJSON, TimeStamp
10 TOA5, Header, Record#	34 CSIJSON, Record#
11 TOA5, Header	35 CSIJSON

Struttura di un programma – Sub Routine e Functions

- Nel linguaggio CRBasic, le subroutine e le funzioni sono due strumenti fondamentali per organizzare e semplificare il codice. Entrambe permettono di isolare blocchi di istruzioni che possono essere richiamati più volte nel programma, evitando così la duplicazione del codice e migliorandone la leggibilità e la manutenzione. Tuttavia, pur avendo una struttura simile, servono a scopi leggermente diversi e si usano in contesti differenti.
- Le Sub Routine, dichiarate con le parole chiave Sub … EndSub, sono blocchi di codice autonomi che eseguono una determinata serie di istruzioni, ma non restituiscono un valore. Sono utili quando si vuole eseguire un'operazione, ad esempio acquisire un gruppo di sensori, accendere e spegnere un alimentatore, gestire una comunicazione seriale o eseguire un ciclo di controllo, ma non è necessario ottenere un risultato numerico da utilizzare altrove. Le subroutine possono accedere alle variabili globali del programma, ma possono anche ricevere dei parametri in ingresso, che verranno utilizzati all'interno del loro corpo. Le Sub Routine vengono richiamate nel codice con l'istruzione 'Call' seguita dal nome della Sub Routine

Struttura di un programma – Sub Routine e Functions

- Le funzioni (Function ... EndFunction), invece, sono simili alle subroutine per struttura, ma hanno una caratteristica fondamentale in più, restituiscono un valore. Proprio come nelle funzioni matematiche o in altri linguaggi di programmazione, una funzione riceve degli argomenti in ingresso, esegue una serie di operazioni, e infine fornisce un risultato che può essere assegnato a una variabile o utilizzato direttamente in un'espressione. Questo le rende particolarmente utili per i calcoli: ad esempio, possiamo definire una funzione che calcoli l'indice di comfort termico, o la conversione da volt a gradi Celsius per un sensore.
- Un'altra differenza importante è che le funzioni sono più "pure", nel senso che dovrebbero limitarsi ad accedere ai dati forniti come argomenti ed evitare effetti collaterali (cioè modificare variabili globali). Le subroutine, al contrario, possono manipolare direttamente lo stato del programma, agendo su variabili pubbliche o chiamando altre routine.
- In sintesi, la subroutine è un blocco di codice riutilizzabile che serve per fare qualcosa, mentre la function è un blocco riutilizzabile che serve per calcolare qualcosa e restituire un valore. Usarle correttamente permette di scrivere programmi più ordinati, flessibili e manutenibili, soprattutto quando la complessità del codice aumenta.

Struttura di un programma – Main Program, Scan e Slow Sequence

- Nel linguaggio CRBasic, la sezione denominata comunemente “Main Program” è il cuore operativo del codice, ovvero la parte in cui viene descritta l’attività esecutiva che il datalogger deve svolgere ciclicamente. Questa sezione inizia formalmente con la parola chiave ‘BeginProg’ e termina implicitamente alla fine del programma con la chiave ‘EndProg’. Al suo interno si trovano una o più sequenze di scansione, che definiscono il comportamento periodico del sistema.
- La struttura più comune e fondamentale del Main Program è la scansione principale, dichiarata con l’istruzione ‘Scan’. La ‘Scan’ è un ciclo a tempo fisso, eseguito ripetutamente dal datalogger a intervalli ben definiti, durante il quale vengono effettuate la lettura dei sensori, l’elaborazione dei dati, l’attivazione di eventuali uscite, il salvataggio nelle DataTable, e ogni altra attività che deve essere eseguita in modo regolare e deterministico.
- Per esempio, una Scan(10, Sec, 0, 0) indica che il datalogger eseguirà il ciclo ogni 10 secondi. Tutto il codice incluso tra Scan … NextScan verrà ripetuto esattamente con questa frequenza. Il valore scelto per l’intervallo di scansione (es. 1, 10, 60 secondi, ecc.) dipende dal tipo di misure che si vogliono fare: sensori dinamici come anemometri o celle di carico richiedono scansioni frequenti, mentre sensori ambientali lenti come quelli per temperatura o radiazione possono essere interrogati anche a intervalli più lunghi.

Struttura di un programma – Main Program, Scan e Slow Sequence

- La scansione principale è progettata per essere deterministica e prioritaria, nel senso che il datalogger garantisce che venga eseguita puntualmente, anche a scapito di altri processi. Questo è particolarmente importante in ambito scientifico e industriale, dove la precisione temporale è cruciale.
- Tuttavia, in molti casi, ci sono operazioni che non è necessario eseguire a ogni ciclo della Scan principale, perché sono lente, poco frequenti, o non critiche in termini temporali. Per queste situazioni CRBasic mette a disposizione la possibilità di definire delle 'SlowSequence', ovvero sequenze di scansione indipendenti, che si comportano in tutto e per tutto come delle Scan secondarie, ma con una frequenza tipicamente più bassa e una priorità inferiore.
- Le 'SlowSequence' vengono anch'esse definite con un blocco Scan ... NextScan, ma racchiuso tra SlowSequence ... EndSequence. Ogni 'SlowSequence' può avere un proprio intervallo di esecuzione (ad esempio ogni 1 minuto, ogni 15 minuti o anche ogni ora) e viene gestita parallelamente alla Scan principale, senza bloccarla. Questo permette, ad esempio, di eseguire una comunicazione seriale o un invio dati via FTP ogni ora, senza interferire con le letture ambientali che avvengono ogni 10 secondi.

