VizCon User's Guide

UCF VizCon Team

March 2022

1 History

In 2004, Professor Mordechai (Moti) Ben-Ari of the Weizmann Institute of Science released an integrated development environment (IDE) known as jBACI. Its purpose was to simulate concurrency in computing in a manner that could teach students about this concept with ease. This is done through multiple displays that are shown while a program is running in real-time; in particular, concurrency is demonstrated through its processes window. jBACI is quite unique in this aspect, as it is a feature that is not found in other simulators to this day. However, being nearly two decades old, jBACI is largely out-of-date. Problems with it include limited language support (Pascal and C–, neither of which are used today), limited reliability outside of the Windows operating system, and a lack of core features and conventions that are common in modern day IDEs.

VizCon is a reimplimentation of jBACI. We have kept all of the features of jBACI that made it useful for educational purposes. However, it is built using modern software and techniques, and it uses an interface and feature set that modern-day programmers will be accustomed to. In addition, it supports macOS and Linux as well as Windows.

2 Installation and Execution

On Windows, download and run the .exe file. The program can be uninstalled normally via the Control Panel or Windows app.

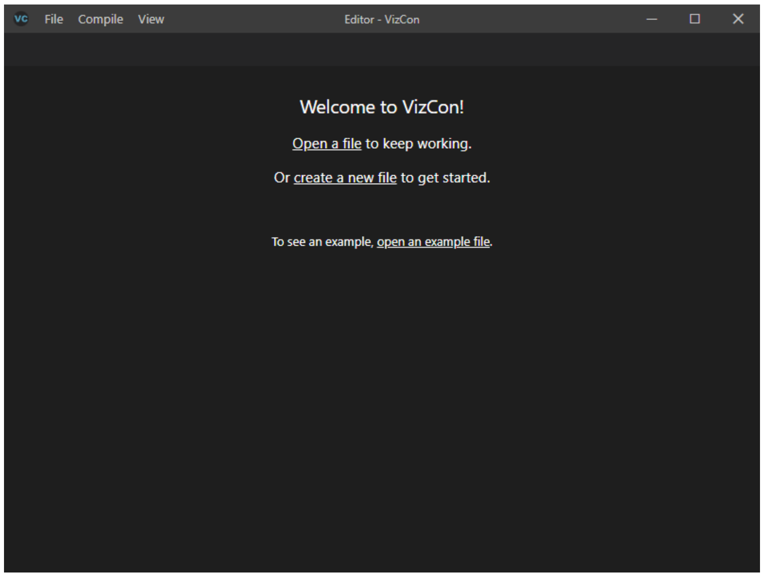
On macOS, download and run the .dmg file. On Debian-based Linux distros, download and run the .deb file. On Red Hat-based distros, download and run the .rpm file.

You must have the following dependencies installed:

$$-gcc >= 8$$

3 The Integrated Development Environment

Upon startup, the program will greet you with a Welcome screen shown in Figure 1.



Welcome to VizCon!

Open a file to keep working.

Or create a new file to get started.

To see an example, open an example file.

Figure 1: Welcome screen

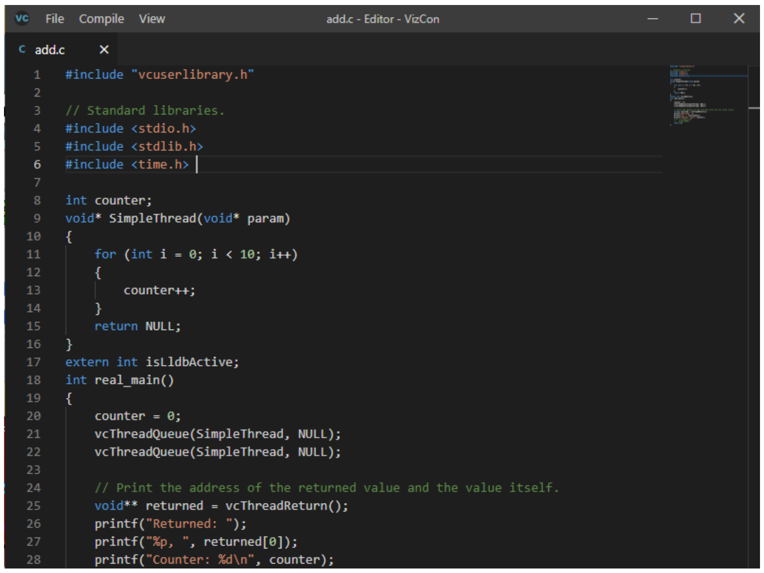
From the Welcome screen, you have three options: you can create a new file, open an existing file from your last saved opened directory, or open a file from the Examples directory. Every example given is stored in resources/examples.

