BIBLIOTECA ALLEGRO

Por Bárbara Castilho

O QUE É ALLEGRO?

Considerações iniciais.

MATERIAIS E MANUAL

- O manual oficial da biblioteca Allegro, com a descrição e modo de utilização de todas as suas funções encontra-se no site http://www.allegro.cc/manual/
- Material de apoio e exemplos se encontram na página pessoal do Prof. Dr. Simão: www.pessoal.utfpr.edu.br/jeansimao/Fundamentos1/Fundamentos1.htm

UM POUCO MAIS SOBRE ALLEGRO

- É uma API para programação multimídia. Contém rotinas para manipulação de imagens, sons e input (entrada de dados por teclado ou joystick). Uma biblioteca bem completa para aprender a programar jogos.
- Originalmente foi desenvolvida para ser empregada na plataforma Atari.
- É uma plataforma transversal e trabalha com muitos compiladores diferentes. Desenvolvido originalmente por Shawn Hargreaves, é agora um projeto com contribuições do mundo inteiro.
- Permite desde a criação de softwares mais complexos (editor de imagens) até funcionar como um "tocador" de MP3.

POR QUE UTILIZAR ALLEGRO?

- **Facilidade de utilização** - Vem com documentação e exemplos detalhados;
- **Simplicidade** - Fácil relação usuário/processo;
- **Livre** - não lhe custará nada e não há nenhuma limitação em seu uso;
- **Recursos Imediatos** - Disponibiliza uma série de recursos de um modo quase imediato.

CONSIDERAÇÕES

- Antes de ser feita uma análise completa de um programa, deve-se levar em conta que essa biblioteca, quando utilizada, possui uma programação em janelas.
- Por tal motivo, alguns detalhes e cuidados devem ser tomados para que não ocorram problemas durante a execução do programa.

CONSIDERAÇÕES

- É interessante que o programador tenha uma organização do projeto, pois isso facilita na detecção e correção de erros de programação (normalmente é o que acontece).
- Montar uma estrutura lógica (esboço) do que será feito ANTES de fazer, isso minimiza as chances de erros.
- Trabalho em equipe, se este funcionar bem, aumenta a produtividade. Comunicação e entendimentos de ambas as partes é essencial.
- Uma boa prática de programação (caso seja necessário utilizar um recurso, quando terminar de usar, libere ele, pois é outra fonte de erros).

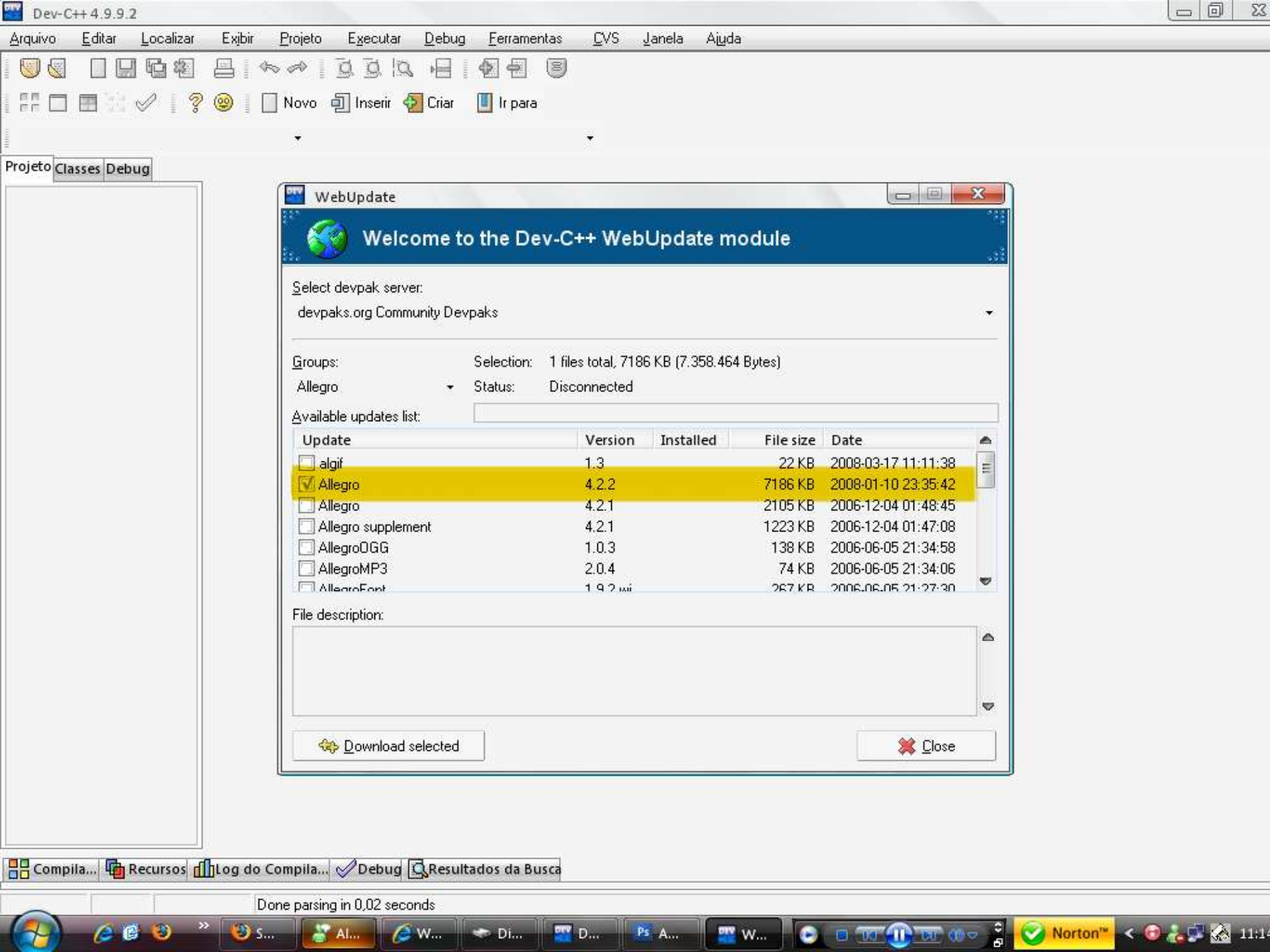
INSTALAÇÃO

Procedimento básico.

INSTALAÇÃO DO ALLEGRO

- O Allegro pode ser instalado em vários compiladores, mas, por ser o mais usado em aula, vamos instalá-lo no Dev C++.

- Inicie o Dev C++ e vá em Ferramentas > Atualizações.
- Em "Select devpack server", selecione "devpacks.org Community Devpacks"
- Clique em Check for updates e aguarde.


Dev-C++ 4.9.9.2
Arquivo Editar Localizar Exibir Projeto Executar Debug Ferramentas CVS Janela Ajuda
Novo Inserir Criar Ir para
Projeto Classes Debug
WebUpdate
Welcome to the Dev-C++ WebUpdate module
Select devpak server:
devpaks.org Community Devpaks
Groups:
Allegro
Selection: 1 files total, 7186 KB (7.358.464 Bytes)
Status: Disconnected
Available updates list:
Update Version Installed File size Date
algif 1.3 22 KB 2008-03-17 11:11:38
Allegro 4.2.2 7186 KB 2008-01-10 23:35:42
Allegro 4.2.1 2105 KB 2006-12-04 01:48:45
Allegro supplement 4.2.1 1223 KB 2006-12-04 01:47:08
AllegroOGG 1.0.3 138 KB 2006-06-05 21:34:58
AllegroMP3 2.0.4 74 KB 2006-06-05 21:34:06
File description:
Download selected
Close
Compila... Recursos Log do Compila... Debug Resultados da Busca
Done parsing in 0,02 seconds
Norton™

INSTALAÇÃO DO ALLEGRO

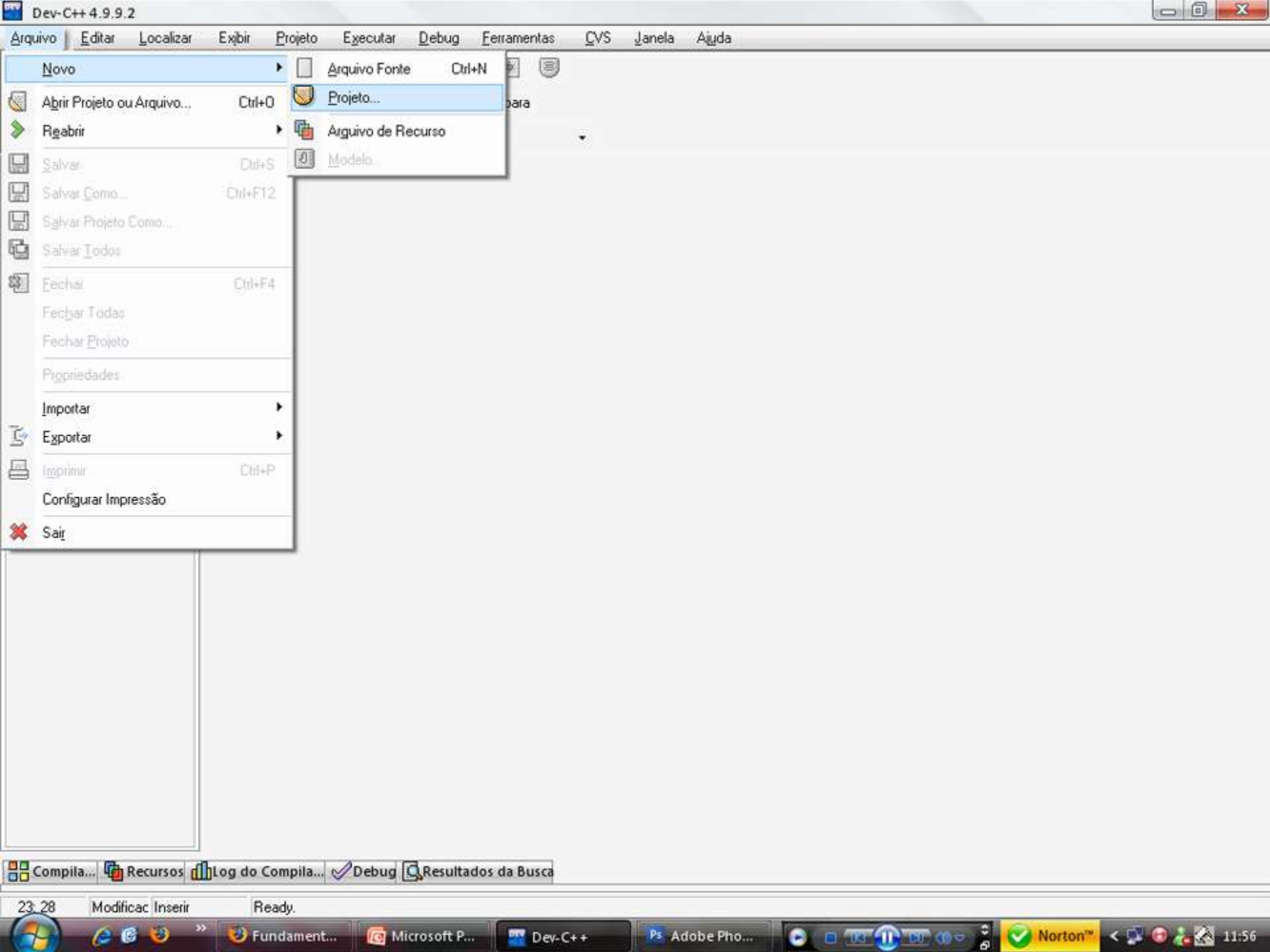
- Após essa janela, é só seguir os passos que serão mostrados. É bem simples e dedutivo (aliás, não passa de simplesmente clicar em avançar).