Struttura di un programma – Main Program, Scan e Slow Sequence

- Un programma può contenere più ‘SlowSequence’, ciascuna con una propria logica. Tuttavia, è importante ricordare che tutte queste sequenze condividono le variabili globali dichiarate con Public o Dim, e quindi il programmatore deve prestare attenzione alla gestione concorrente dei dati. In generale, il datalogger è in grado di gestire bene queste situazioni, ma è buona pratica evitare che due sequenze accedano in scrittura alla stessa variabile nello stesso momento.
- In conclusione, la sezione Main Program in CRBasic è l’area dove si articola la logica esecutiva del sistema. Al suo interno, la Scan principale garantisce la periodicità delle operazioni core, mentre le SlowSequence permettono di gestire attività ausiliarie meno frequenti senza compromettere la precisione del ciclo principale. Questa architettura modulare consente di scrivere programmi flessibili, efficienti e ottimizzati per ogni tipo di applicazione, dal monitoraggio ambientale alla telemetria industriale.

Concetti di programmazione – Operatori

- Quando si parla di programmazione, si intende l'arte di scrivere istruzioni comprensibili per un sistema automatico, come un datalogger o un computer, in modo da fargli eseguire delle operazioni in modo logico, ordinato e ripetibile. Programmare significa quindi descrivere in linguaggio formale un insieme di passaggi che trasformano dati in risultati. Per farlo, è necessario conoscere alcuni concetti fondamentali, che sono comuni alla maggior parte dei linguaggi di programmazione.
- «Operatori matematici e logici»
  Uno dei primi elementi da comprendere è l'uso degli operatori, che servono a eseguire operazioni sui dati. Gli operatori aritmetici, come +, -, *, /, permettono di sommare, sottrarre, moltiplicare o dividere numeri. Esistono anche operatori di confronto, come =, <>, <, >, <=, >=, che servono a verificare se due valori sono uguali, diversi, maggiori o minori l'uno rispetto all'altro. A questi si aggiungono gli operatori logici, come AND, OR, NOT, che permettono di combinare più condizioni e sono fondamentali nelle decisioni del programma. Questi strumenti costituiscono la base del ragionamento computazionale: permettono di eseguire calcoli e valutare situazioni.

Concetti di programmazione – Esempi su operatori

```
' Dichiarazione delle variabili numeriche
Public A, B As Float
Public Somma, Differenza, Prodotto, Quoziente, Potenza As Float
Public Resto As Long

' Dichiarazione delle variabili di confronto
Public Maggiore, Uguale, Diverso, MinoreUguale As Boolean

' Variabili per logica
Public Temp, RH As Float
Public Condizione1, Condizione2, Risultato_AND, Risultato_OR, Risultato_NOT As Boolean

' Variabile logica d'esempio
Public StatoAllarme As Boolean
```

Concetti di programmazione – Esempi su operatori

```
BeginProg
Scan(10, Sec, 0, 0)
   A = 10
   B = 2

   ' Operatori Aritmetici
   Somma = A + B          ' 10 + 2 = 12
   Differenza = A - B     ' 10 - 2 = 8
   Prodotto = A * B       ' 10 * 2 = 20
   Quoziente = A / B      ' 10 / 2 = 5
   Potenza = A ^ B        ' 10^2 = 100
   Resto = 10 MOD 3       ' Risultato: 1 (resto della divisione intera)
```

Concetti di programmazione – Esempi su operatori

```
  ' Operatori di confronto
  Maggiore = A > B          ' True
  Uguale = A = B            ' False
  Diverso = A <> B          ' True
  MinoreUguale = A <= B     ' False

  ' Operatori logici
  If (Temp > 25 AND RH > 70) OR StatoAllarme = True Then
     Risultato_AND = True
  EndIf
 NextScan
EndProg
```

Concetti di programmazione – Istruzioni condizionali e cicli

- Una volta che abbiamo i mezzi per fare operazioni e confronti, entra in gioco uno degli aspetti più importanti della programmazione: le istruzioni condizionali. Queste sono costrutti logici che permettono al programma di prendere decisioni in base a determinate condizioni. La più comune è la struttura If ... Then ... Else, che dice al datalogger: “Se questa condizione è vera, allora fai questo, altrimenti fai qualcos’altro”. Una variante è la Select … Case che invece dice al datalogger: “Nel caso questa condizione vale ‘A’ fai questo, se vale ‘B’ fai questo …. altrimenti fai quest’altro”. Queste istruzioni sono fondamentali per adattare il comportamento del programma alle situazioni reali, come accendere un riscaldatore se la temperatura è troppo bassa, o inviare un allarme se un sensore supera una soglia.
- A fianco delle condizioni, troviamo un altro concetto essenziale: i cicli, o strutture di iterazione. Questi servono per ripetere automaticamente una o più istruzioni finché una certa condizione è verificata o per un numero definito di volte. In CRBasic, la Scan stessa è una forma di ciclo temporizzato: ogni tot secondi, il programma esegue tutto ciò che è racchiuso tra Scan … NextScan. Esistono anche costrutti più generici, come For ... Next o While ... Wend, che permettono di ripetere un blocco di codice, ad esempio per analizzare una serie di valori o attendere una condizione. I cicli sono la chiave per automatizzare le operazioni ripetitive e ridurre la ridondanza del codice.

