

```
TMLElement ptr1 =  
("a[href='https://www.kinderc.  
ptr1.className = "link";  
ptr1.style["color"] = "violet";  
  
typedef HTMLMediaElement Player;  
  
fetch("/songs-endpoint.php", {"POST",  
{\ "reqInfo\":"\ "songs\"}"}))  
.then([](Request& response) {  
    if(response.status == 200) {  
  
        var objSong = response.JSON();  
        int songNumber;  
  
        var  
HTMLMediaElement
```

#kinderc th"];

Sviluppare WebApp funzionali utilizzando C++.

```
objSong[0]["audioURL"];  
audioPlayerPtr->attributes["src"] =  
} else $("dialog.errorbox").innerHTML = R"(  
<h1>Sembra che ci sia un errore.</h1>  
<br />  
<div>  
    La canzone che hai richiesto pot  
    essere più disponibile. Controlla oppure  
</div>  
on essere più tardi.
```

Sommario

Introduzione	1
Che cos'è KinderC	1
Premesse per il programmatore	2
Link Utili	2
Repository GitHub del progetto	2
Compilatore CLANG	2
Impostazione dell'ambiente di lavoro	2
Struttura di un'applicazione KinderC	2
Hello World	3
Strutturazione del file sorgente	4
Tipi di dato	4
Tipi di dato fondamentali	4
Tipi di dato aggiuntivi	5
Tipizzazione automatica delle variabili	5
Importazione ed esportazione dei metodi	5
La macro <code>imported</code>	5
La macro <code>exported</code>	6

Introduzione

Che cos'è KinderC

KinderC non è altro che una libreria per il linguaggio C++ che permette, attraverso l'utilizzo del compilatore CLANG, di realizzare siti e applicazioni web snelle e performanti. Il codice scritto dal programmatore viene compilato in un file binario in formato `wasm`, utilizzando la tecnologia `WebAssembly`, che viene poi incluso in una pagina HTML e mandato in esecuzione al suo caricamento. Presenta alcuni vantaggi rispetto all'utilizzo di JavaScript standard, che qui elenchiamo:



- **Velocità:** Essendo compilata, un'applicazione KinderC tende ad essere più veloce rispetto ad una WebApp scritta in JavaScript standard.
- **Efficienza:** Servendosi di un linguaggio a tipizzazione forte, il programmatore può decidere di allocare variabili grandi quanto meglio crede. Può inoltre allocare la memoria in maniera dinamica.
- **Sicurezza:** Il codice scritto viene compilato in un file `WebAssembly` binario. Risulta quindi abbastanza complesso andare a decompilarlo per ritornare al codice sorgente, cosa che rappresenta un vantaggio in termini di sicurezza.
- **Semplicità:** Essendo JavaScript un linguaggio C-like, è facile imparare a scrivere codice utilizzando KinderC. Inoltre, tantissimi metodi richiamano le librerie C standard oppure i metodi propri di JavaScript client-side.

Premesse per il programmatore

Il programmatore che intende imparare a utilizzare la libreria dovrebbe avere queste conoscenze di base:

- **Conoscenza del linguaggio C/C++** e della sua sintassi, almeno a livello superficiale.
- **Conoscenza di HTML, CSS, JavaScript.**

Se così non fosse, si consiglia di consultare guide sul linguaggio C (ed eventualmente C++, se si intende programmare a oggetti) prima di proseguire.

Link Utili

Repository GitHub del progetto

Questa guida, insieme al file header e alle implementazioni di tutti i metodi della libreria, sono pubblicamente disponibili su GitHub a questo link: <https://github.com/nboano/kinderc>

Compilatore CLANG

Per compilare i nostri codici, ci serviremo del compilatore clang, scaricabile a questo link:

<https://releases.llvm.org/download.html>

Al fine che tutto funzioni completamente, è necessario che l'eseguibile di clang sia aggiunto al PATH di sistema.









Impostazione dell'ambiente di lavoro

Per sviluppare utilizzando la libreria è necessario scaricarla dal repo GitHub, il cui link si trova nel paragrafo precedente. Fatto ciò, è necessario anche scaricare e installare il compilatore clang.

N.B. Si consiglia di clonare il repository in una cartella di facile accesso, specie per semplificare le operazioni di inclusione della libreria e di compilazione da terminale. In tutti gli esempi di questo manuale, il repository sarà stato clonato nella cartella **D:\kinderc**. Si adattino i comandi in relazione a dove si è scelto di collocare il file header e lo script per la compilazione.