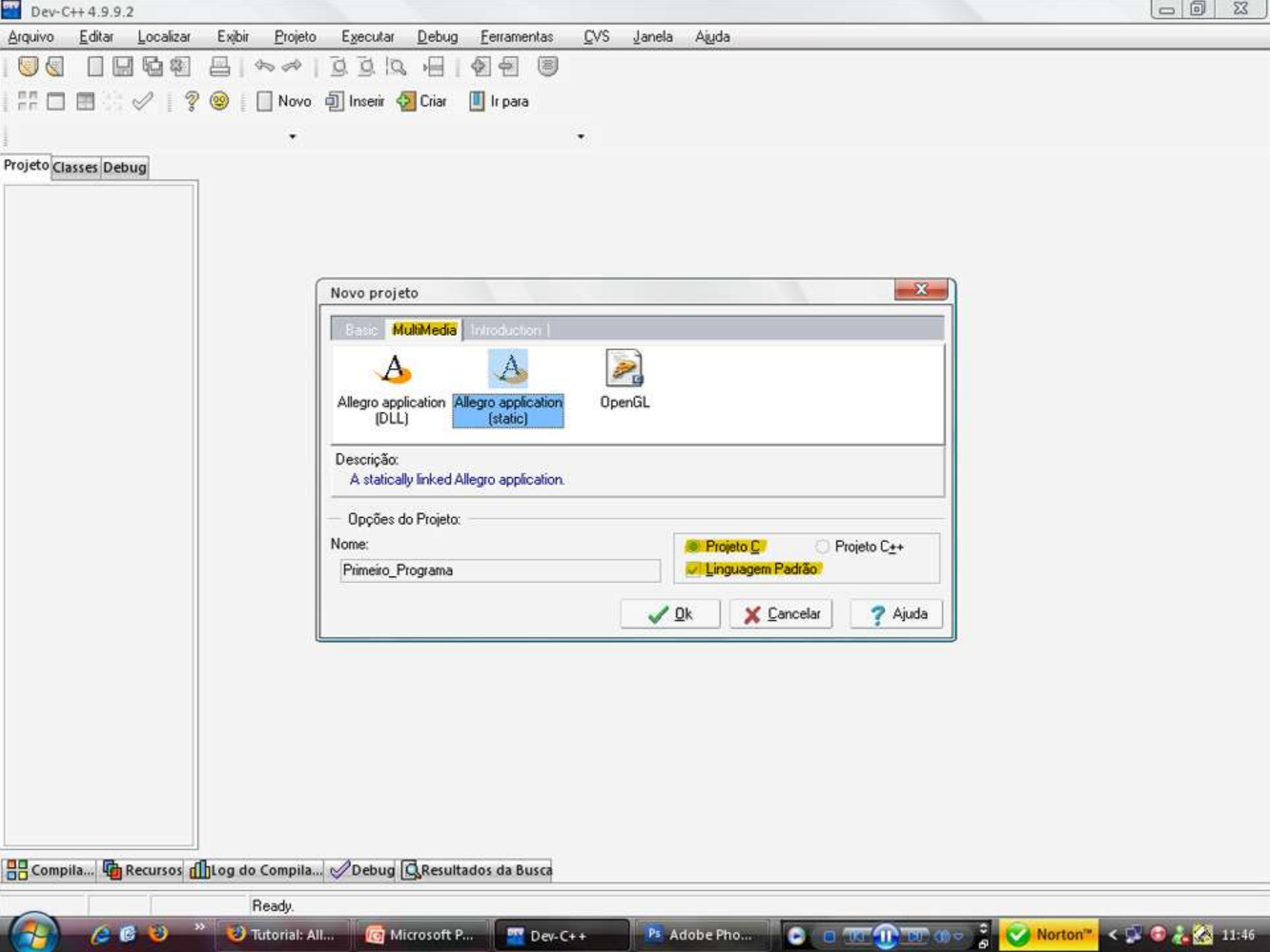
O PRIMEIRO PROGRAMA

Conhecendo os conceitos básicos.

O PRIMEIRO PROGRAMA COM O ALLEGRO

- Vá em Arquivo > Novo > Projeto;
- Clique em MultiMedia;
- Selecione Allegro application (static);
- No canto direito inferior, selecione Projeto C e Linguagem padrão;
- Dê um nome ao programa e salve na pasta que desejar.


Dev-C++ 4.9.9.2
Arquivo
Editar
Localizar
Exibir
Projeto
Executar
Debug
Ferramentas
CVS
Janela
Ajuda
Novo
Abrir Projeto ou Arquivo... Ctrl+O
Reabrir
Salvar Ctrl+S
Salvar Como... Ctrl+F12
Salvar Projeto Como...
Salvar Todos
Fechar Ctrl+F4
Fechar Todas
Fechar Projeto
Propriedades
Importar
Exportar
Imprimir Ctrl+P
Configurar Impressão
Sair
Arquivo Fonte Ctrl+N
Projeto...
Arquivo de Recurso
Modelo...
Compila...
Recursos
Log do Compila...
Debug
Resultados da Busca
23: 28
Modificac
Inserir
Ready.
Fundament...
Microsoft P...
Dev-C++
Adobe Pho...
Norton™
11:56


Dev-C++ 4.9.9.2
Arquivo Editar Localizar Exibir Projeto Executar Debug Ferramentas CVS Janela Ajuda
Novo Inserir Criar Ir para
Projeto Classes Debug
Novo projeto
Basic MultiMedia Introduction
Allegro application (DLL)
Allegro application (static)
OpenGL
Descrição:
A statically linked Allegro application.
Opções do Projeto:
Nome:
Primeiro_Programa
Projeto C
Projeto C++
Linguagem Padrão
Ok
Cancelar
Ajuda
Compila...
Recursos
Log do Compila...
Debug
Resultados da Busca
Ready.
Tutorial: All...
Microsoft P...
Dev-C++
Adobe Pho...
Norton™
11:46