Concetti di programmazione – Esempi

```
' Variabili numeriche e logiche
Public Valore As Long
Public ContatoreFor, ContatoreWhile, ContatoreDo As Long
Public SommaFor, SommaWhile, SommaDo As Long

Public CondizioneA, CondizioneB As Boolean
Public Scelta As Long

' Variabili di test per risultati
Public RisultatoCond1, RisultatoCond2, RisultatoCase As String * 20

BeginProg
Scan(10, Sec, 0, 0)
```

Concetti di programmazione – Esempi su condizionali

```
' IF...THEN...ELSE
Valore = 12
If Valore > 10 Then
  RisultatoCond1 = "Maggiore di 10"
Else
  RisultatoCond1 = "Minore o uguale"
EndIf

' IF...THEN senza ELSE
CondizioneA = True
CondizioneB = False
RisultatoCond2 = ""
If CondizioneA AND NOT CondizioneB Then
  RisultatoCond2 = "Solo A è vero"
EndIf
' SELECT CASE
Scelta = 2
Select Case Scelta
  Case 1
    RisultatoCase = "Uno"
  Case 2
    RisultatoCase = "Due"
  Case Else
    RisultatoCase = "Altro"
EndSelect
```

Concetti di programmazione – Esempi di ciclo

```
' FOR...NEXT
SommaFor = 0
For ContatoreFor = 1 To 5
  SommaFor = SommaFor + ContatoreFor
Next

' WHILE...WEND
ContatoreWhile = 1
SommaWhile = 0
While ContatoreWhile <= 5
  SommaWhile = SommaWhile + ContatoreWhile
  ContatoreWhile = ContatoreWhile + 1
Wend
```

```
' DO...LOOP UNTIL
ContatoreDo = 1
SommaDo = 0

Do
  SommaDo = SommaDo + ContatoreDo
  ContatoreDo = ContatoreDo + 1
Loop Until ContatoreDo > 10

NextScan
EndProg
```

Concetti di programmazione – Istruzioni condizionali a tempo

- In CRBasic, l'istruzione 'IFTime' viene utilizzata per eseguire un blocco di codice a intervalli regolari, senza dover ricorrere a una SlowSequence. È una delle istruzioni più semplici ma più potenti per la gestione temporale all'interno della Scan principale, e consente di attivare delle azioni basate sull'orologio del datalogger.
- La sintassi generale è:
  IF IFTime(TintoInt, Interval, Units) Then
  dove 'TintoInt' è l'offset che rende vera la condizione ed è legata ad 'Interval' e alla 'Units', se per esempio 'Interval' = 60 e 'Units' = 'min', un 'TintoInt' = a 0 vuol dire che la condizione è vera al minuto 0 di ogni ora, se è 5 al quinto minuto di ogni ora. Nel caso l'istruzione fosse: IFTime(5, 30, min) Then, la condizione è vera al quinto minuto di ogni mezzora.
- Una variante è la IF TimeIsBetween (BeginTime, EndTime, Interval, Units) dove la condizione è vera tra BeginTime e EndTime

Concetti di programmazione – Esempio di IFTime e TimeIsBetween

```
BeginProg
  Scan (1,Sec,3,0)
    PanelTemp (PTemp,15000)
    TCDiff (TCTemp,1,mv200C,1,TypeT,PTemp,True,0,15000,1.0,0)

    IF IFTime (0,60,min) Then              'Esegue la SendData ad ogni primo minuto dell'ora
      SendData (ComRS232,0,4094,Test)
    EndIf
    IF TimeIsBetween(3,6,60,min)
      SendData (ComRS232,0,4094,Test)      'Esegue la SendData tra il minuto 3 e 6 di ogni ora
    EndIf

NextScan
EndProg
```

Concetti di programmazione – Conclusioni

- Oltre a questi elementi, man mano che si acquisisce maggiore esperienza, si scopre quanto sia importante organizzare il codice in modo modulare e riutilizzabile. Questo si fa utilizzando subroutine e funzioni, che sono blocchi di codice indipendenti, pensati per eseguire compiti specifici. Le funzioni possono anche restituire un valore. Questo approccio permette di scrivere programmi più ordinati, più facili da leggere e da mantenere, e soprattutto consente di evitare di scrivere più volte lo stesso codice.
- Un altro principio fondamentale della programmazione è quello della gestione delle variabili, che abbiamo già approfondito in precedenza. È importante sapere quali variabili sono visibili e in quali parti del programma, quando vengono inizializzate e se i loro valori devono essere conservati nel tempo o solo temporanei.
- Infine, ogni programma, anche il più semplice, deve sempre tenere conto di una cosa: l'esecuzione sequenziale. Il codice viene eseguito nell'ordine in cui è scritto, salvo quando una struttura condizionale o un ciclo lo modifica. È quindi essenziale pianificare bene l'ordine delle istruzioni, perché un errore nella sequenza può produrre risultati inattesi, anche se le singole righe sono corrette.

Concetti di programmazione – Consigli utili

- Durante la scrittura di un programma, è molto importante rispettare la cosiddetta indentazione. L'indentazione di un programma è il modo in cui il codice viene visivamente organizzato attraverso rientri (spazi o tabulazioni) all'inizio delle righe. Non influenza direttamente il funzionamento del codice per il compilatore (in CRBasic), ma è fondamentale per la leggibilità, la chiarezza e la manutenzione di un programma.
- In pratica, l'indentazione serve a mostrare graficamente la struttura logica del codice, indicando quali istruzioni appartengono a un blocco (es. a un If, a un ciclo For, a una Scan, a una Sub).

Concetti di programmazione – Consigli utili

- Esempio corretto con indentazione:

```
Scan(10, Sec, 0, 0)
   If Temp > 30 Then
      PortSet(1, 1)
   EndIf
NextScan
```

- Esempio senza indentazione

```
Scan(10, Sec, 0, 0)
If Temp > 30 Then
PortSet(1, 1)
EndIf
NextScan
```

