

UNIVERSIDADE ESTADUAL DE MS
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO
AEDI

Biblioteca

Professor: Nilton

E-mail: nilton@comp.uems.br

Biblioteca

- É uma coleção de componentes de programas que podem ser reutilizados em outros programas
- A maioria das linguagem tem bibliotecas
- *C Standard Library* é a biblioteca padrão de C
- Um programa em C conhece a existência de uma biblioteca através da diretiva *#include*
- *#include* torna um arquivo de cabeçalho “.h” parte da unidade de tradução.

Arquivos de cabeçalho da *C Standard Library*

- **<assert.h>**: Implementa ajuda na detecção de erros em versões de depuração de programas
- **<ctype.h>**: Funções para conversão de maiúsculas, minúsculas e outros tratamentos de caracteres, tais como teste dígito, letra, etc
- **<errno.h>**: Teste de códigos de erro reportados pelas funções de bibliotecas
- **<float.h>**: Define limites e precisão de variáveis de ponto flutuante
- **<limits.h>**: Define constantes para limites dos tipos de dados (_MIN, _MAX).

Arquivos de cabeçalho da *C Standard Library*

- **<locale.h>**: Especifica constantes de acordo com a localização específica, como moeda, data, etc
- **<math.h>**: Funções matemáticas comuns em computação
- **<setjmp.h>**: Define as macros *setjmp* e *longjmp* para saltos de chamadas e retornos de uma rotina para localizar erros graves no código
- **<signal.h>**: Implementa definições para receber e fazer o tratamento de sinais. Um sinal é um evento que ocorre durante a execução de um programa, por exemplo, divisão por zero, acesso inválido a memória, operação overflow, etc
- **<stdarg.h>**: Acesso dos argumentos passados para rotinas com parâmetro variável (*int printf(const char *__format, ...)*).

Arquivos de cabeçalho da *C Standard Library*

- **<stddef.h>**: Define vários tipos de dados e macros. Muitas dessas definições aparecem em outros arquivos de cabeçalho
- **<stdio.h>**: Tratamento de entrada/saída
- **<stdlib.h>**: Implementa funções para diversas operações, incluindo conversão, alocação de memória, controle de processo, funções de busca e ordenação
- **<string.h>**: Tratamento de strings
- **<time.h>**: Trata de tipos de data e hora.

Biblioteca de usuário em C

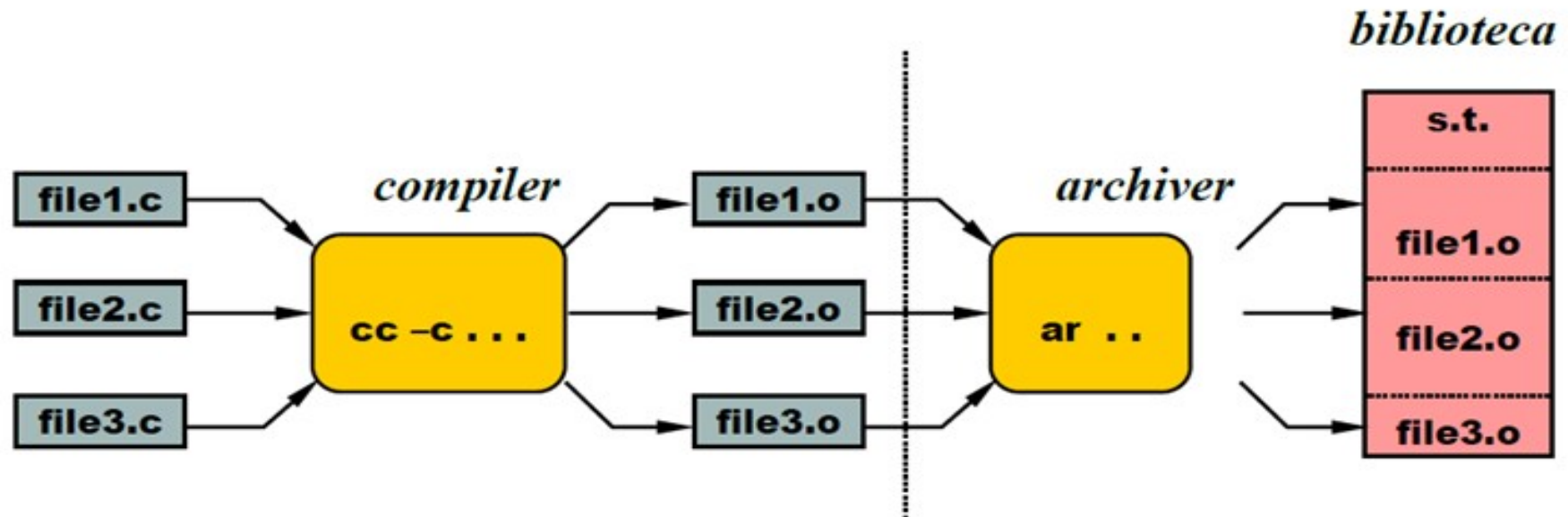
- É uma biblioteca **particular** para reutilização de seus componentes
- Cria-se uma rotina em arquivo e sua chamada em outros arquivo
- **Biblioteca estática**
 - O código da biblioteca é incluído dentro do executável
 - Ocupa mais memória
- **Biblioteca dinâmica**
 - O código da biblioteca é compartilhado com vários executáveis
 - Ocupa menos memória
- No Linux biblioteca estática (.a) e dinâmica (.so)
- No Windows biblioteca estática (.a) e dinâmica (.dll)

