Recursos Avançados de C++

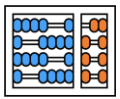
Módulo 3

Prof. Dr. Bruno B. P. Cafeo

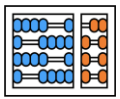
Instituto de Computação
Universidade Estadual de Campinas

Agenda

- Introdução
- The Windows Programming Model
- Hello World
- Controles e components
- Gerenciamento de eventos



Windows API



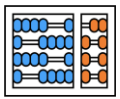
UNICAMP

Windows API

- A Windows API, informalmente conhecida como WinAPI, é o conjunto principal de interfaces de programação de aplicativos (APIs) da Microsoft disponível nos sistemas operacionais Windows.

- A Windows API se refere a várias implementações de plataforma que são frequentemente chamadas por seus próprios nomes, como o Win32 API.

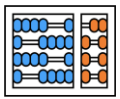
- A maioria dos programas do Windows interage com a Windows API.



Windows API

- A Windows API é focada principalmente na linguagem de programação C.

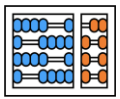
- No entanto, ela pode ser usada por qualquer compilador ou montador capaz de lidar com estruturas de dados de baixo nível e convenções de chamada prescritas.



Windows API

- A Windows API fornece funções para criar e gerenciar janelas e controles na interface do usuário, como botões e barras de rolagem.

- Ela lida com entrada de mouse e teclado, bem como outras funcionalidades relacionadas à GUI.

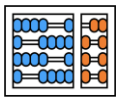


História da Windows API

- A Windows API evoluiu ao longo dos anos, expandindo suas funcionalidades e mantendo a compatibilidade com versões anteriores.

- A transição de Win16 para Win32 marcou uma grande mudança na API.

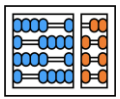
- A Windows API passou por várias versões, como Win16, Win32, Win32s e Win64.



Outras implementações

- Projetos como Wine, ReactOS e Odin fornecem implementações alternativas da Windows API para sistemas não-Windows.

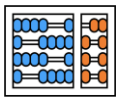
- DosWin32 e HX DOS Extender emulam a API para permitir a execução de programas Windows no ambiente DOS.



Wrappers Libraries

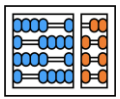
- A Microsoft desenvolveu wrappers, como a Microsoft Foundation Class Library (MFC), para permitir uma interação mais abstrata com a Windows API.

- Outras bibliotecas, como o Windows Template Library (WTL) e Active Template Library (ATL), também oferecem alternativas.

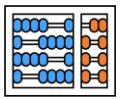


Principais Categorias de Funções

- Base Services
- Advanced Services
- Graphics Device Interface
- User Interface
- Common Dialog Box Library
- Common Control Library
- Windows Shell
- Network Services