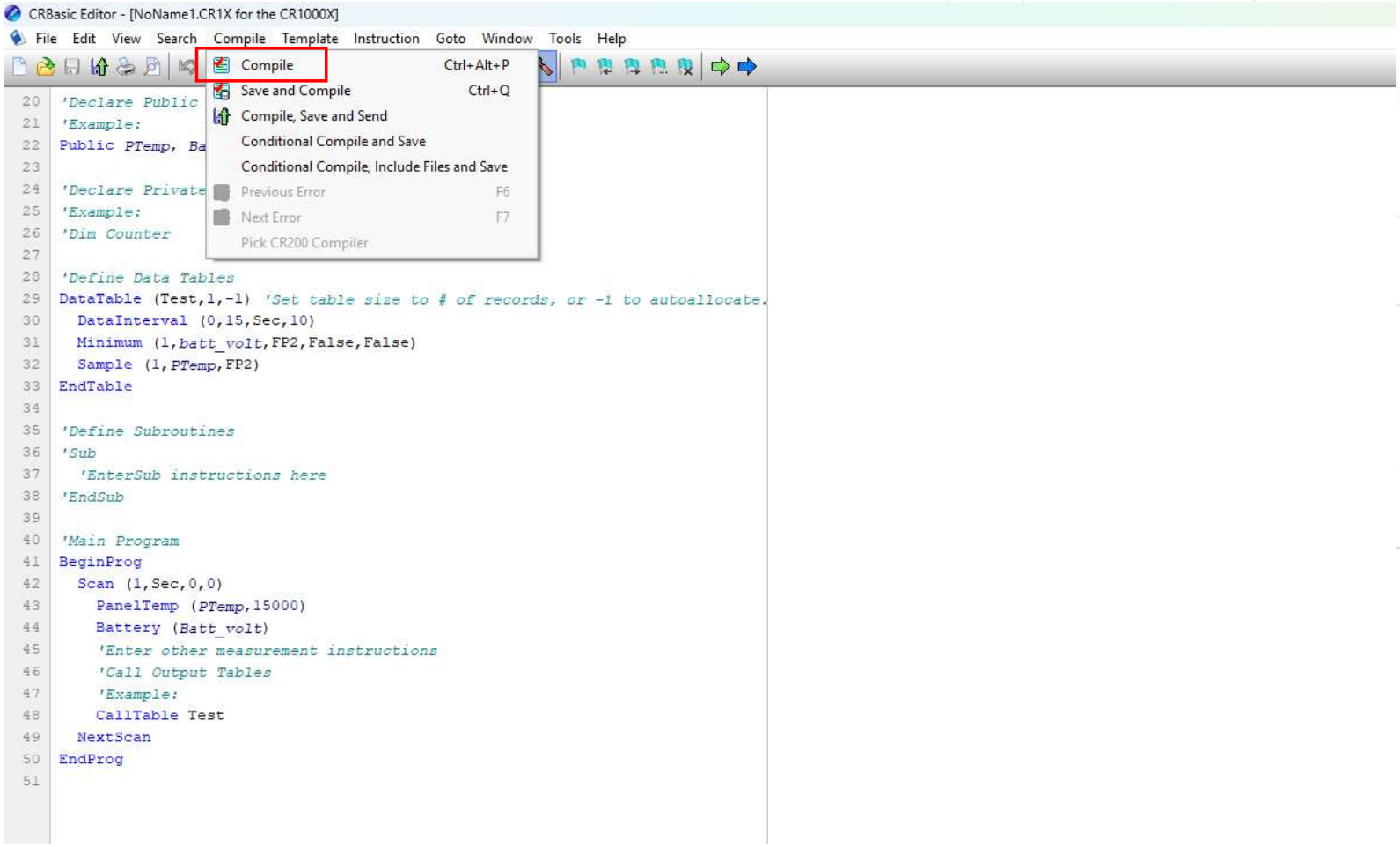
- Entrambi gli esempi sono perfettamente funzionanti ma il primo è decisamente più facile da leggere.
- Un altro consiglio importantissimo è l'uso di righe di commento per descrivere istruzioni complesse o passaggi del programma. In CRBasic se un qualsiasi testo viene scritto dopo il simbolo dell'apice ', il testo viene convertito in commento e non sarà più visibile al codice, il colore del testo scritto come commento è verde, in modo che sia facilmente riconoscibile.