The toolbar on the top contains the basic menu options expected in an IDE, such as the ability to save or open a file. The keyboard shortcuts for these menu items are listed to the right of the names. The user can also zoom the text in the program in or out. Along with this are additional options exclusive to this program. VizCon's IDE can be separated into two parts: the editor and the visualizer. These components can be toggled between in the View menu.

3.1 Editing and Compiling

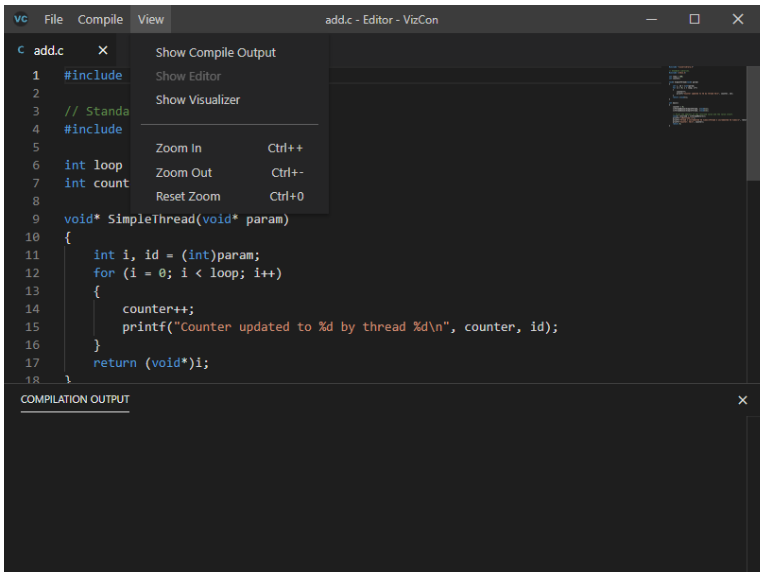
Figure 2 shows a brief overview of the editor.

Shown in the figure is the IDE running one of VizCon's example programs, add.c. The content of the files are shown in a text area, and can be edited as one would expect. Once a program is written, it can either be compiled and run all at once or simply compiled without running from the Compile menu. All source files must be written with the .c extension, as C is the only language that the compiler supports. Compilation output can be shown from the View menu as seen in Figure 3.



```c
#include "vcuserlibrary.h"

// Standard libraries.
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int counter;
void* SimpleThread(void* param)
{
    for (int i = 0; i < 10; i++)
    {
        counter++;
    }
    return NULL;
}
extern int isLldbActive;
int real_main()
{
    counter = 0;
    vcThreadQueue(SimpleThread, NULL);
    vcThreadQueue(SimpleThread, NULL);

    // Print the address of the returned value and the value itself.
    void** returned = vcThreadReturn();
    printf("Returned: ");
    printf("%p, ", returned[0]);
    printf("Counter: %d\n", counter);
```