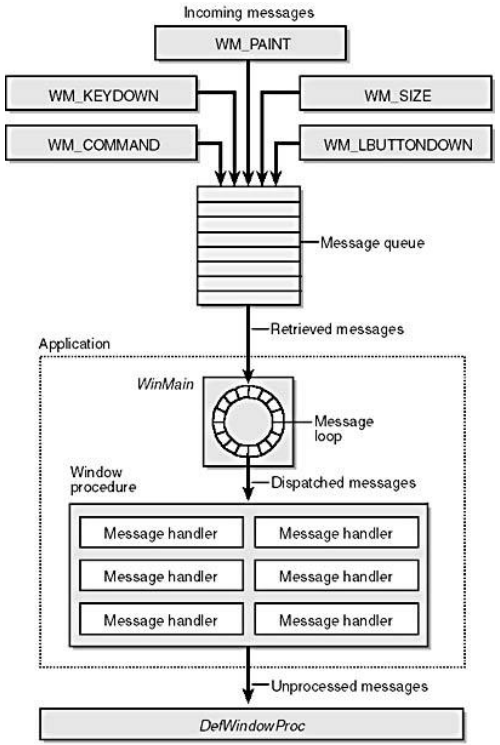


The Windows Programming Model

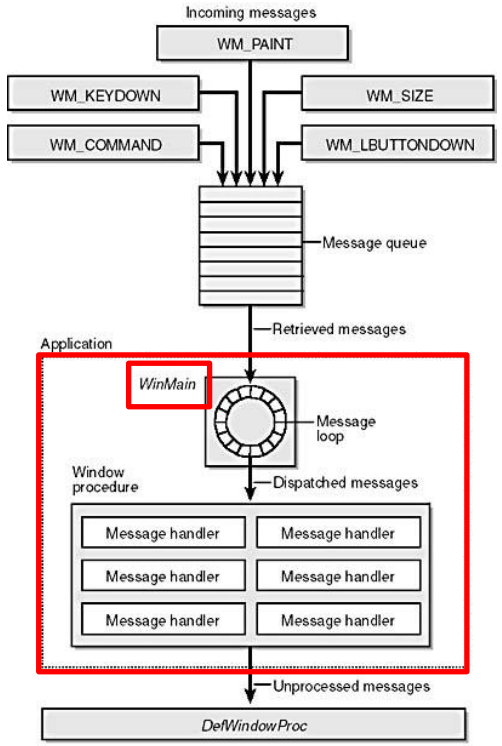


UNICAMP

The Windows Programming Model



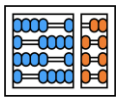
The Windows Programming Model



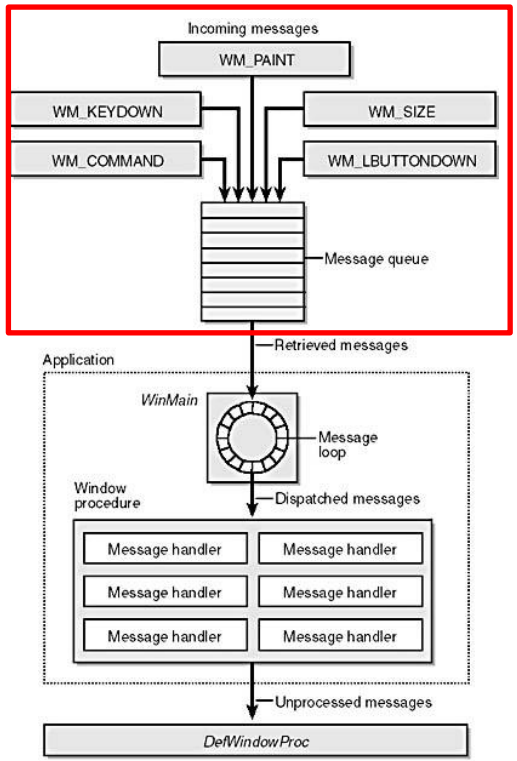
Janelas

- Janelas são elementos fundamentais das interfaces do Windows.

- Elas representam áreas retangulares na tela onde você pode exibir informações, interagir com o usuário e exibir controles como botões e caixas de texto.



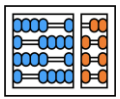
The Windows Programming Model



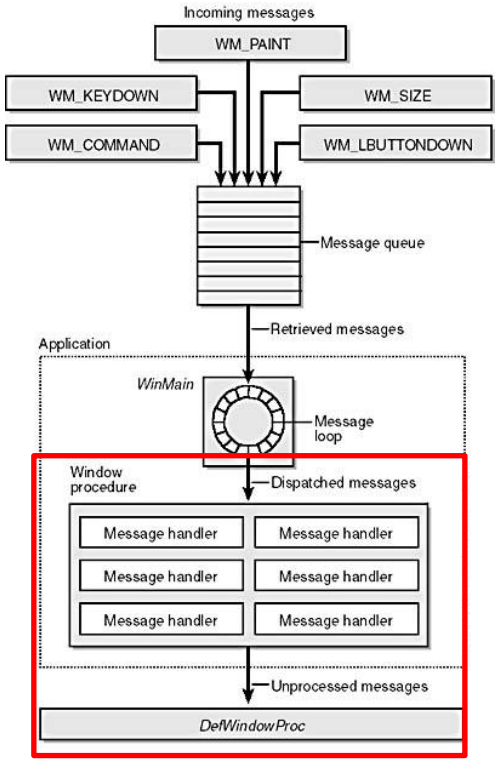
Mensagens

- As mensagens são o mecanismo de comunicação entre janelas no sistema Windows.