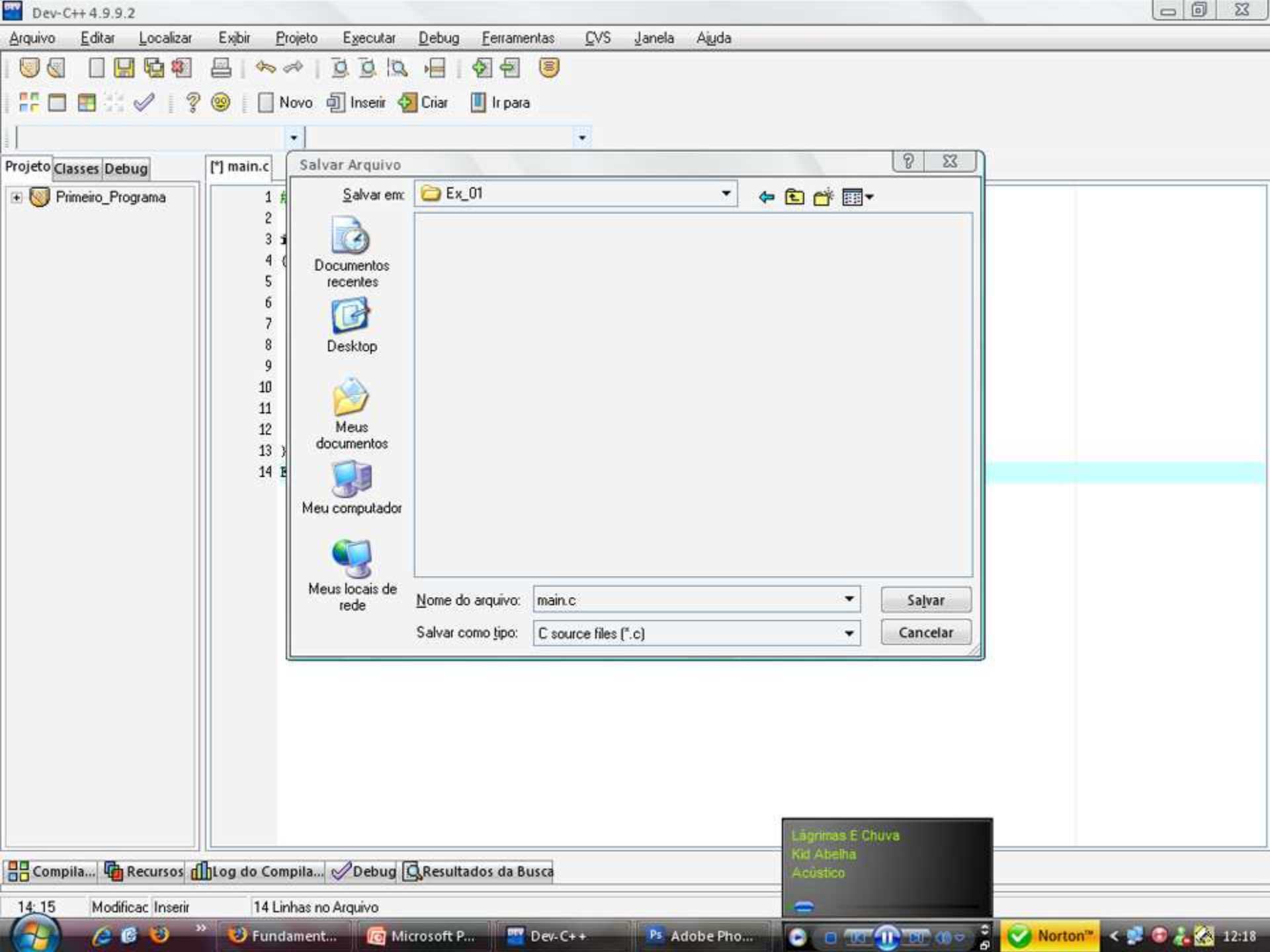
O PRIMEIRO PROGRAMA COM O ALLEGRO

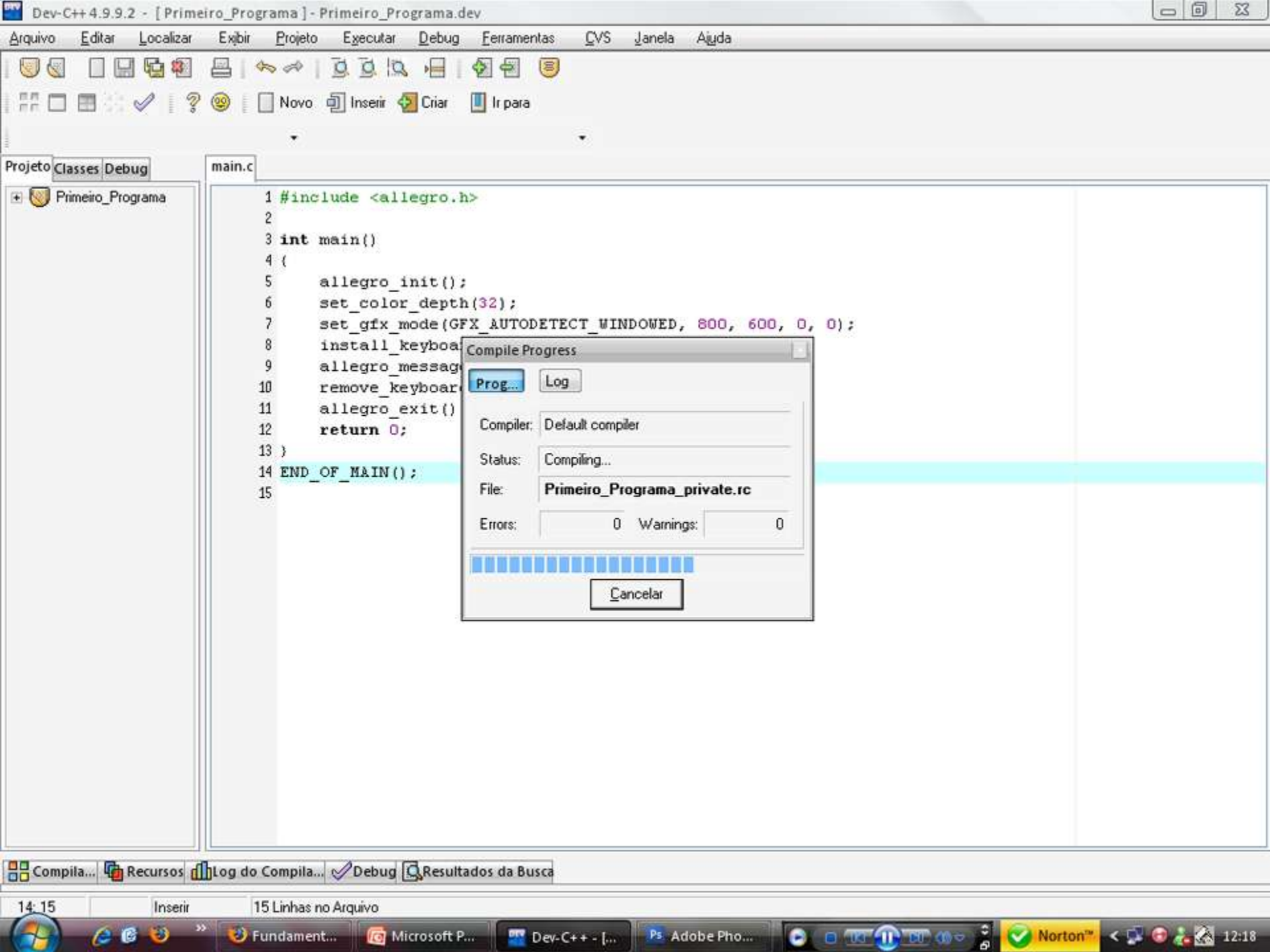
- Após salvar o projeto, você verá que abrirá uma janela com um código – que é padrão que aparece inicialmente nos projetos do Allegro. Para testar se a instalação está correta, copie e cole o código a seguir substituindo o anterior.

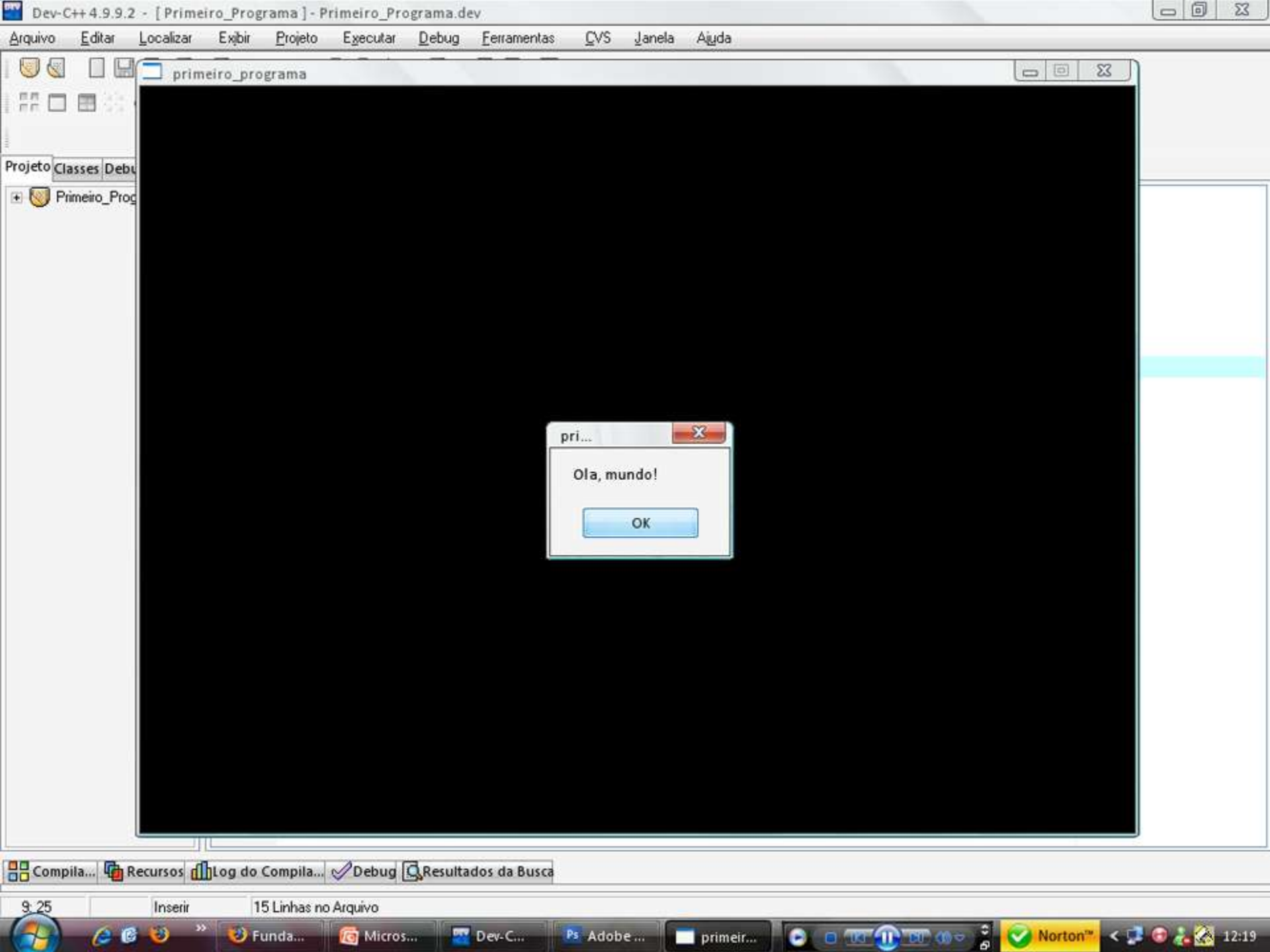
```
#include <allegro.h>

int main()
{
    allegro_init();
    set_color_depth(32);
    set_gfx_mode(GFX_AUTODETECT_WINDOWED, 800, 600, 0, 0);
    allegro_message("Olá, mundo!");
    allegro_exit();
    return 0;
}
END_OF_MAIN();
```

O PRIMEIRO PROGRAMA COM O ALLEGRO

- Agora, compile o código, salve o arquivo main.c no mesmo diretório que foi salvo o projeto. Se tudo estiver certo, você verá a seguinte janela:


Dev-C++ 4.9.9.2
Arquivo Editar Localizar Exibir Projeto Executar Debug Ferramentas CVS Janela Ajuda
Novo Inserir Criar Ir para
Projeto Classes Debug
Primeiro_Programa
[*] main.c
Salvar Arquivo
Salvar em: Ex_01
Documentos recentes
Desktop
Meus documentos
Meu computador
Meus locais de rede
Nome do arquivo: main.c
Salvar como tipo: C source files (*.c)
Salvar
Cancelar
Compila... Recursos Log do Compila... Debug Resultados da Busca
14: 15 Modificac Inserir 14 Linhas no Arquivo
Lágrimas E Chuva
Kid Abelha
Acústico
Fundament... Microsoft P... Dev-C++ Adobe Pho... Norton™ 12:18


Dev-C++ 4.9.9.2 - [Primeiro_Programa] - Primeiro_Programa.dev
Arquivo Editar Localizar Exibir Projeto Executar Debug Ferramentas CVS Janela Ajuda
Novo Inserir Criar Ir para
Projeto Classes Debug
Primeiro_Programa
main.c
#include <allegro.h>
int main()
{
allegro_init();
set_color_depth(32);
set_gfx_mode(GFX_AUTODETECT_WINDOWED, 800, 600, 0, 0);
install_keyboa
allegro_messag
remove_keyboar
allegro_exit()
return 0;
}
END_OF_MAIN();
Compile Progress
Prog...
Log
Compiler: Default compiler
Status: Compiling...
File: Primeiro_Programa_private.rc
Errors: 0 Warnings: 0
Cancelar
Compila...
Recursos
Log do Compila...
Debug
Resultados da Busca
14: 15
Inserir
15 Linhas no Arquivo
Fundament...
Microsoft P...
Dev-C++ - [...
Adobe Pho...
Norton™
12:18


Dev-C++ 4.9.9.2 - [Primeiro_Programa] - Primeiro_Programa.dev
Arquivo Editar Localizar Exibir Projeto Executar Debug Ferramentas CVS Janela Ajuda
primeiro_programa
Projeto Classes Debu
Primeiro_Prog
pri...
Ola, mundo!
OK
Compila... Recursos Log do Compila... Debug Resultados da Busca
9: 25 Inserir 15 Linhas no Arquivo
Funda... Micros... Dev-C... Adobe ... primeir... Norton™ 12:19

O PRIMEIRO PROGRAMA COM O ALLEGRO

- Se você conseguiu fazer os passos e estiver vendo a tela acima PARABÉNS, a instalação foi efetuada com sucesso.
- Caso não apareça essa janela, reinstale a biblioteca (caso essa seja a primeira tentativa de criar um programa com o Allegro), o compilador (caso reinstalar a biblioteca não funcione) ou reveja o código.