Concetti di programmazione – Consigli utili

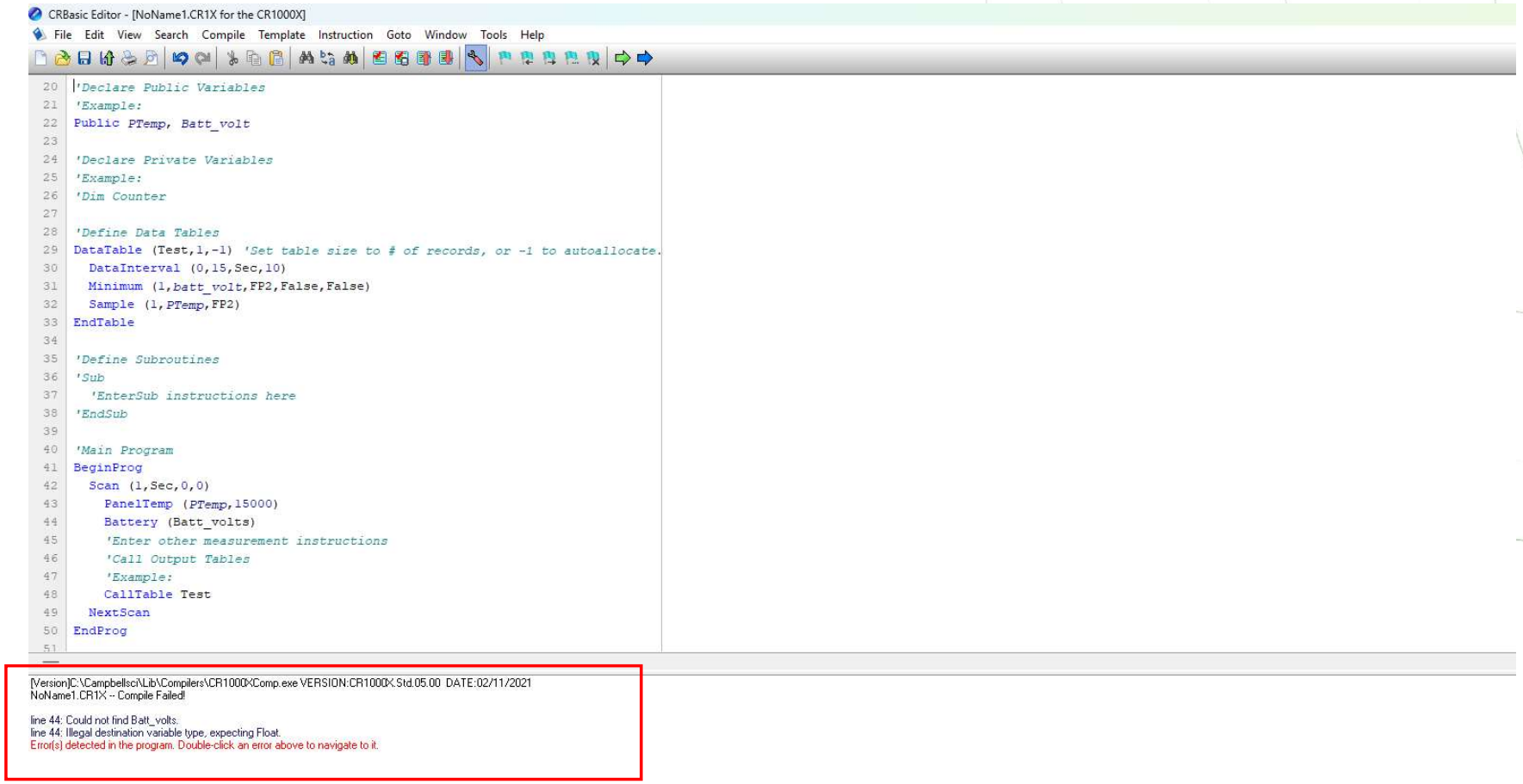
```
'apertura porta RS232 collegata al LI-840
'  SerialOpen (ComRS232,115200,0,150,1000,0)
'  SerialFlush(ComRS232)
SerialOpen(ComRS232,9600,3, 150,1000) ' Aprire la porta seriale (porta 2, ba
'  Delay(0,1,mSec) ' Attendere per l'inizializzazione della porta seriale
SerialFlush(ComRS232) ' Pulire il buffer della porta seriale

'Apertura porta I2C collegata ad Arduino
PortPairConfig (C7,1)
I2COpen (C7,100)

'Configure the datalogger as a Modbus Slave
ModbusSlave (503,115200,1,ModbusRegST(),ModbusCoil)

PakBusClock (1)

MFC_Volt = 5000

Scan (1,Sec,5,0)
  PanelTemp (PTemp,15000)
  Battery (Batt)

  'ExciteV (Vx1,MFC_Volt,0)     'Setta il flusso del Mass Flow Controller a m

  Call Zero_MSB_Display
  Timestamp_ST = Year&Month_MM&DOM_DD&Hour_HH&Minute_MM
  Timestamp_IC = Year&Month_MM&DOM_DD&Hour_HH&Minute_MM&Second_SS
  NomeFile = Year+Month_MM+DOM_DD

  Sprintf(TIME_STAMP_ACT,"%04u%02u%02u",YYYY,MM,DD)
  If TS_IC="" Then TS_IC=TIME_STAMP_ACT
  If TS_ST="" Then TS_ST=TIME_STAMP_ACT
  If TS_HL="" Then TS_HL=TIME_STAMP_ACT

  'Timing Storage ICOS
  If TimeIsBetween (1,37,3600,sec) OR TimeIsBetween (300,337,3600,sec) OR Tin
    NLiv = 1
    NLiv_lb = "Liv_21m"
  EndIf

  If TimeIsBetween (38,74,3600,sec) OR TimeIsBetween (338,374,3600,sec) OR Ti
    NLiv = 2
    NLiv_lb = "Liv_17m"
  EndIf

  If TimeIsBetween (75,112,3600,sec) OR TimeIsBetween (375,412,3600,sec) OR T
    NLiv = 3
    NLiv_lb = "Liv_12m"
  EndIf

  If TimeIsBetween (113,149,3600,sec) OR TimeIsBetween (413,449,3600,sec) OR
    NLiv = 4
    NLiv_lb = "Liv_9m"
  EndIf
```

```
CodeMux_ST(4) = "011"
CodeMux_ST(5) = "100"
SerialOpen(ComRS232,9600,3, 150,1000) ' Aprire la porta seriale (porta 2
SerialFlush(ComRS232) ' Pulire il buffer della porta seriale
PortPairConfig (C7,1)
I2COpen (C7,100)
ModbusSlave (503,115200,1,ModbusRegST(),ModbusCoil)
PakBusClock (1)
MFC_Volt = 5000
Scan (1,Sec,5,0)
PanelTemp (PTemp,15000)
Battery (Batt)
Call Zero_MSB_Display
Timestamp_ST = Year&Month_MM&DOM_DD&Hour_HH&Minute_MM
Timestamp_IC = Year&Month_MM&DOM_DD&Hour_HH&Minute_MM&Second_SS
NomeFile = Year+Month_MM+DOM_DD
Sprintf(TIME_STAMP_ACT,"%04u%02u%02u",YYYY,MM,DD)
If TS_IC="" Then TS_IC=TIME_STAMP_ACT
If TS_ST="" Then TS_ST=TIME_STAMP_ACT
If TS_HL="" Then TS_HL=TIME_STAMP_ACT
If TimeIsBetween (1,37,3600,sec) OR TimeIsBetween (300,337,3600,sec) OR
NLiv = 1
NLiv_lb = "Liv_21m"
EndIf
If TimeIsBetween (38,74,3600,sec) OR TimeIsBetween (338,374,3600,sec) OR
NLiv = 2
NLiv_lb = "Liv_17m"
EndIf
If TimeIsBetween (75,112,3600,sec) OR TimeIsBetween (375,412,3600,sec) O
NLiv = 3
NLiv_lb = "Liv_12m"
EndIf
If TimeIsBetween (113,149,3600,sec) OR TimeIsBetween (413,449,3600,sec)
NLiv = 4
NLiv_lb = "Liv_9m"
EndIf
If TimeIsBetween (150,187,3600,sec) OR TimeIsBetween (450,487,3600,sec)
NLiv = 5
NLiv_lb = "Liv_6m"
EndIf
If TimeIsBetween (188,224,3600,sec) OR TimeIsBetween (488,524,3600,sec)
NLiv = 6
NLiv_lb = "Liv_3m"
EndIf
If TimeIsBetween (225,262,3600,sec) OR TimeIsBetween (525,562,3600,sec)
NLiv = 7
NLiv_lb = "Liv_1m"
EndIf
If TimeIsBetween (263,299,3600,sec) OR TimeIsBetween (563,599,3600,sec)
NLiv = 8
NLiv_lb = "Liv_05m"
EndIf
```