- Elas representam eventos ou ações que ocorrem, como cliques de mouse, pressionamentos de teclas ou atualizações de janelas.



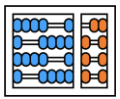
The Windows Programming Model



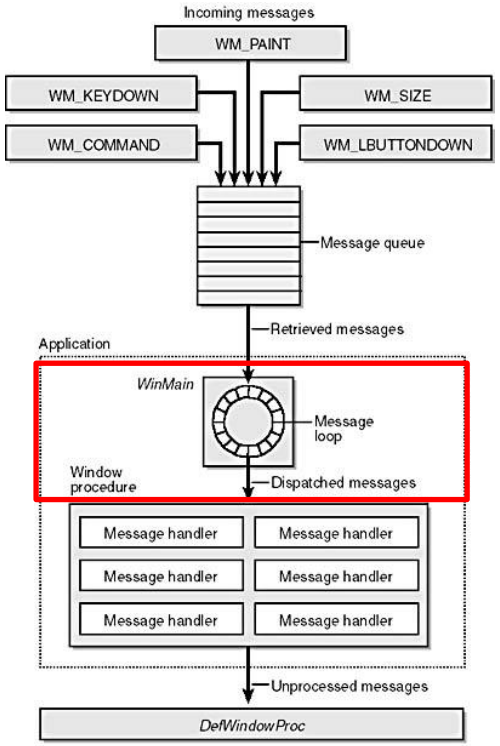
Procedimento de Janelas

- Procedimentos de Janela são funções que processam mensagens enviadas a uma janela específica.

- Cada janela tem seu próprio procedimento de janela, que define como a janela deve responder às mensagens recebidas.



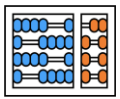
The Windows Programming Model



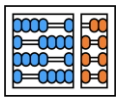
Ciclo de Mensagens

- O Ciclo de Mensagens é o processo pelo qual as mensagens são tratadas pelo Windows.

- O sistema Windows envia mensagens para as janelas, que são processadas pelos procedimentos de janela associados a elas.



Win32 API
(“Hello World!”)



Criação e gerenciamento de janelas

- Função `WinMain`: Ponto de entrada da aplicação Windows.

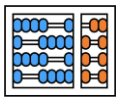
```
int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine,
                   int nCmdShow)
```

- Função `WndProc`: Procedimento de janela responsável por processar mensagens.

```
LRESULT CALLBACK WndProc(HWND hwnd, UINT msg, WPARAM wParam, LPARAM lParam)
```

Referência WinMain: https://learn.microsoft.com/en-us/windows/win32/api/winbase/nf-winbase-winmain

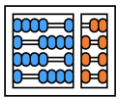
Referência WndProc: https://learn.microsoft.com/en-us/windows/win32/api/winuser/nc-winuser-wndproc



Criação e gerenciamento de janelas

- Mensagens e Loop de Mensagens: As mensagens são eventos enviados ao aplicativo, como cliques de mouse e pressionamentos de tecla. O loop de mensagens é o coração da aplicação, onde as mensagens são processadas.

```
1   MSG msg;
2   while (GetMessage(&msg, NULL, 0, 0)) {
3       TranslateMessage(&msg);
4       DispatchMessage(&msg);
5   }
6
```

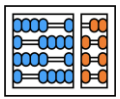


Window Class (WNDCLASS)

- Uma Window Class define um conjunto de comportamentos que várias janelas terão ao longo da execução.

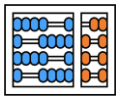
- Toda janela deve ser associada a uma window class.

- Apesar do nome, a window class não é uma classe (conceito de classes em C++). Na verdade, ela é uma estrutura de dados usada internamente pelo sistema operacional.