Biblioteca estática no Linux

.a

Criando uma biblioteca estática

- São duas fases:
 - Criar arquivos de códigos-objetos (*.o) usando a opção **-c** do compilador **cc**
 - Juntar os códigos-objetos na biblioteca estática usando o aplicativo **ar**



Criando a biblioteca estática libunits.a

- Ela será composta por três arquivos: **length.c**, **volume.c** e **mass.c**

length.c

```
const float VALOR=2.5;

float in_to_cm (float in) // inches to cm
{
    return (in * VALOR);
}
```

volume.c

```
float gal_to_l (float gal) // gallons to liters
{
    return (gal * 3.79);
}
```

mass.c

```
float oz_to_g (float oz) // ounces to grams
{
    return (oz * 28.35);
}
```

- Gerar os arquivos de códigos objetos: `$ cc -c length.c volume.c mass.c`
- Arquivar os objetos na biblioteca libunits.a: `$ ar r libunits.a length.o volume.o mass.o`

Preparando para usar a libunits.a

- É necessário criar um arquivo contendo os protótipos das funções existentes na biblioteca **libunits.a**. Esse arquivo será chamado de **units.h**.

units.h

```
float in_to_cm (float);  
float gal_to_l (float);  
float oz_to_g (float);
```

- O programa que usar qualquer função da biblioteca **libunits.a** precisa que seja incluído em seu início o arquivo **units.h** através da diretiva **#include**. Só assim, será satisfeito o princípio fundamental da linguagem C, ou seja, só se pode usar uma variável ou função se tiver sido declarada previamente.

Armazenando arquivos no Linux:

libunits.a e units.h

- O armazenamento dos arquivos **libunits.a** e **units.h** deve ser feito em local que seja de fácil acesso aos programadores

- Exemplo:

- Para bibliotecas estáticas

- /lib
 - /usr/lib

- Para arquivos de cabeçalho

- /include
 - /usr/include

`#include <units.h>`

`#include "units.h"`

- Se colocar units.h na pasta corrente de trabalho

- Se colocar units.h em “/home/users/projects/include”

`#include "/home/users/projects/include/units.h"`

Compilando um programa fonte

- Considere o programa código-fonte *convert.c*

```
#include <stdio.h>
#include "units.h"
int main(void)
{
    float inches, centimetros;
    printf ( "NORMAL Escreva um valor polegadas:" );
    scanf("%f", &inches);
    centimetros = in_to_cm(inches);
    printf("O seu valor em centimetros=%f\n",centimetros);
    return(0);
}
```

convert.c

- Usando a biblioteca

```
$ cc -o convert.exe convert.c libunits.a
```

- Sem usar a biblioteca

```
$ cc -o convert.exe convert.c length.o mass.o volume.o
```

Exercício

- Altere a constante do arquivo `length.c` para o valor 2.54, atualize a biblioteca estática e crie um novo executável. Para substituir uma parte de uma biblioteca estática pesquise o “man ar”.