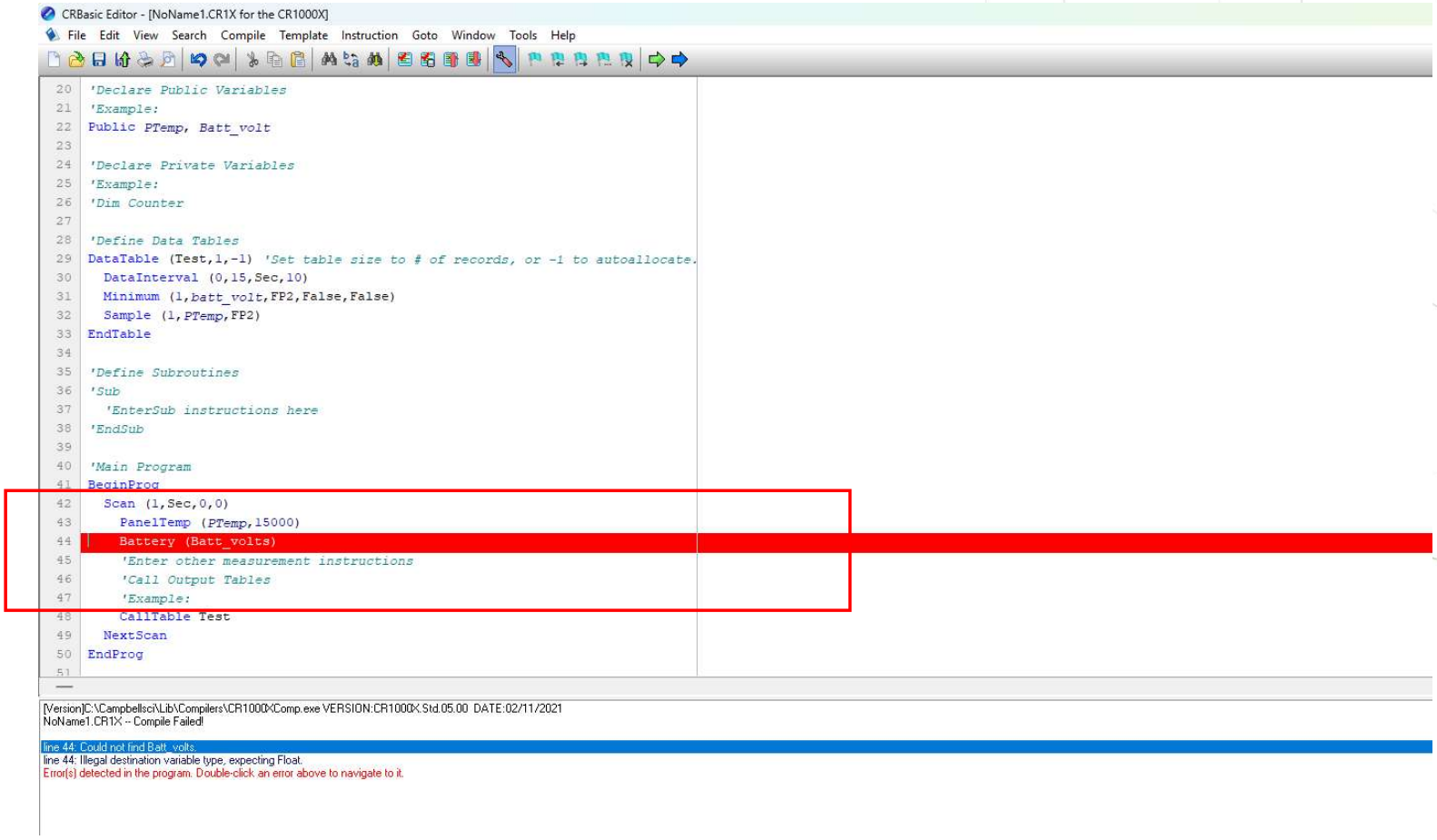
Compilazione programma, verifica errori ed invio.



Compilazione programma, verifica errori ed invio.



Compilazione programma, verifica errori ed invio.