Window Class (WNDCLASS)

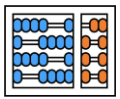
```
1  // Register the window class.
2  const wchar_t CLASS_NAME[]  = L"Registrando uma janela";
3
4  WNDCLASS wc = { };
5
6  wc.lpfnWndProc   = WindowProc;
7  wc.hInstance     = hInstance;
8  wc.lpszClassName = CLASS_NAME;
9
```



Window Class (WNDCLASS)

- Você deve definir os seguintes membros da estrutura:
  - `lpfnWndProc` é um ponteiro para uma função definida pela aplicação chamada procedimento da janela ou window proc. O procedimento da janela define a maior parte do comportamento da janela. Por enquanto, esse valor é uma declaração antecipada de uma função.
  - `hInstance` é o identificador da instância da aplicação. Obtenha esse valor do parâmetro hInstance de wWinMain.
  - `lpszClassName` é uma sequência de caracteres que identifica a classe da janela.

Referência WNDCLASS: https://learn.microsoft.com/en-us/windows/win32/api/winuser/ns-winuser-wndclassa



Criando uma janela

```
HWND CreateWindowEx(
  [in]           DWORD     dwExStyle,
  [in, optional] LPCSTR    lpClassName,
  [in, optional] LPCSTR    lpWindowName,
  [in]           DWORD     dwStyle,
  [in]           int       X,
  [in]           int       Y,
  [in]           int       nWidth,
  [in]           int       nHeight,
  [in, optional] HWND      hWndParent,
  [in, optional] HMENU     hMenu,
  [in, optional] HINSTANCE hInstance,
  [in, optional] LPVOID    lpParam
);
```

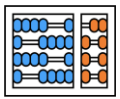
```
1  hwndMain = CreateWindowEx(
2      0,
3      CLASS_NAME,
4      L"Aprendendo Win32",
5      WS_OVERLAPPEDWINDOW,
6      CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT,
7      NULL,
8      NULL,
9      hInstance,
10     NULL
11 );
12
13 if (hwndMain == NULL) {
14     return 0;
15 }
16
```

Referência: https://learn.microsoft.com/en-us/windows/win32/api/winuser/nf-winuser-createwindowexw

Código básico

```
#include <windows.h>

LRESULT CALLBACK WindowProc(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam);

int WINAPI wWinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, PWSTR pCmdLine, int nCmdShow){
    // Register the window class.
    const wchar_t CLASS_NAME[]  = L"Sample Window Class";

    WNDCLASS wc = { };

    wc.lpfnWndProc   = WindowProc;
    wc.hInstance     = hInstance;
    wc.lpszClassName = CLASS_NAME;

    RegisterClass(&wc);

    // Create the window.

    HWND hwnd = CreateWindowEx(
        0,                              // Optional window styles.
        CLASS_NAME,                     // Window class
        L"Learn to Program Windows",    // Window text
        WS_OVERLAPPEDWINDOW,            // Window style

        // Size and position
        CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT,

        NULL,       // Parent window
        NULL,       // Menu
        hInstance,  // Instance handle
        NULL        // Additional application data
        );
```

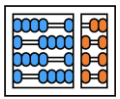
```
    if (hwnd == NULL)
    {
        return 0;
    }

    ShowWindow(hwnd, nCmdShow);

    // Run the message loop.

    MSG msg = { };
    while (GetMessage(&msg, NULL, 0, 0) > 0)
    {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }

    return 0;
}

LRESULT CALLBACK WindowProc(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam){
    switch (uMsg){
    case WM_DESTROY:
        PostQuitMessage(0);
        return 0;

    }
    return DefWindowProc(hwnd, uMsg, wParam, lParam);
}
```



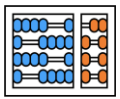
UNICAMP

Operações com janela

- Mostrar janela: `ShowWindow(hWnd, nCmdShow);`
- Atualizar janela: `UpdateWindow(hWnd);`
- Fechando janela: `DestroyWindow(hWnd);`
- Encerrando o programa: `PostQuitMessage(0);`



Win32 API
(Controles e componentes)

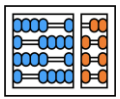


UNICAMP

Controles e componentes

- Botões, caixas de texto e rótulos são elementos de interface do usuário que os aplicativos podem criar e gerenciar.

- Para criar controles básicos, como botões, você usa a função `CreateWindow`.