O PRIMEIRO PROGRAMA COM O ALLEGRO – ANÁLISE DO CÓDIGO

- 1 - #include <allegro.h>

  Assim como se faz com qualquer outra biblioteca, é necessário incluir o arquivo de cabeçalho do Allegro. Para tal, **APÓS** todas as outras diretivas #include das bibliotecas padrão (ex: stdio.h, stdlib.h), inclua o

  #include <allegro.h>

O PRIMEIRO PROGRAMA COM O ALLEGRO – ANÁLISE DO CÓDIGO

- 2 – int allegro_init();

  Função responsável por iniciar a biblioteca Allegro no programa. Sem esse comando não conseguimos utilizar nenhuma função do Allegro. Por isso, antes de chamar qualquer outro comando, chame a função

  allegro_init();

O PRIMEIRO PROGRAMA COM O ALLEGRO – ANÁLISE DO CÓDIGO

- 3 – set_color_depth(32); - Essa função define a resolução das cores de todas as imagens que você vai carregar durante o projeto. Pode-se usar 8, 15, 16, 24 e 32 bits. O melhor é deixar em 32 bits.
- 4 - set_gfx_mode(GFX_AUTODETECT_WINDOWED, 800, 600, 0, 0); - Esse comando é responsável por detectar a placa de vídeo, determinar o tamanho da tela em pixel e o posicionamento inicial x,y. A recomendação é só mexer nos valores 800 e 600 (no nosso exemplo), para alterar a resolução da tela de jogo – nesse caso, inicia-se uma janela 800*600 pixels. Para que o jogo seja em tela inteira, troque GFX_AUTODETECT_WINDOWED por GFX_AUTODETECT_FULLSCREEN.
- 5 - allegro_message("Ola, mundo!"); - Caixa de diálogo. Pode ser invocada a qualquer instante posterior a devida inicialização. Uma possível utilização é como um auxílio ao debug.
- 6 - allegro_exit(); - Ao final do programa, utilizamos essa função para retirar os domínios do Allegro sobre o computador. Lembrando que esse recurso não desaloca memória alocada dinamicamente, coisa que deve ser feita usando o método apropriado para cada um.
- 7 - END_OF_MAIN(); - Método que deve estar sempre presente no final da int main(), basicamente informa a biblioteca que terminou o seu fluxo de execução e que deve terminar o processo.

APRIMORANDO O CÓDIGO

Conhecendo as funções mais utilizadas para a elaboração do jogo.

APRIMORANDO O CÓDIGO INICIAL – INSTALANDO TECLADO E MOUSE

- install_keyboard(); - Inicializa o teclado para ser usado pelo Allegro. Deve-se chamar esta função antes de qualquer outra função de teclado. Após a chamada desta função, não se pode mais utilizar as funções padrões de acesso ao teclado (scanf, getchar, etc.)

- install_mouse(); - Inicializa o mouse para ser usado pelo Allegro. É necessário chamar esta função antes de qualquer outra função de mouse. Retorna -1 se ocorrer algum erro; caso contrário, retorna o número de botões do mouse.

EXTRA 1 – FUNÇÕES MAIS UTILIZADAS DO TECLADO

- extern volatile char key[KEY_MAX]; - Caso alguma tecla específica tenha sido apertada, é atribuído *true* ao evento, do contrário, é *false*. Serve para recuperar informações do usuário ou utilizar o teclado para movimentar um personagem dentro de um jogo, por exemplo.

EXEMPLO DE USO:

```
if(key[KEY_M])
{
    executar algum comando
}
```

- int keypressed(); - Segue o mesmo princípio do key[KEY_MAX];, contudo o valor retornado é o seu código scancode (usado pelo Allegro), podendo ser utilizado também para ser um limitador de operações (executa uma parte do código se e somente se uma tecla qualquer for pressionada).

- void clear_keybuf(); - Limpa o buffer de memória relacionado ao teclado (é útil quando se trabalha com loops recuperando informação).

EXTRA 2 – FUNÇÕES MAIS UTILIZADAS DO MOUSE

- void show_mouse(BITMAP *bmp); - Função utilizada para exibir o mouse em um bitmap. O padrão é utilizar show_mouse(screen).
- extern volatile int mouse_x; extern volatile int mouse_y; extern volatile int mouse_b; - Respectivamente, guardam o valor do mouse no eixo x, no eixo y e o valor do botão pressionado.

  Bit 1 = Botão esquerdo do mouse
  Bit 2 = Botão direito do mouse
  Bit 4 = Rodinha do mouse (se houver).

- void scare_mouse(); - "Esconde" o cursor.
- extern volatile int freeze_mouse_flag; - Evita que o cursor seja atualizado com frequência (evita que fique "piscando").

APRIMORANDO O CÓDIGO INICIAL – FUNÇÕES DE TEXTO

- Existem diversas funções com formas de alinhamento de texto no Allegro. A mais comum é a **textout_ex**, texto escrito da esquerda para a direita.

void textprintf_ex(BITMAP *bmp (1), const FONT *f (2), int x (3), int y (4), int color (5), int bg(6), const char *fmt, ... (7));

1. Bitmap onde será impresso o texto;
2. A fonte usada para escrever o texto na tela. A fonte padrão é font;
3. Coordenada X inicial do texto;
4. Coordenada Y inicial do texto;
5. Cor do texto. Para usar as cores, usamos a função makecol(R, G, B);. Essa função transforma o valor **RGB** em um valor inteiro responsável por identificar a cor que queremos utilizar. As cores **RGB** são formadas por uma combinação de 3 valores;
6. Cor do background do texto;
7. O texto em si.

```
textprintf_ex(screen,font,25,33,makecol(255,0,0),-1, "Digite o texto aqui!");
```

APRIMORANDO O CÓDIGO INICIAL – O LAÇO DE REPETIÇÃO PRINCIPAL

- Esse laço de repetição principal servirá para que a janela do programa não se feche até que o usuário queira que isso aconteça. É simples, basta adicionar o seguinte código no programa:

```
while (!key[KEY_ESC])
{
        comandos a serem executados
}
```

```
#include <allegro.h>

int main()
{
    allegro_init();
    set_color_depth(32);
    set_gfx_mode(GFX_AUTODETECT_WINDOWED, 800, 600, 0, 0);
    install_mouse();
    install_keyboard();

    while(!key[KEY_ESC])
    {
        show_mouse(screen);
        freeze_mouse_flag;
        textprintf_ex(screen,font,25,33,makecol(255,0,0),-1, "Exemplo!");

    }
    allegro_exit();
    return 0;
}
END_OF_MAIN();
```

APRIMORANDO O CÓDIGO INICIAL – COLOCANDO SOM

- Existe apenas uma função de configuração do som no Allegro, que inicializa tanto os dispositivos digitais quanto os dispositivos MIDI. Esta função, install_sound, possui três parâmetros: o primeiro indica o controlador de som digital a ser usado pelo Allegro; o segundo, o controlador de som MIDI; e o terceiro existe apenas por motivos de compatibilidade com versões antigas do Allegro, e deve ser ignorado passando-se o valor **NULL**.
- A melhor alternativa para evitar erros é deixar que o próprio Allegro ache automaticamente qual a melhor configuração ...

```
install_sound (DIGI_AUTODETECT, MIDI_AUTODETECT, NULL);
```

EXTRA 3 – FUNÇÕES DE SOM NO ALLEGRO

- O Allegro disponibiliza algumas variedades de formatos de sons, cobrindo o necessário para o desenvolvimento de qualquer jogo. Nesse exemplo, serão usados os formatos MIDI e SAMPLE (WAV). Lembrando que há até o suporte para arquivos *.mp3. A configuração para esse formato é algo mais complicado e não é algo essencial para se fazer um bom jogo. Logo, para quem se interessar, pode pesquisar sobre Fmod ou AllegroMP3.

- *Para arquivos *.mid (as "músicas" do jogo):*

  - MIDI *load_midi(const char *filename); - Alocação de memória para um arquivo *.mid. Seria como se estivesse "carregando" o arquivo no jogo.

```
MIDI * som = NULL;
*som = load_midi("nomedoarquivo.mid");
```

  - int play_midi(MIDI *midi, int loop); - Toca a MIDI especificada por **midi**, parando de tocar qualquer música que estivesse sendo tocada anteriormente. Se a flag **loop** estiver setada, a música será tocada até que a função seja novamente chamada para tocar outra música, ou a função stop_midi seja chamada. Caso a flag **loop** não esteja setada, a música irá parar de tocar ao alcançar o final do MIDI. Retorna um valor diferente de zero se um erro ocorrer.

```
play_midi(som, FALSE); -> toca somente uma vez
play_midi(som, 100);     -> toca até que se chame a função stop_midi();
```

  - void stop_midi(); - Encerra o fluxo de execução de um determinado midi.

```
stop_midi (som);
```

EXTRA 3 – FUNÇÕES DE SOM NO ALLEGRO

- *Para arquivos *.wav (os efeitos sonoros do jogo):*
  - SAMPLE *load_sample(const char *filename); - Alocação de memória para um arquivo *.mid. Seria como se estivesse "carregando" o arquivo no jogo.

```
SAMPLE* efeito = NULL;
*efeito = load_sample("nomedoarquivo.wav");
```

  - int play_sample(const SAMPLE *spl, int vol, int pan, int freq, int loop); - "Toca" de fato o SAMPLE para o usuário ouvir em um determinado instante. Alguns parâmetros são necessários: SAMPLE *spl = SAMPLE a ser "tocado"; int vol = intensidade inicial (0 a 255); int pan = distribuição do som nas caixas (0 a 255, onde 0 existe apenas na caixa esquerda e 255 na caixa direita); int freq = indica a velocidade como será "tocada" (1000 para a mesma velocidade, 2000 para o dobro e 500 para a metade da velocidade); int loop = quantidade de vezes que será "tocada".

```
play_sample(efeito, 255,127, 1000,0);
```

  - void stop_sample(const SAMPLE *spl); - Encerra o fluxo de execução de um determinado SAMPLE.

```
stop_sample (efeito);
```

TIMER NO ALLEGRO

Como colocar um cronômetro no jogo.