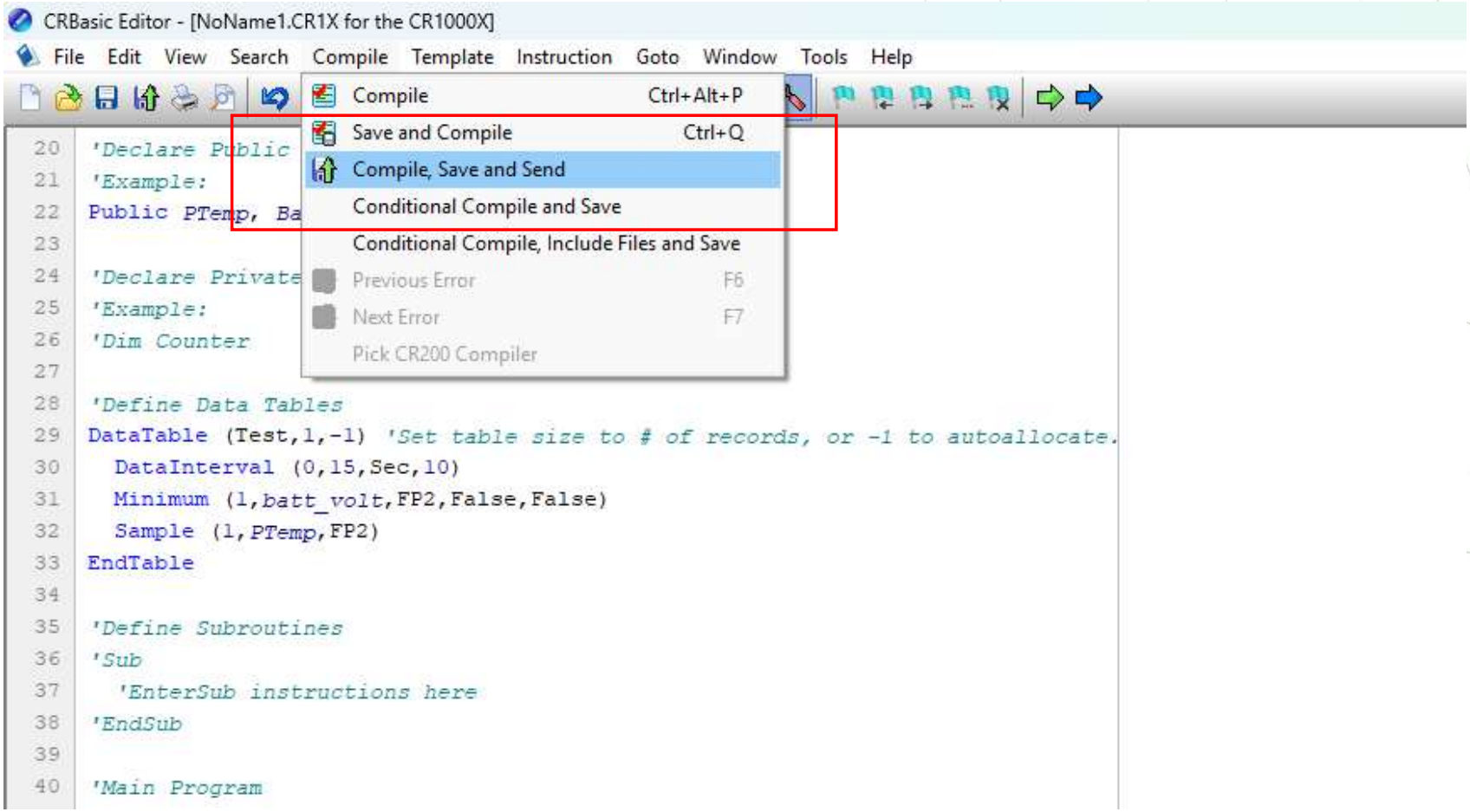


Compilazione programma, verifica errori ed invio.

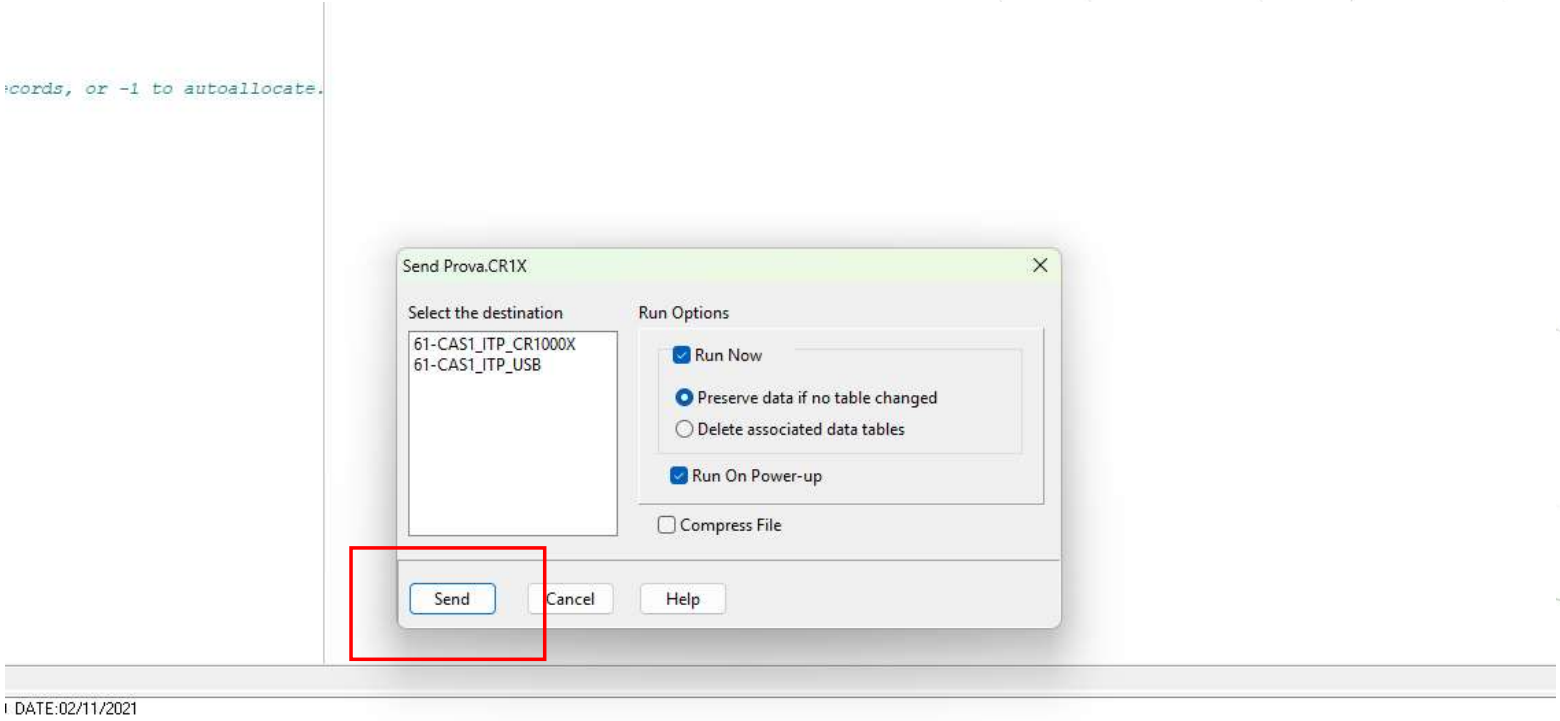
```
'Call Output Tables
'Example:
CallTable Test
NextScan
EndProg
```

```
[Version]C:\Campbellsci\Lib\Compilers\CR1000XComp.exe VERSION:CR1000X.Std.05.00 DATE:02/11/2021
NoName1.CR1X -- Compiled in PipelineMode.
ProgSignature = 41535.
```