Controles e componentes

- Criando um botão:

```
CreateWindow(L"BUTTON", L"Meu Botão", WS_CHILD | WS_VISIBLE, 10, 10, 100, 30, hWnd,
NULL, hInstance, NULL);
```

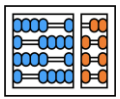
- Criando um listview:

```
CreateWindow(WC_LISTVIEW, L"Listview", WS_CHILD | WS_VISIBLE | WS_BORDER |
LVS_REPORT, 10, 10, 300, 200, hWnd, NULL, hInstance, NULL);
```

- Criando uma barra de status:

```
CreateWindow(STATUSCLASSNAME, NULL, WS_CHILD | WS_VISIBLE, 0, 0, 0, 0, hWnd, NULL,
hInstance, NULL);
```

Referência: https://learn.microsoft.com/en-us/windows/win32/controls/individual-control-info

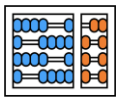


Controles e componentes

- MessageBox:

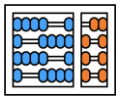
```
MessageBox(hWnd, L"Isso é uma caixa de diálogo!", L"Título", MB_OK | MB_ICONINFORMATION);
```

- Menu:
  - `CreateMenu();`
  - `CreatePopupMenu();`
  - `SetMenu(HWND hWnd, HMENU hMenu);`
  - `BOOL AppendMenuW(HMENU hMenu, UINT uFlags, UINT_PTR uIDNewItem, LPCWSTR lpNewItem);`



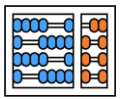
Controles e components (Menu)

```
1   hMenu = CreateMenu();
2
3   // Adiciona opções no menu principal
4   AppendMenu(hMenu, MF_STRING, ID_MENU_ITEM_1, L"Opção 1");
5
6   // Cria um submenu e adiciona duas opções a ele
7   HMENU hSubMenu = CreatePopupMenu();
8   AppendMenu(hSubMenu, MF_STRING, ID_SUBMENU_ITEM_1, L"Submenu 1");
9   AppendMenu(hSubMenu, MF_STRING, ID_SUBMENU_ITEM_2, L"Submenu 2");
10
11  // Adiciona a segunda opção com o submenu
12  AppendMenu(hMenu, MF_POPUP, (UINT)hSubMenu, L"Opção 2");
13
14  // Adiciona opções no menu principal
15  AppendMenu(hMenu, MF_STRING, ID_MENU_ITEM_3, L"Opção 3");
16
17  SetMenu(hwnd, hMenu);
18
```



UNICAMP

Win32 API
(Gerenciamento de eventos)

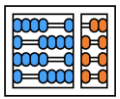


UNICAMP

Mensagens

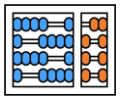
Evento	Descrição
WM_CREATE	Enviado quando uma janela está sendo criada.
WM_DESTROY	Enviado quando uma janela está sendo destruída.
WM_CLOSE	Enviado quando o usuário tenta fechar uma janela.
WM_SIZE	Enviado quando o tamanho de uma janela é alterado.
WM_MOVE	Enviado quando uma janela é movida para uma nova posição.
WM_MOUSEMOVE	Enviado quando o mouse é movido dentro da área de uma janela.
WM_LBUTTONDOWN	Enviado quando o botão esquerdo do mouse é pressionado.
WM_LBUTTONUP	Enviado quando o botão esquerdo do mouse é liberado.
WM_KEYDOWN	Enviado quando uma tecla do teclado é pressionada.
WM_COMMAND	Enviado quando um comando é emitido por um controle ou menu.

Referência: https://wiki.winehq.org/List_Of_Windows_Messages



Gerenciando eventos

```
LRESULT CALLBACK WndProc(HWND hwnd, UINT msg, WPARAM wParam, LPARAM lParam
) {
    switch (msg) {
        case WM_CREATE:
            // Tratar evento de criação da janela
            return 0;

        case WM_CLOSE:
            // Tratar evento de fechamento da janela
            if (MessageBox(hwnd, L"Deseja fechar a janela?", L"Fechar
            Janela", MB_ICONQUESTION | MB_YESNO) == IDYES) {
                DestroyWindow(hwnd);
            }
            return 0;

        case WM_PAINT:
            // Tratar evento de pintura da janela
            PAINTSTRUCT ps;
            HDC hdc = BeginPaint(hwnd, &ps);
            // Desenhar na janela usando o contexto de dispositivo (HDC)
            hdc
            EndPaint(hwnd, &ps);
            return 0;

        case WM_DESTROY:
            // Tratar evento de destruição da janela
            PostQuitMessage(0);
            return 0;

        default:
            return DefWindowProc(hwnd, msg, wParam, lParam);
    }
    return 0;
}
```



Mensagem personalizada

- As mensagens personalizadas são usadas para permitir a comunicação personalizada entre diferentes partes de um aplicativo Win32.