Biblioteca dinâmica no Linux

.so

Criando uma biblioteca dinâmica

- São duas fases:
 - Criar arquivos de códigos-objetos (*.o) usando a opção **-c** e **-fPIC** do compilador cc
 - ❖ -fPIC permite que a biblioteca compartilhada possa ser carregada em qualquer endereço da memória
 - Juntar os códigos-objetos na biblioteca dinâmica usando o compilador cc

Criando a biblioteca dinâmica libunits.so

1. Gerar os arquivos de códigos-objetos:

```
$ cc -c -fPIC length.c volume.c mass.c
```

2. Arquivar os objetos na biblioteca **libunits.so**

```
$ cc -shared -fPIC -o libunits.so length.o volume.o mass.o
```


Compilação & Utilização

- Compilando usando a biblioteca

```
$ cc -o convert.exe convert.c libunits.so
```

- Preparar para usar a biblioteca

Problema em executar `convert.exe` ? Investigue a variável do shell `LD_LIBRARY_PATH`

```
$ LD_LIBRARY_PATH=$LD_LIBRARY_PATH:.  
$ export LD_LIBRARY_PATH
```

Tarefa: implemente uma *biblioteca estática* com as funcionalidades a seguir e construa o main usando essas funcionalidades

- Receber uma cadeia de caracteres e retornar a cadeia invertida
- Receber dia, mês, ano, e uma letra que corresponde ao formato de data a ser impresso. Se o formato = 'A', mostra DD/MM/AA; se 'B', mostra DD.MM.AAAA
- Receber uma cadeia de caracteres e retornar 'S' se a cadeia é um palíndromo ou 'N' caso contrário
- Receber um vetor de números e retornar o maior número
- Receber uma cadeia de caracteres, uma letra e uma posição e retornar a cadeia de caracteres com a nova letra na posição indicada. Os demais caracteres deverão ser reposicionados na cadeia


Tarefa: implemente uma *biblioteca dinâmica* com as funcionalidades a seguir e construa o main usando essas funcionalidades

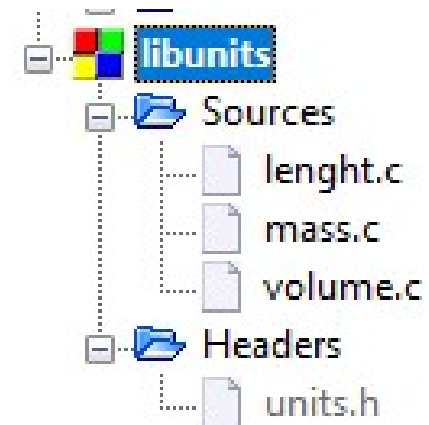
- Receber uma cadeia de caracteres e retornar a cadeia invertida
- Receber dia, mês, ano, e uma letra que corresponde ao formato de data a ser impresso. Se o formato = 'A', mostra DD/MM/AA; se 'B', mostra DD.MM.AAAA
- Receber uma cadeia de caracteres e retornar 'S' se a cadeia é um palíndromo ou 'N' caso contrário
- Receber um vetor de números e retornar o maior número
- Receber uma cadeia de caracteres, uma letra e uma posição e retornar a cadeia de caracteres com a nova letra na posição indicada. Os demais caracteres deverão ser reposicionados na cadeia

Biblioteca estática no Windows usando o CodeBlocks

.a

Criando biblioteca estática com o CodeBlocks

- *File* → *New* → *Project* → 
Static library
- Escrever o código fonte para as funções (*.c) e o arquivo de cabeçalho com os protótipos de função (*.h)



- Construir biblioteca (.a) (*Build* → *Build*)
 - *<pasta_projeto>\bin\Debug* ou *<pasta_projeto>\bin\Release*