Per scrivere codice è possibile utilizzare un qualsiasi editor di testo. Tuttavia, si raccomanda l'uso di **Visual Studio Code**, particolarmente leggero e con un'evidenziazione della sintassi ricca e capibile.

Nell'immagine è possibile vedere i file della libreria, che dovrebbero essere presenti sulle vostre macchine prima di iniziare a lavorare.

 .git	20/11/2022 21:34	Cartella di file	
 .vs	20/11/2022 21:34	Cartella di file	
 code	19/11/2022 02:22	Cartella di file	
 .gitattributes	16/11/2022 15:23	File GITATTRIBUTES	3 KB
 .gitignore	16/11/2022 15:23	File GITIGNORE	7 KB
 kccompil.bat	17/11/2022 21:45	File batch Windows	1 KB
 kinderc.hpp	20/11/2022 21:20	C/C++ Header	33 KB
 kinderc.js	19/11/2022 11:14	JavaScript File	2 KB

Struttura di un'applicazione KinderC

La WebApp di compone fondamentalmente di tre parti:

- **UN FILE HTML**, di solito nominato *index.html*, che contiene la struttura della pagina, e che nel tag head presenta un riferimento agli altri file.
- **UNO SCRIPT *kinderc.js***, soprannominato “glue code”. Esso si trova nella libreria da voi scaricata e ha il compito di mettere in comunicazione l’interprete JavaScript, e quindi il DOM stesso, con il compilato da voi scritto.

La cosa più comoda per includere il *kinderc.js* è utilizzare la CDN:

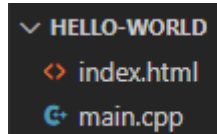
<https://cdn.jsdelivr.net/gh/nboano/kinderc/kinderc.js>

- **UN FILE SORGENTE *main.cpp***, che verrà compilato in un file binario *main.wasm*, che contiene effettivamente il codice.

Hello World

Cominciamo quindi con un primo esempio, un classico Hello World.

Creiamo una cartella sul nostro computer a cui possiamo accedere da browser, utilizzando un WebServer a nostra scelta. È importante aprire le applicazioni passando da un WebServer e non semplicemente cliccando sul file *index.html*, in quanto il browser non permette di richiedere il file *wasm* quando aperto così.



All’interno della cartella, creiamo un nuovo file *index.html*, insieme ad un file *main.cpp*.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>KinderC Hello World</title>

  <script src="https://cdn.jsdelivr.net/gh/nboano/kinderc/kinderc.js"></script>
  <assembly src="main.wasm"></assembly>
</head>
<body>

</body>
</html>
```

Nel file HTML devono essere specificati il percorso del futuro file *wasm* compilato, e anche quello del file *kinderc.js*, come da esempio.

Creiamo quindi anche il file sorgente, *main.cpp*, su cui andremo a codificare il nostro primo esempio.

```
#include "D:\kinderc\kinderc.hpp"

int main() {
    printf("<h1>Hello World!</h1>");
}
```

Nel file sorgente C++ includiamo la libreria precedentemente scaricata.

Ora è il momento di compilare. Apriamo un terminale (anche quello interno di VSCode), e spostiamoci nella cartella del progetto.

Lanciamo il seguente comando per compilare:

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

Windows PowerShell
Copyright (C) Microsoft Corporation. Tutti i diritti riservati.

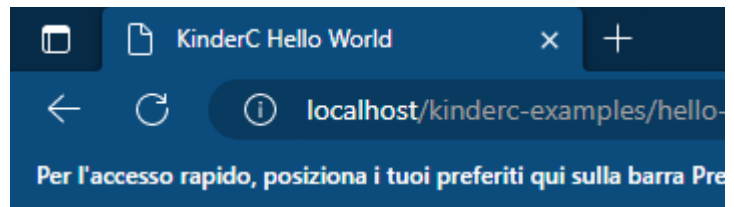
Installa la versione più recente di PowerShell per nuove funzionalità e miglioramenti. https://aka.ms/PSWindows

PS D:\IIS\www\kinderc-examples\hello-world> D:\kinderc\kccompile main.cpp main.wasm
PS D:\IIS\www\kinderc-examples\hello-world>

```

Ovviamente il comando va adattato a seconda di dove avete deciso di collocare il file *kccompile.bat*.

Se non ci sono errori, il tutto dovrebbe funzionare.



Hello World!