Figure 2: VizCon's editor

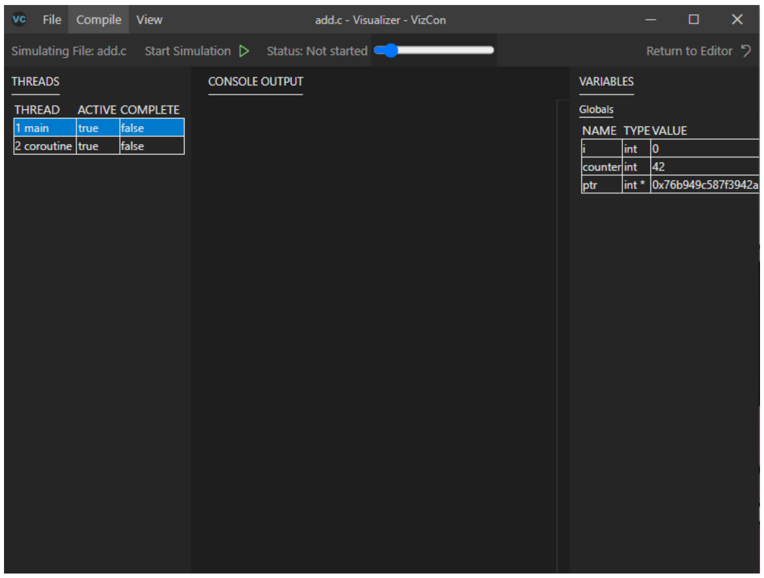


```c
#include

// Standa
#include

int loop
int count

void* SimpleThread(void* param)
{
    int i, id = (int)param;
    for (i = 0; i < loop; i++)
    {
        counter++;
        printf("Counter updated to %d by thread %d\n", counter, id);
    }
    return (void*)i;
}
```

Figure 3: Compilation window

3.2 Running the Program

When the Compile and Run option is selected, VizCon switches to its visualizer. Figure 4 shows a brief overview.



THREAD	ACTIVE	COMPLETE
1 main	true	false
2 coroutine	true	false

NAME	TYPE	VALUE
i	int	0
counter	int	42
ptr	int *	0x76b949c587f3942a

Figure 4: VizCon's visualizer

The threads in the program are shown in the left column, along with whether or not that thread is currently active and if it has completed. The compilation output is shown in the middle column. The right column shows all of the variable names in the program, as well as their respective values. In order to run the program, the Start Simulation button must be pressed. The user can return to the editor by pressing the Return to Editor button on the top right, or by showing it in the View menu.

4 Concurrency Constructs

4.1 Data Types

VizCon's user library (vcuserlibrary.h) provides a set of data types to aid with concurrent programming in a cross-platform manner.

4.1.1 threadFunc

A data type representing a function that gets taken in during thread creation.

4.1.2 VcSem (or VcSemaphore)

VizCon's built-in semaphore data type.

4.1.3 VcMutex

VizCon's built-in mutex data type.

4.2 vcThreadQueue

Prepares a thread instance with the function and arguments.

4.2.1 Syntax

```
void vcThreadQueue(threadFunc func, void *arg);
```

4.3 vcThreadStart

Starts all threads that were created in previous calls to vcThreadQueue.

4.3.1 Syntax

```
void vcThreadStart();
```

4.4 vcThreadReturn

Starts all threads that were created in previous calls to vcThreadQueue, and returns their values.

4.4.1 Syntax

```
void **vcThreadReturn();
```

4.5 vcMutexCreate

Creates a mutex.

4.5.1 Syntax

```
vcMutex vcMutexCreate();
```

4.6 vcMutexLock

Attempt to acquire a lock on the mutex, waiting if it is not yet available.

4.6.1 Syntax

```
void vcMutexLock(vcMutex mutex);
```

4.7 vcMutexTryLock

Attempt to acquire a lock on the mutex, returning immediately if it is not yet available.

4.7.1 Syntax

```
    int vcMutexTrylock(vcMutex mutex);
```

4.8 vcMutexUnlock

Release the lock on a mutex.

4.8.1 Syntax

```
    void vcMutexUnlock(vcMutex mutex);
```