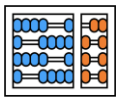
- Passos
  - Definir o código da mensagem personalizada:

```
#define WM_MINHA_MENSAGEM (WM_USER + 1)
```

  - Postar a mensagem personalizada

```
PostMessage(hWnd, WM_MINHA_MENSAGEM, wParam, lParam);
```

  - Receber e processar a mensagem personalizada
    - No procedimento de janela (`WndProc`), é preciso adicionar um caso para tratar a mensagem personalizada: `case WM_MINHA_MENSAGEM`



Mensagem personalizada

```c
#include <windows.h>

// Identificador da mensagem personalizada
#define WM_MINHA_MENSAGEM (WM_USER + 1)

// Procedimento de Janela (WndProc)
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam) {
    switch (message) {
        case WM_COMMAND:
            switch (LOWORD(wParam)) {
                case ID_MEUBOTAO:
                    // Quando o botão é clicado, envie a mensagem personalizada
                    PostMessage(hWnd, WM_MINHA_MENSAGEM, 0, 0);
                    break;
            }
            break;

        case WM_MINHA_MENSAGEM:
            // Quando a mensagem personalizada é recebida, exiba um diálogo de mensagem
            MessageBox(hWnd, L"Mensagem Personalizada Recebida!", L"Informação", MB_ICONINFORMATION);
            break;

        case WM_DESTROY:
            PostQuitMessage(0);
            break;

        default:
            return DefWindowProc(hWnd, message, wParam, lParam);
    }
    return 0;
}
```

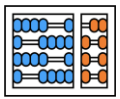
```c
int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nCmdShow) {
    // Registre a classe da janela
    WNDCLASS wc = {};
    wc.lpfnWndProc = WndProc;
    wc.hInstance = hInstance;
    wc.lpszClassName = L"MyWindowClass";
    RegisterClass(&wc);

    // Crie a janela principal
    HWND hWnd = CreateWindow(L"MyWindowClass", L"Exemplo de Mensagem Personalizada", WS_OVERLAPPEDWINDOW,
        CW_USEDEFAULT, CW_USEDEFAULT, 400, 200, NULL, NULL, hInstance, NULL);

    if (!hWnd) {
        return 1;
    }

    // Crie um botão na janela
    CreateWindow(L"BUTTON", L"Meu Botão", WS_CHILD | WS_VISIBLE, 10, 10, 100, 30, hWnd, (HMENU)ID_MEUBOTAO, hInstance, NULL);

    ShowWindow(hWnd, nCmdShow);
    UpdateWindow(hWnd);

    // Loop de Mensagens
    MSG msg;
    while (GetMessage(&msg, NULL, 0, 0)) {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }

    return msg.wParam;
}
```

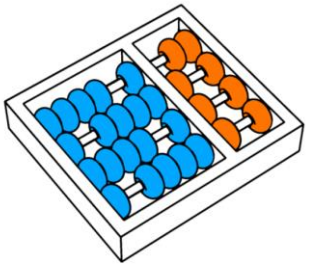
Estudar!

- Typedefs, Tipos de ponteiros, Precisão de tipos de ponteiros
- Notação húngara
- Funções UNICODE e ANSI
- TCHAR

https://learn.microsoft.com/en-us/windows/win32/learnwin32/windows-coding-conventions

https://learn.microsoft.com/en-us/windows/win32/learnwin32/working-with-strings





INSTITUTO DE COMPUTAÇÃO

UNICAMP

Prof. Dr. Bruno B. P. Cafeo

Sala 04
Instituto de Computação - Unicamp
Av. Albert Einstein, 1251
Cidade Universitária
Campinas – SP
13083-852

https://ic.unicamp.br/~cafeo/
cafeo@ic.unicamp.br