TIMER NO ALLEGRO

- Timer é de suma importância num jogo. Com ele você pode fazer missões com tempo, pode pegar o tempo que o jogador demorou para salvar o seu jogo, pode fazer contagens regressivas e diversas outras coisas.
- E também é muito simples de ser criado. Precisaremos de:
  - Uma variável global;
  - Uma função void que incrementará essa variável global;
  - Alguns códigos próprios da biblioteca Allegro;

EXTRA 4 – FUNÇÕES DE TIMER NO ALLEGRO

- int install_timer(); - Função análoga as funções install_keyboard(); e install_mouse(); mas usada para controlar as funções de tempo do jogo;
- int install_int(void (*proc)(), int speed); - Inicializa um temporizador para que, a cada intervalo de **speed** milisegundos, a função **proc** seja chamada;
- LOCK_VARIABLE(variavel); LOCK_FUNCTION(Funcao); e END_OF_FUNCTION(*Funcao*) - As macros **LOCK_VARIABLE** e **LOCK_FUNCTION** bloqueiam a área de memória ultilizada pela variável ou função, enquanto a macro **END_OF_FUNCTION** determina o final de uma função dentro do programa;
- void rest(unsigned int time); - Tempo de espera dado em milissegundos (fazer o programa "dormir"). Pode ser empregado para aumentar o tempo de execução de um laço de repetição (controlador de tempo). Utilização: rest(1000); (tempo de espera de um segundo);
- void rest_callback(long time, void (*callback)()) - Esta função faz com que o programa espere pelo tempo especificado no parametro, durante a espera, ela executa a função enviada no segundo parametro.

```
#include <allegro.h>

#define MAX_X 200
#define MAX_Y 30

void Setup();
void Load();
void Tempo(void);
void Finalizar();

int TEMPO = 0;
int Pause = 0;

int main() {
        Setup();
        Load();

   BITMAP* Buffer = create_bitmap(MAX_X, MAX_Y);

        while (!key[KEY_ESC])
   {
       if(key[KEY_P])
       {
          if (Pause == 0)
          {
            remove_int(Tempo);
            Pause = 1;
          }

          else
          {
             install_int(Tempo, 1000);
             Pause = 0;
          }

       }

       if (Pause == 0)
       {
          if(TEMPO % 60 < 10)
            textprintf_ex(Buffer, font, 50, 5, makecol(255,255,255), -
 1, "Tempo: %2d:0%d" ,TEMPO / 60, TEMPO % 60);
          else
             textprintf_ex(Buffer, font, 50, 5, makecol(255,255,255), -
 1, "Tempo: %2d:%2d" ,TEMPO / 60, TEMPO % 60);

             textprintf_ex(Buffer, font, 40, 18, makecol(255,255,255),
 -1, "[P] para pausar");
       }

       if (Pause == 1)
       {
          if(TEMPO % 60 < 10)
            textprintf_ex(Buffer, font, 50, 5, makecol(255,0,0), -1,
 "Tempo: %2d:0%d" ,TEMPO / 60, TEMPO % 60);
          else
             textprintf_ex(Buffer, font, 50, 5, makecol(255,0,0), -1,
 "Tempo: %2d:%2d" ,TEMPO / 60, TEMPO % 60);

             textprintf_ex(Buffer, font, 30, 18, makecol(255,0,0), -1,
 "[P] para continuar");

       }
       blit(Buffer, screen, 0,0,0,0, MAX_X, MAX_Y);
       clear(Buffer);
        }

        Finalizar();
        return 0;
}
END_OF_MAIN()
```

```
void Setup()
{
   set_uformat( U_ASCII );
   allegro_init();
   set_color_depth(32);

        set_gfx_mode(GFX_AUTODETECT_WINDOWED,MAX_X,MA
        X_Y,0,0);


        install_timer();
        install_keyboard();
        install_mouse();
   install_sound (DIGI_AUTODETECT, MIDI_AUTODETECT, NULL);

     if
        (set_gfx_mode(GFX_AUTODETECT_WINDOWED,MAX_X,MA
        X_Y,0,0) < 0)
        {

            char *erro_allegro = "Erro ao tentar iniciar gráficos ...";
            textout_ex(screen, font, erro_allegro ,(MAX_X/2)-
        (text_length(font , erro_allegro)/2),(MAX_Y/2)-
        (text_height(font)), makecol(255,0,0), 0);
            readkey();
            allegro_exit();
            exit(1);
       }

      if (install_sound(DIGI_AUTODETECT, MIDI_AUTODETECT,
        NULL) < 0)
       {
            char *erro_allegro2 = "Erro ao tentar iniciar som ...";
            textout_ex(screen, font, erro_allegro2,(MAX_X/2)-
        (text_length(font, erro_allegro2)/2),(MAX_Y/2)-
        (text_height(font)), makecol(255,0,0), 0);
            readkey();
            allegro_exit();
            exit(1);
       }
      set_window_title("TIMER");
}

void Load()
{
  LOCK_VARIABLE(TEMPO);
  LOCK_FUNCTION(Tempo);
  install_int(Tempo, 1000);
}

void Tempo (void)
{
        TEMPO++;
}
END_OF_FUNCTION(decrementa_tempo);

void Finalizar()
{
          clear(screen);
          char *fim = "FIM";
          textout_ex(screen, font, fim,(MAX_X/2)-(text_length(font,
        fim)/2),(MAX_Y/2)-(text_height(font)), makecol(255,0,0),
        0);
          clear_keybuf();
          readkey();
          allegro_exit();
}
```

DESENHANDO NA TELA DO JOGO

Conceito básico para a criação de jogos.

DESENHOS NO ALLEGRO

- Tendo em vista que a principal característica do Allegro é o uso da parte gráfica, existem diversas funções de desenho. Porém, antes de explicá-las, é necessário entender o modo como o Allegro trata a parte gráfica.

- Primeiramente, é preciso ressaltar que a tela é tratada, no caso dessa biblioteca, como um bitmap também (chamado screen).

- Há duas formas de se apresentar uma imagem com o Allegro: desenhando na tela através de funções específicas (algumas das quais serão apresentadas em breve nesse material) ou carregando desenhos criados por um outro programa (photoshop, paint ou obtidas na internet). Inicialmente, vamos aprender a desenhar diretamente através das funções do Allegro.