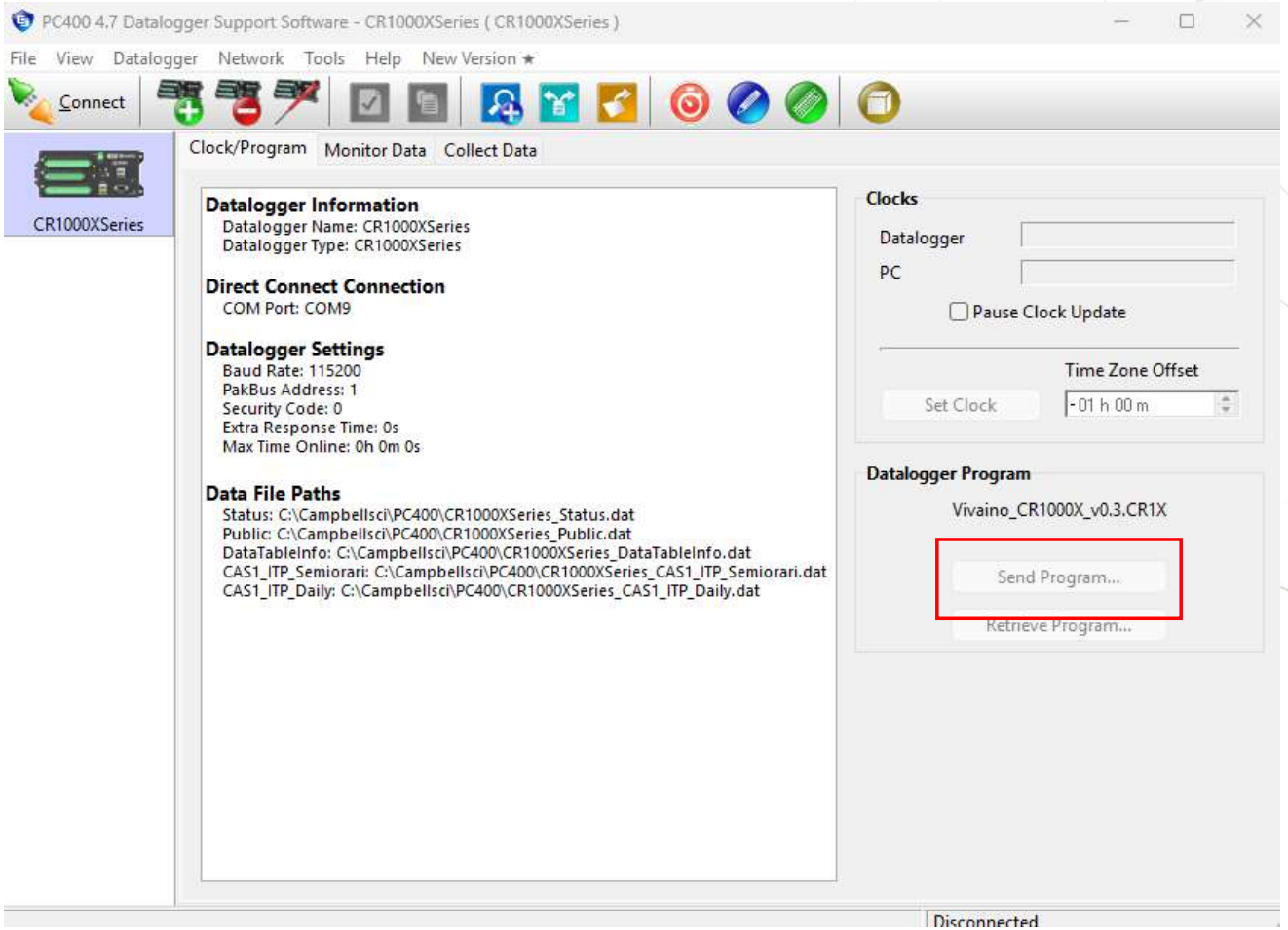
Compilazione programma, verifica errori ed invio.



Compilazione programma, verifica errori ed invio.



Compilazione programma, verifica errori ed invio.



THANKS!

IR0000032 – ITINERIS, Italian Integrated Environmental Research Infrastructures System
(D.D. n. 130/2022 - CUP B53C22002150006) Funded by EU - Next Generation EU PNRR-Mission 4 “Education and Research” - Component 2: “From research to business” - Investment 3.1: “Fund for the realisation of an integrated system of research and innovation infrastructures”

Finanziato dall'Unione europea NextGenerationEU
Ministero dell'Università e della Ricerca
Italiadomani