Usando biblioteca estática

- *File* → *New* → *Project* →

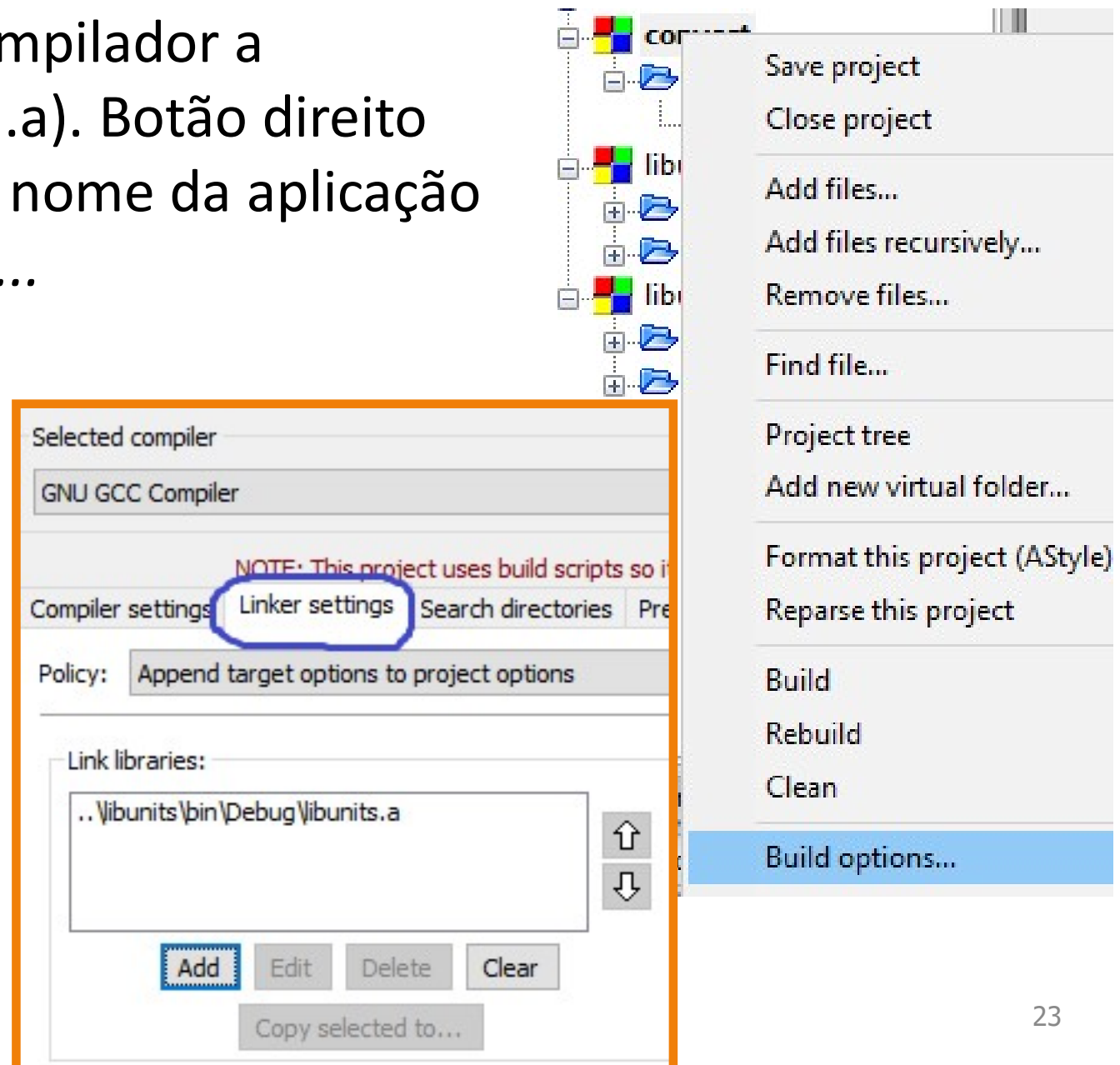


- Escrever o *main.c*, e não se esquecer do *#include ...* para incluir a referência da sua biblioteca particular
- Compilando...

```
----- Build: Debug in convert (compiler: GNU GCC Compiler)-----  
i686-w64-mingw32-gcc-8.1.0.exe -Wall -g -Wall -Og -g -c C:\Users\Nilton\Desktop\convert\main.c -o obj\Debug\main.o  
i686-w64-mingw32-g++.exe -o bin\Debug\convert.exe obj\Debug\main.o  
obj\Debug\main.o: In function `main':  
C:/Users/Nilton/Desktop/convert/main.c:11: undefined reference to `in_to_cm'  
collect2.exe: error: ld returned 1 exit status  
Process terminated with status 1 (0 minute(s), 0 second(s))  
2 error(s), 0 warning(s) (0 minute(s), 0 second(s))
```

Usando biblioteca estática

- Informar ao compilador a localização da (.a). Botão direito mouse sobre o nome da aplicação e *Build options...*
- Adicionando a localização no *Linker settings*
- Compilar
- Executar



Biblioteca dinâmica no Windows usando o CodeBlocks

.dll

Criando biblioteca dinâmica com o CodeBlocks

- O processo é muito similar ao apresentado para a construção de uma biblioteca estática no Windows

- Usar o modelo criado pelo
File → New → Project →



- Onde a .dll deve estar quando for executar a aplicação?
 - O diretório do qual o aplicativo foi carregado
 - O diretório do sistema. Use a [função GetSystemDirectory](#) (C++) para obter o caminho desse diretório (C:\Windows\System32)
 - O diretório do sistema de 16 bits. Não há nenhuma função que obtém o caminho deste diretório, mas ele é pesquisado
 - O diretório do Windows. Use a [função GetWindowsDirectory](#) (C++) para obter o caminho desse diretório (C:\Windows)
 - O diretório atual
 - Os diretórios listados na variável de ambiente PATH. **Note** que isso não inclui o caminho por aplicativo especificado pela chave de registro App Paths.