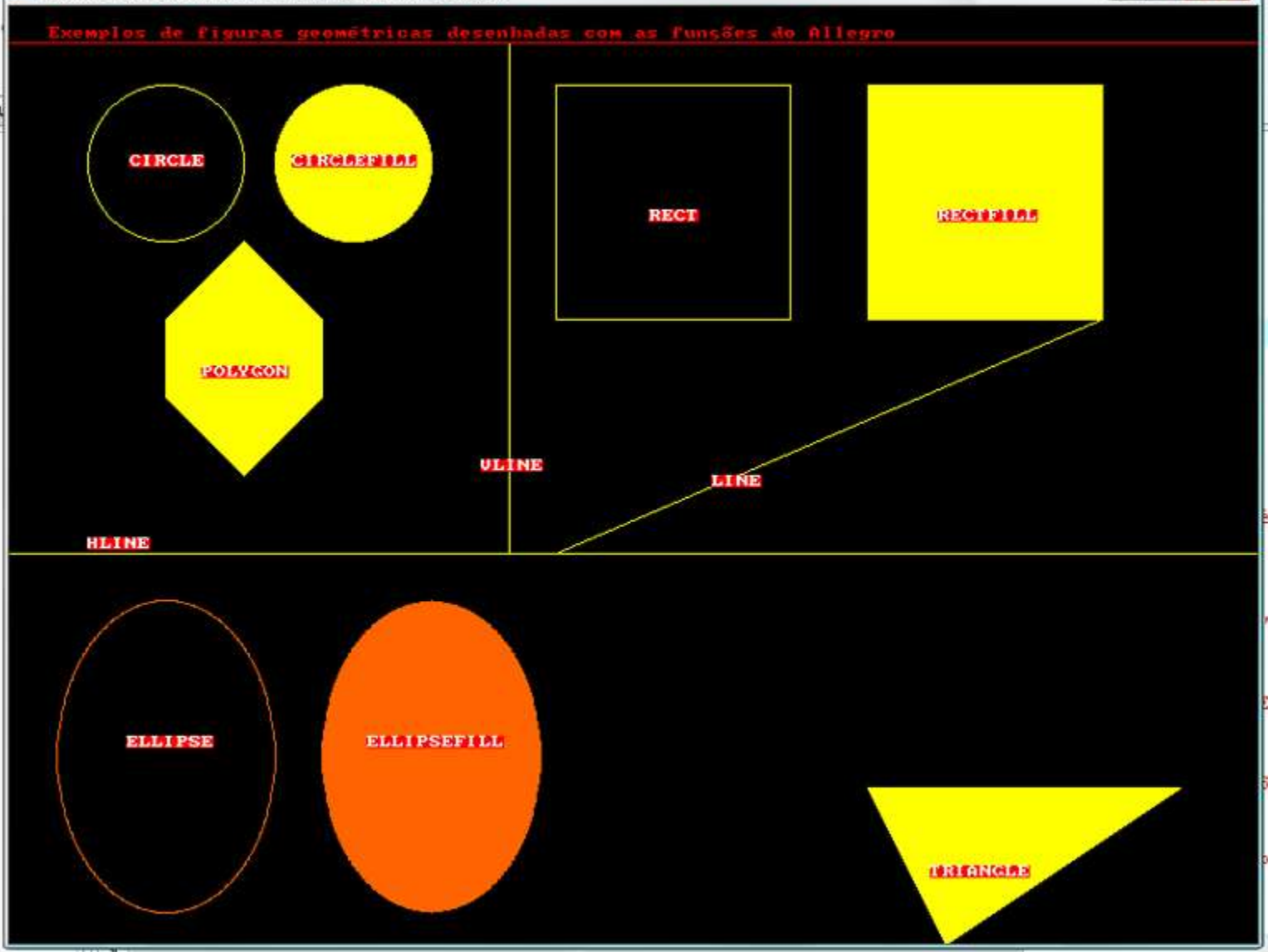
DESENHOS NO ALLEGRO – FUNÇÕES DE DESENHOS DA BIBLIOTECA

- void line(BITMAP *bmp, int x1, int y1, int x2, int y2, int color); - Desenha uma linha, no bitmap apontado por **bmp**, da coordenada (**x1**, **y1**) até a coordenada (**x2**, **y2**), utilizando a cor especificada por **color.**
- void hline(BITMAP *bmp, int x1, int y, int x2, int color); - Desenha uma linha horizontal, no bitmap apontado por **bmp**, da coordenada (**x1**, **y**) até a coordenada (**x2**, **y**), utilizando a cor especificada por **color**.
- void vline(BITMAP *bmp, int x, int y1, int y2, int color); - Desenha uma linha vertical, no bitmap apontado por **bmp**, da coordenada (**x**, **y1**) até a coordenada (**x**, **y2**), utilizando a cor especificada por **color**.
- void rect(BITMAP *bmp, int x1, int y1, int x2, int y2, int color); - Desenha a borda de um retângulo, no bitmap apontado por **bmp**, da coordenada (**x1**, **y1**) até a coordenada (**x2**, **y2**), utilizando a cor especificada por **color**.
- void rectfill(BITMAP *bmp, int x1, int y1, int x2, int y2, int color); - Desenha um retângulo, no bitmap apontado por **bmp**, da coordenada (**x1**, **y1**) até a coordenada (**x2**, **y2**), utilizando a cor especificada por **color**.
- void triangle(BITMAP *bmp, int x1, int y1, int x2, int y2, int x3, int y3, int color); - Desenha um triângulo, no bitmap apontado por **bmp**, com vértices (**x1**, **y1**), (**x2**, **y2**) e (**x3**, **y3**).
- void polygon(BITMAP *bmp, int vertices, int *points, int color); Desenha um polígono, no bitmap apontado por **bmp**, com **vertices** vértices especificados pelo vetor **points** de pares de coordenadas (x, y), com cor **color**.
- void circle(BITMAP *bmp, int x, int y, int radius, int color); - Desenha uma circunferência, no bitmap apontado por **bmp**, com centro (**x**, **y**) e raio **radius**, utilizando a cor especificada por **color**.
- void circlefill(BITMAP *bmp, int x, int y, int radius, int color); - Desenha um círculo, no bitmap apontado por **bmp**, com centro (**x**, **y**) e raio **radius**, utilizando a cor especificada por **color**.

Dev-C++ 4.9.9.2 - [Projeto1] - Projeto1.dev

Arquivo Editar Localizar Exibir Projeto Executar Debug Ferramentas CVS Janela Ajuda

FIGURAS GEOMÉTRICAS DESENHADAS COM O ALLEGRO



Projeto Classes Debu

Projeto1

main.c

Compila... Recursos Log do Compila... Debug Resultados da Busca

8: 42 Modificac Inserir 56 Linhas no Arquivo

Norton™ 13:12

DESENHANDO NO ALLEGRO – CARREGANDO BITMAPS

- O Allegro suporta alguns formatos de imagens. Sempre usaremos imagens no formato *.bmp (imagens bitmap de 24 bits), por ser um formato fácil de ser criado (o Paint é capaz de criar um arquivo com essas especificações) e ser um formato suportado. Imagens do tipo *.jpg, *.gif e *.png não são suportadas a princípio, porém, assim como no caso do som, há a possibilidade de colocar esses formatos, porém, isso novamente não é algo essencial para a criação desse jogo.

- extern BITMAP *screen; - Para facilitar o uso de suas funções de manipulação de bitmaps, o Allegro trata a tela também como um bitmap, que é definido no arquivo allegro.h como screen. Assim, nas funções de manipulação de bitmaps, sempre que desejarmos apresentar diretamente na tela algum gráfico, passaremos como argumento da função a variável screen.

- BITMAP *create_bitmap(int width, int height); - Um objeto da classe BITMAP. Utilizando-se desse método, é possível alocar dinamicamente memória para desenhar, escrever ou colocar imagens dentro das dimensões limitadas por *width* e *height*.

```
BITMAP *Imagem = create_bitmap(800, 600);
```

DESENHANDO NO ALLEGRO – CARREGANDO BITMAPS

- BITMAP *load_bitmap(const char *filename, RGB *pal); - Com essa função, é possivel carregar uma imagem no formato bmp. O parâmetro RGB *pal é uma palheta de cores, podendo ser omitido. Contudo se o programador deseja utilizar criando um sistema de cores), deve criar um objeto do tipo PALETTE e executar ações referentes a isso. Inicialmente é recomendado deixar esse parâmetro como NULL.

- extern PALETTE desktop_palette; - Esta paleta era utilizada pelo Atari ST. É utilizada pelos programas de teste e exemplo do Allegro e é a paleta padrão utilizada pelo mesmo, caso nenhuma outra paleta seja setada.

```
BITMAP *desenho = NULL;
desenho = load_bitmap("endereçodaimagem.bmp", desktop_palette);
```

DESENHANDO NO ALLEGRO - OUTRAS FUNÇÕES DE BITMAPS

- void clear(BITMAP *bitmap); - Limpa o bitmap em questão para a cor 0.
- void clear_to_color(BITMAP *bitmap, int color); - Limpa o bitmap apontado por **bitmap** para a cor especificada por **color**.
- void putpixel(BITMAP *bmp, int x, int y, int color); - Desenha um ponto, no bitmap apontado por **bmp**, na coordenada (x, y), utilizando a cor especificada por **color**.
- int getpixel(BITMAP *bmp, int x, int y); - Retorna o código de cor da coordenada (x, y) no bitmap apontado por **bmp**; retorna -1 caso o ponto esteja fora do bitmap.

DESENHANDO NO ALLEGRO – EXIBINDO OS BITMAPS

- As funções de desenho do próprio Allegro não necessitam de mais uma função específica para desenhá-los na tela. Basta colocar como bitmap de destino screen. Porém, quando carregamos um bitmap externo é preciso que usemos uma função para imprimir na tela. Aliás, o único bitmap que é automaticamente desenhado, por motivos óbvios, e exibido, é o screen. Todos os outros necessitam de uma função das que serão descritas aqui para exibir os bitmaps.

- void blit(BITMAP *source, BITMAP *dest, int source_x, int source_y, int dest_x, int dest_y, int width, int height); - Copia uma área retangular, de largura **width** e altura **height**, da coordenada (**source_x**, **source_y**) do bitmap apontado por **source** para a coordenada (**dest_x**, **dest_y**) do bitmap apontado por **dest**.

- void draw_sprite(BITMAP *bmp, BITMAP *sprite, int x, int y); - Copia inteiramente o bitmap apontado por **sprite** na coordenada (**x**, **y**) do bitmap apontado por **bmp**. Equivalente a blit(**sprite**, **bmp**, 0, 0, x, y, **sprite->w**, **sprite->h**).

  A grande diferença entre as funções blit e draw_sprite é que a primeira desenha o bitmap como ele é, enquanto a segunda trata como transparente as cores 0 (normalmente preto, no modo 8 bits) e rosa claro (nos outros modos).

- void rotate_sprite(BITMAP *bmp, BITMAP *sprite, int x, int y, fixed angle); - Rotaciona um BITMAP, desenhando em x e y, com a rotação determinada por angle em relação ao centro da imagem.

- void pivot_sprite(BITMAP *bmp, BITMAP *sprite, int x, int y, int cx, int cy, fixed angle); - Similar ao rotate_sprite(...), diferencia pois o programador define qual é o ponto de rotação (cx, cy).

```
#include <allegro.h>

int main()
{
    allegro_init();
    set_color_depth(32);
    set_gfx_mode(GFX_AUTODETECT_WINDOWED, 800, 600, 0, 0);
    install_mouse();
    install_keyboard();

    BITMAP* fundo   = load_bitmap("images//fundo.bmp", desktop_pallete);
    BITMAP* desenho = load_bitmap("images//boneco.bmp", desktop_pallete);

    while(!key[KEY_ESC])
    {
        clear_to_color(screen,makecol(255, 255, 255));
        draw_sprite(screen, desenho, 100, 100);
        blit(desenho, screen, 0,0,250,100, 129, 119);
    }
    allegro_exit();
    return 0;
}
END_OF_MAIN();
```

DESENHANDO NO ALLEGRO – ARRAY DE IMAGENS

- Assim como existe a necessidade de utilizar imagens para ilustração durante um desenvolvimento de jogo também existe a necessidade de usar arrays de imagens.

```
BITMAP *imagens[4];
imagens[0] = load_bitmap(“img1.bmp", NULL);
imagens[1] = load_bitmap("img2.bmp", NULL);
imagens[2] = load_bitmap("img3.bmp", NULL);
imagens[3] = load_bitmap("img4.bmp", NULL);
```

- O comando para desenhar a logo na tela ficaria da seguinte forma:

```
draw_sprite(screen, imagens[0], 0, 0);
```

EXTRA 5 - DATAFILES

- DataFile é um tipo especial de arquivo, com extensão .dat, e que pode armazenar diversos tipos de arquivos, como BitMaps, MIDIs, fontes, paletas, que podem ser carregados e utilizados durante a execução do programa.
- Para criar um arquivo DataFile existe um utilitário chamado *Grabber*, que se encontra no diretório **tools**, dentro do diretório principal do Allegro. O programa possui uma interface que facilita a confecção do arquivo DataFile.
- Para acessar um arquivo DataFile num programa, utiliza-se o tipo **DATAFILE**, da seguinte maneira:

```
DATAFILE *dat;
```

- Depois de determinar um ponteiro para o arquivo DataFile, pode-se abri-lo utilizando a função load_datafile, descrita abaixo:

```
DATAFILE *load_datafile(const char *filename);
```

Abre o arquivo de nome **filename** como um arquivo datafile, preparando-o para a leitura de seus dados. Retorna **NULL** caso ocorra algum erro.