Strutturazione del file sorgente

Il file sorgente, ovvero quello che viene compilato da terminale, deve includere obbligatoriamente la libreria all'inizio per poter funzionare. È possibile includere file e librerie esterne, come in un ambiente C standard, utilizzando una *#include*.

Appena l'albero DOM viene caricato in memoria (cioè, appena la pagina viene caricata) il metodo *int main()* viene mandato in esecuzione. Proprio come in un'applicazione console, quindi, è necessario inserire tutto il codice che vogliamo venga eseguito all'interno del metodo principale.

Tipi di dato

Come in ogni linguaggio a tipizzazione forte, anche in C e C++ abbiamo una serie di tipi dato diversi. Oltre ai tipi dato di sistema, utilizzando KinderC ne vengono aggiunti altri due.

Tipi di dato fondamentali

NOME DEL TIPO	DIMENSIONE	DESCRIZIONE	FLAG per <i>String::Format</i>
<i>char</i>	1 byte	Rappresenta un carattere ASCII.	%c
<i>wchar_t</i>	2 byte	Rappresenta un carattere Unicode.	Da Implementare
<i>short</i>	2 byte	Rappresenta un intero su 16bit.	%i
<i>int</i>	4 byte	Rappresenta un intero su 32bit.	%i
<i>long</i>	8 byte	Rappresenta un intero su 64 bit.	%i

<code>float</code>	4 byte	Rappresenta un numero decimale.	%f
<code>double</code>	8 byte	Come il float, ma con prec. doppia.	%d

Tipi di dato aggiuntivi

NOME DEL TIPO	DIMENSIONE	DESCRIZIONE
<code>string</code>	4 + N byte	Rappresenta una sequenza di N <code>char</code> allocata in heap.
<code>object</code>	8 byte	Rappresenta un puntatore a un oggetto JavaScript di qualunque tipo.

Tipizzazione automatica delle variabili

Generalmente, quando dichiariamo una variabile, è necessario specificare il suo tipo.

```
int n = 15;
const double PI = 3.14159;
```

Tuttavia, a seconda delle nostre preferenze, è anche possibile ometterlo, lasciando tipizzare il sistema. Le due scritture presentate sotto sono assolutamente equivalenti.

```
var mStr = "Ciao a tutti";
auto str2 = "Seconda stringa";
```

Importazione ed esportazione dei metodi

Le due macro presentate permettono di esportare dei metodi e richiamarli da JavaScript oppure di importare metodi JavaScript esistenti.

La macro `imported`

Va messa nel prototipo del metodo per indicare che viene importato da JavaScript. Ecco un esempio di metodo che ritorna il timestamp UNIX:

```
#include "D:\kinderc\kinderc.hpp"

imported unsigned long ottieniTempo();

int main() {
    printf("Secondi passati dal 1 GEN 1970: %i", ottieniTempo());
}
```

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
```

```

<title>KinderC Hello World</title>

<script src="https://cdn.jsdelivr.net/gh/nboano/kinderc/kinderc.js"></script>
<assembly src="main.wasm"></assembly>

<script>
    env.ottieniTempo = function() {
        return Date.now() / 1000;
    }
</script>
</head>
<body>

</body>
</html>

```

Secondi passati dal 1 GEN 1970: 1669063428

La macro `exported`

Al contrario della precedente, la macro `exported` rende un metodo implementato in C++ visibile da JavaScript, e dalla pagina in generale.

```

#include "D:\kinderc\kinderc.hpp"

exported void esponenziale(int base, int esponente) {
    long risultato = base;
    for (int i = 0; i < esponente - 1; i++) risultato *=
base;

    $("#lblRisultato").innerText = String::Format("%i", risultato);
}

int main() {}

```


Calcola

Il risultato è: 1594323

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>KinderC Hello World</title>

    <script src="https://cdn.jsdelivr.net/gh/nboano/kinderc/kinderc.js"></script>
    <assembly src="main.wasm"></assembly>

</head>
<body>
    <input type="number" placeholder="Base" id="txtBase">
    <br>
    <input type="number" placeholder="Esponente" id="txtEsponente">
    <br><br>

```

```
<button onclick="esponenziale(document.querySelector('#txtBase').valueAsNumber,  
document.querySelector('#txtEsponente').valueAsNumber)">Calcola</button>  
<br><br>  
Il risultato &egrave;; <b id="lblRisultato"></b>  
</body>  
</html>
```

NOTA BENE I metodi importati/esportati possono avere solo parametri NUMERICI, interi o a virgola mobile (non è possibile passare stringhe o oggetti).