EXTRA 6 – OUTRAS FUNÇÕES INTERESSANTES

- set_uformat(U_ASCII); - Coloque esse código ANTES do allegro_init();. Assim, você poderá usar acentuação nos textos de seus jogos;

- _set_window_title("Título da janela"); - Coloca o título que você quiser na janela do seu programa;

IMPORTANTÍSSIMO – AS DESTRUTORAS

- Sempre que você for usar imagens, fontes, midis, samples ou datafiles na memória do PC lembre-se de colocar o comando para destruí-las no final do código. Mesmo sendo apenas imagens para teste.
- Dessa forma seu computador não vai ficar lento, não vai travar e o compilador não vai sofrer tanto a cada execução.
  - void destroy_bitmap(BITMAP *bitmap); - destrutora de bitmaps;
  - void destroy_midi(MIDI *midi); - destrutora de midis;
  - void destroy_sample(SAMPLE *spl); - destrutora de samples;
  - void unload_datafile(DATAFILE *dat); - destrutora de datafiles;
- Existem outras destrutoras, como a remove_keyboard();, mas estas não são necessárias pois já estão incluídas em allegro_exit();.

CARREGANDO ARQUIVOS

- Um cuidado que se deve ter é que caso o caminho fornecido para os arquivos não for válido (não existir a imagem nesse local) e não foi tomado cuidado para analisar antes de usar caso a imagem tenha sido propriamente carregada, ao se tentar usar pela primeira vez a imagem, acontecerá um *crash no programa. Caso um local exato seja informado, deve seguir* esse padrão.

Exemplo: o programador quer acessar uma imagem que se encontra na pasta C:\Imagens\ cujo nome é Imagem.bmp. Para carregar esse bitmap, deve proceder da seguinte maneira: BITMAP *Imagem = load_bitmap("c://imagens//imagem.bmp", NULL);. Note as barras, caso não seja feito desta maneira, também acontecerá um erro. Agora caso queira carregar uma imagem que esteja na mesma pasta do programa, o modo alternativo para se carregar uma imagem é o seguinte: BITMAP *Imagem = load_bitmap("imagem.bmp", NULL);.

VALIDANDO AS VARIÁVEIS

- Assim como nos programas sem utilizar a biblioteca Allegro, é necessário que as variáveis sejam validadas (verificar se não há algo resultando em algum erro, como problemas nas placas de som ou vídeo).
- Com isso, o código começa a ficar grande, então, o uso das funções aqui é adequado.

TRANSPARÊNCIA DE BITMAPS

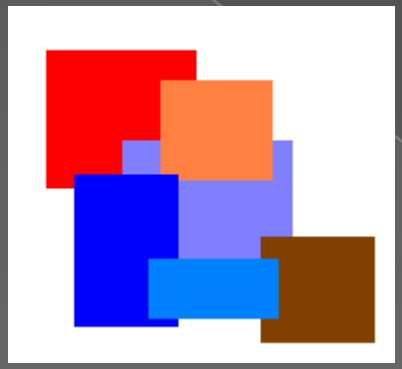
Desenhando apenas as partes desejadas de uma imagem.

TRANSPARÊNCIA DE BITMAPS

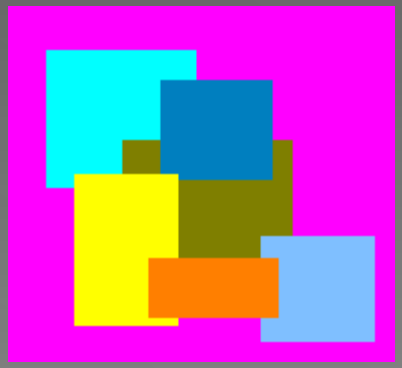
- A transparência em um bitmap serve para isolar a imagem desejada a ser exibida de seu fundo. Um especial cuidado deve ser tomado na hora de realizar tal procedimento, pois qualquer variação pode não gerar o efeito visado (a não correta isolação da imagem).

- Para ser feito isso, o fundo da imagem deve ter a cor (em RGB [255, 0, 255]) magenta. Isso não se aplica só ao fundo (seria uma aplicação), pode ser usado em qualquer lugar do bitmap onde se deseja uma transparência.

- Para que a transparência funcione, é necessário o uso da função *draw_sprite*.



Peça sem transparência no fundo



Peça com transparência no fundo

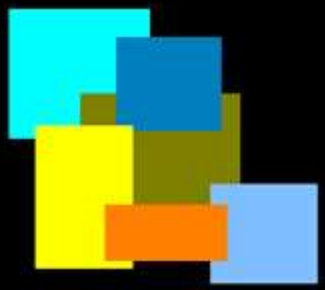
Dev-C++ 4.9.9.2 - [Projeto1] - Projeto1.dev

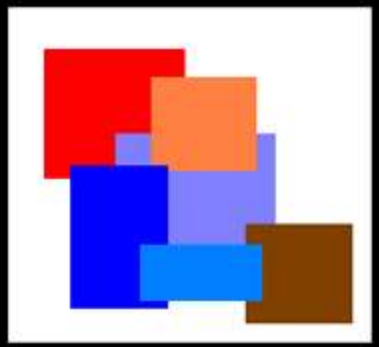
Arquivo Editar Localizar Exibir Projeto Executar Debug Ferramentas CVS Janela Ajuda

TESTE DE TRANSPARÊNCIA

COM TRANSPARENCIA

SEM TRANSPARENCIA





Projeto Classes Debu

Projeto1

Compila... Recursos Log do Compila... Debug Resultados da Busca

27: 14 Inserir 75 Linhas no Arquivo

Micros... (2) Alex ... Xemele ... 02.bmp... Adobe ... Dev-C+... TESTE D... Norton™ 01:00

DOUBLE BUFFERING

Resolvendo o problema de tela "piscando".

DOUBLE BUFFERING

- Quando começamos a desenhar vários objetos primitivos na tela, e movemos ele através do teclado (na execução do jogo) ou mesmo animamos figuras *.bmp, muitas vezes percebemos que a imagem do jogo fica piscando. O nome deste efeito é flicker, onde o monitor acaba imprimindo uma tela em preto antes da nova imagem do jogo ser desenhada na variável screen, formando essa impressão de que a tela está piscando.
- Para contornar este tipo de problema, existem várias técnicas de animação. No exemplo, será usada a mais popular delas, o double buffering.
- Desenhamos, neste *buffer*, os objetos que devem ser apresentados na tela. Após isso, desenhamos o conteúdo do *buffer* na tela, fazendo com que os objetos apareçam. Limpamos, então, o *buffer*, desenhamos os objetos novamente em suas novas posições, passamos o conteúdo do *buffer* para a tela, e assim por diante.

```
#include <allegro.h>

#define MAX_X 800
#define MAX_Y 600

void Setup();
void Finalizar();


int main() {
            Setup();

            BITMAP *img1    = load_bitmap("01.bmp", NULL);
            BITMAP *fundo   = load_bitmap("fundo.bmp", NULL);
            BITMAP * buffer = create_bitmap(MAX_X, MAX_Y);


            while (!key[KEY_ESC])
   {

         blit(fundo, buffer, 0,0,0,0,MAX_X, MAX_Y);
         draw_sprite(buffer, img1, 100,200);

         blit(buffer, screen, 0,0,0,0, MAX_X, MAX_Y);

            }


   destroy_bitmap(fundo);
   destroy_bitmap(img1);
   destroy_bitmap(buffer);

            Finalizar();
            return 0;
}
END_OF_MAIN()

void Setup()
{
   set_uformat( U_ASCII );
   allegro_init();
   set_color_depth(32);
   set_gfx_mode(GFX_AUTODETECT_WINDOWED,MAX_X,MAX_Y,0,0);


            install_timer();
            install_keyboard();
            install_mouse();
   install_sound (DIGI_AUTODETECT, MIDI_AUTODETECT, NULL);

      if (set_gfx_mode(GFX_AUTODETECT_WINDOWED,MAX_X,MAX_Y,0,0) < 0)
         {

            char *erro_allegro = "Erro ao tentar iniciar gráficos ...";
            textout_ex(screen, font, erro_allegro ,(MAX_X/2)-(text_length(font ,
            erro_allegro)/2),(MAX_Y/2)-(text_height(font)), makecol(255,0,0), 0);
              readkey();
              allegro_exit();
              exit(1);
         }

         if (install_sound(DIGI_AUTODETECT, MIDI_AUTODETECT, NULL) < 0)
          {
              char *erro_allegro2 = "Erro ao tentar iniciar som ...";
              textout_ex(screen, font, erro_allegro2,(MAX_X/2)-(text_length(font,
            erro_allegro2)/2),(MAX_Y/2)-(text_height(font)), makecol(255,0,0), 0);
              readkey();
              allegro_exit();
              exit(1);
          }

      set_window_title("COM DOUBLE BUFFERING");
}


void Finalizar()
{

            clear(screen);
            char *fim = "FIM - PRESSIONE QUALQUER TECLA PARA SAIR";
            textout_ex(screen, font, fim,(MAX_X/2)-(text_length(font,
            fim)/2),(MAX_Y/2)-(text_height(font)), makecol(255,0,0), 0);
            rest(1000);
            readkey();
            clear_keybuf();
            allegro_exit();
}
```

MOVIMENTAÇÃO DE IMAGENS

Uma das partes primordiais de um jogo.

MOVIMENTAÇÃO DE IMAGENS

- Nos jogos, a movimentação de personagens é muito importante para o desenrolar do jogo.
- Para que possamos programar essa movimentação, devemos atribuir coordenadas X e Y para o personagem que será movimentado. Um jeito rápido é fácil de se criar "estruturas padrão de personagens" é criando uma struct. Para ilustrarmos esse tópico, será usada uma estrutura para um quadrado.

```
struct Objeto
{
	int x, y;
	BITMAP* img;

}Quadrado;
```

MOVIMENTAÇÃO DE IMAGENS

- Após as devidas inicializações, temos uma estrutura Quadrado, que possui coordenadas X e Y na tela. Na hora de desenhar com o auxílio da função draw_sprite, ao invés de colocar um valor fixo, como estávamos fazendo anteriormente, vamos colocar os valores X e Y atribuídos a essa estrutura quadrado.

OU SEJA ...

```
draw_sprite(Buffer, Quadrado.img, Quadrado.x, Quadrado.y);
```

MOVIMENTAÇÃO DE IMAGENS

- Agora, vamos tratar da movimentação em si.
- Uma vez que possuímos variáveis e que essas variáveis são os valores enviados à função que está imprimindo as imagens na tela, para que exista o efeito de movimento, basta que façamos algo para que esses valores X e Y variáveis mudem. Isso pode ser facilmente feito utilizando Ifs cuja condição para que seja executado o comando de permitir a movimentação seja pressionar a tecla designada para a movimentação do personagem.

EXEMPLIFICANDO ...

```
if(key[KEY_LEFT])
    {
        Quadrado.x = Quadrado.x - 2;
    }


    if(key[KEY_RIGHT])
    {
        Quadrado.x = Quadrado.x + 2;
    }

    if(key[KEY_UP])
    {
        Quadrado.y = Quadrado.y - 2;
    }

    if(key[KEY_DOWN])
    {
        Quadrado.y = Quadrado.y + 2;
    }
```

MAS HÁ UM PROBLEMA COM ESSE CÓDIGO....

MOVIMENTAÇÃO DE IMAGENS

- O problema existente com esse código é que se for pressionado por um certo tempo uma mesma tecla, o quadradinho some da tela!

- Para solucionar esse problema, vamos iniciar o conceito de colisão.

COLISÃO

Parte I – Colisão com o terreno.

CONCEITO DE COLISÃO EM JOGOS

- Um dos pontos mais relevantes e mais importantes dentro de um jogo pode ser considerado a correta formulação de um sistema de colisão. Pois dependendo do jogo, tal evento é crucial para sua jogabilidade, ou é o foco principal do jogo (exemplo: sinuca).
- O grau de complexidade desse sistema depende e muito de quão perfeito que o programador quer que este seja ou então da geometria em questão. Hipóteses simplificadoras podem ser muito úteis na hora de sua implementação.

COLISÃO COM O TERRENO NESSE EXEMPLO

- Nesse caso, para que o quadrado não saia dos limites da tela, o valor de X e Y não podem ser menor que zero e a soma da largura (no caso de X) com a posição X e a soma da altura(no caso de Y) com Y não podem ser maiores do que o tamanho da tela.

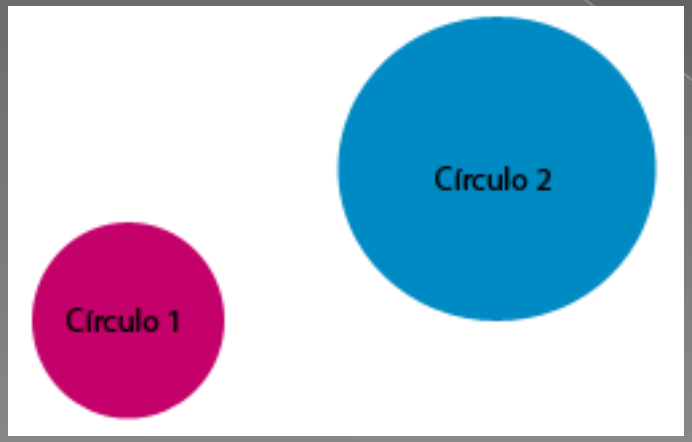
```
if((key[KEY_LEFT]) && (Quadrado.x > 1))
   {
        Quadrado.x = Quadrado.x - 2;
   }


   if((key[KEY_RIGHT])&&((Quadrado.x + Quadrado.img->w) < MAX_X-1))
   {
        Quadrado.x = Quadrado.x + 2;
   }

   if((key[KEY_UP])&&(Quadrado.y >1))
   {
        Quadrado.y = Quadrado.y - 2;
   }

   if((key[KEY_DOWN]) && ((Quadrado.y + Quadrado.img -> h) < MAX_Y -1))
   {
        Quadrado.y = Quadrado.y + 2;
   }
```

COLISÃO

Parte II – Colisão de círculos.

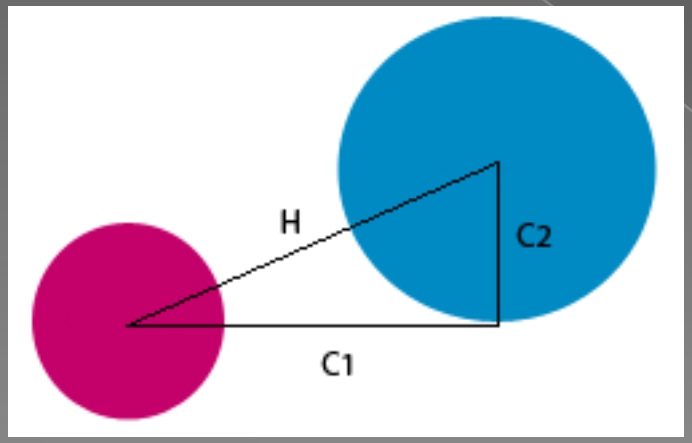
COLISÃO COM OUTROS BITMAPS - Círculos

- A colisão entre círculos é a mais simples de se fazer. Basta calcular a distância entre os dos centros. Se essa distância for maior do que a soma dos raios, não há colisão. Caso contrário, está havendo a colisão.



COLISÃO COM OUTROS BITMAPS - Círculos

- Para calcular essa distância entre os dois centros dos círculos, basta usar a fórmula de Pitágoras ($H^2 = C_1^2 + C_2^2$), onde C_1 é a distância no eixo X dos centros, C_2 é a distância no eixo Y e H é a distância entre os dois centros.



COLISÃO COM OUTROS BITMAPS - Círculos

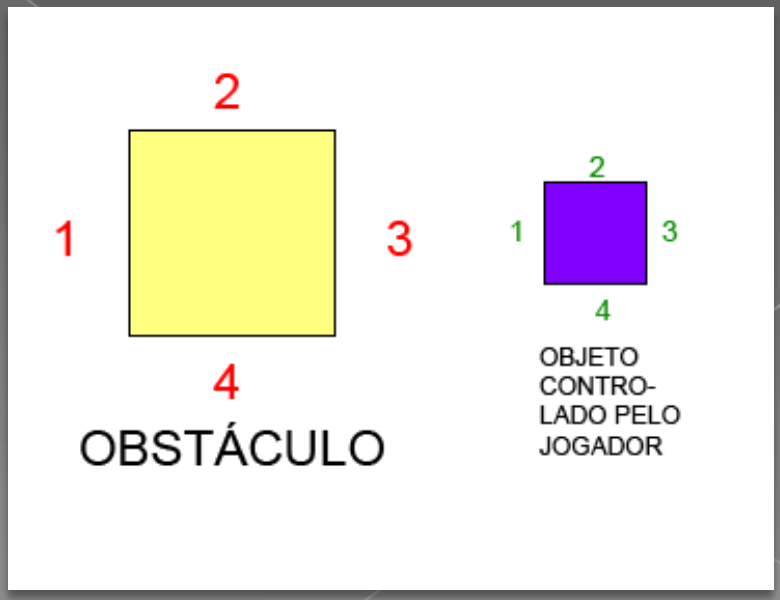
- Agora, basta fazer a análise no código:
  - Se a hipotenusa for menor do que a soma do raio do círculo 1 e do círculo 2, há colisão;
  - Senão, não há.

COLISÃO

Parte III – Colisão por Bounding Box.

COLISÃO COM OUTROS BITMAPS - Bounding Box

- Se pensarmos em dois bitmaps como dois retângulos, a colisão ocorre quando um dos retângulos "invade" a área do outro. Ou seja ...



COLISÃO COM OUTROS BITMAPS - Bounding Box

- Para que NÃO exista colisão entre esses bitmaps, as seguintes condições devem ser atendidas:

- O valor de X do quadrado roxo deve ser maior do que a soma do valor de X do quadrado amarelo com o valor de sua largura;
- O valor da soma de X do quadrado roxo mais a soma de sua largura devem ser menores do que o valor X do quadrado amarelo;
-
- O valor de Y do quadrado roxo deve ser maior do que a soma do valor de Y do quadrado amarelo com o valor de sua altura;
- O valor da soma de Y do quadrado roxo mais a soma de sua altura devem ser menores do que o valor Y do quadrado amarelo;

COLISÃO COM OUTROS BITMAPS - Bounding Box

- Passando essa lógica para uma função que retorne 0 se não houver colisão e, caso contrário, retorne 1, teremos:

```
int Colide()
{
	if(Quadrado[0].x > (Quadrado[1].x + Quadrado[1].img -> w))
		return 0;

	if((Quadrado[0].x + Quadrado[0].img -> w) < Quadrado[1].x)
		return 0;

	if(Quadrado[0].y > (Quadrado[1].y + Quadrado[1].img -> h))
		return 0;

	if((Quadrado[0].y + Quadrado[0].img -> h) < Quadrado[1].y)
		return 0;

	return 1;
}
```

COLISÃO COM OUTROS BITMAPS - Bounding Box

- Essa forma de colisão, apesar da simplicidade, possui um defeito um tanto grave: pelo fato de analisar retângulo com retângulo, a colisão pode ser imperfeita (geralmente detectando colisão em um ponto cujos bitmaps teoricamente não estão se tocando). Porém, é preciso lembrar que as partes que estão sendo ocultas pela cor transparente (makecol(255,0,255)) também fazem parte dos bitmaps e essas partes, nesses casos, estão causando a colisão.

COLISÃO

Parte IV – Considerações finais.

COLISÃO COM OUTROS BITMAPS – Considerações Finais

- Essas são as formas mais simples e utilizadas de colisão. Porém, não são as únicas. Para uma colisão de maior precisão, é utilizada a técnica Pixel Perfect (a qual consiste em uma análise pixel por pixel de todas as imagens na tela naquele momento).
- Há diversos jeitos (uns bem fáceis e outros um tanto complexos) de se implementar esse tipo de colisão. Uma idéia simples é fazer a análise através da cor do bitmap. Se, em um pixel qualquer da tela, a cor dos bitmaps que estão colidindo for diferente da cor usada para transparência, há colisão. Caso contrário, não há.

CONSIDERAÇÕES FINAIS

CONSIDERAÇÕES FINAIS

- Com as informações contidas nessa apresentação, é possível fazer um jogo básico. Porém, para coisas um pouco mais aperfeiçoadas, é necessário o interesse de quem está fazendo o jogo em buscar novas informações.
- Outra coisa que vale ser frisado é que a cópia de qualquer código aqui contido sem que a pessoa entenda o que ele quer dizer é completamente desaconselhado.

LINKS ÚTEIS

Alguns sites interessantes para pesquisar sobre o assunto.

LINKS ÚTEIS

- http://www.allegro.cc/ - site oficial da biblioteca;
- http://bdjogos.com/ - site com diversos tutoriais e exemplos;
- http://equipe.nce.ufrj.br/adriano/c/apostila/allegro/docs/allegro.html - site com explicações muito boas de funções e peculiaridades do allegro.