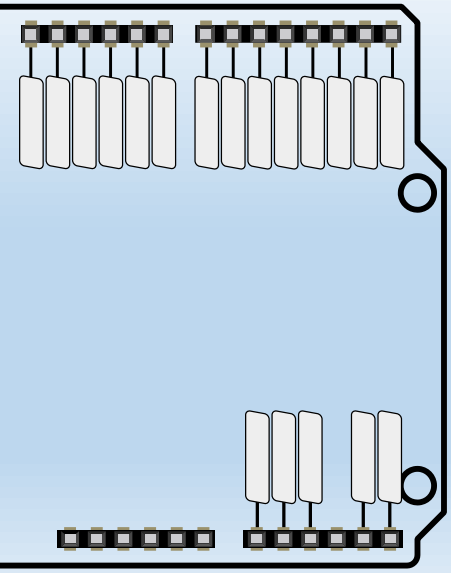
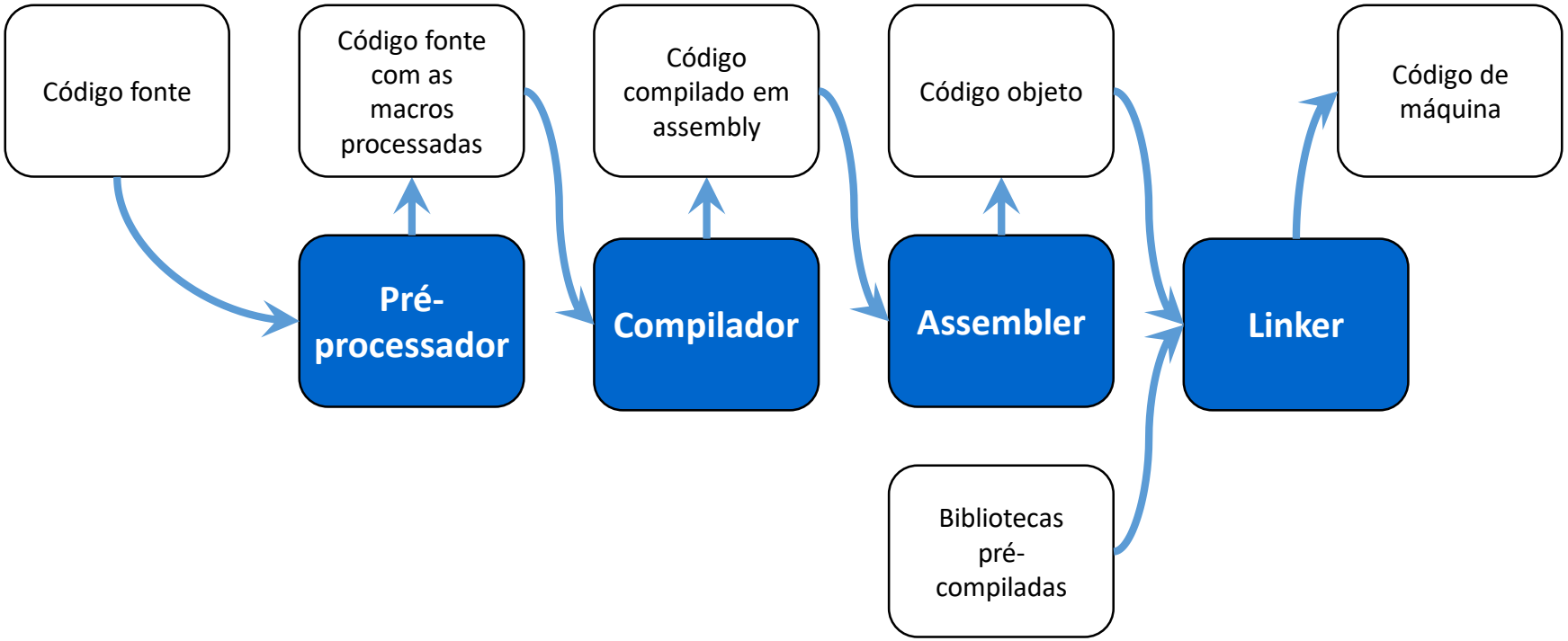
Linguagem C

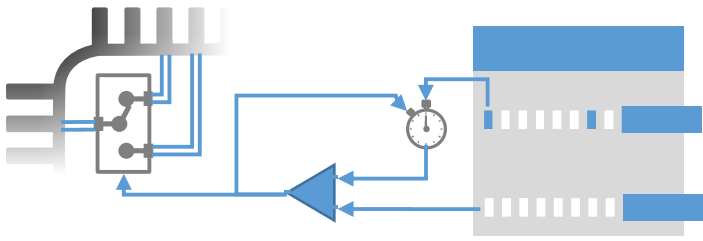


C é peculiar, cheia de falhas, e um enorme sucesso.

Dennis M. Ritchie

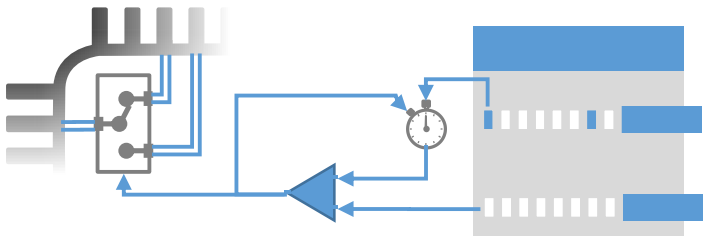
Processo de compilação





Linguagem C

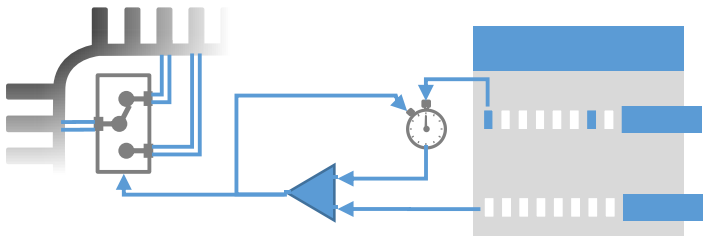
- Um programa C é uma coleção de funções criadas pelo programador ou funções de biblioteca
- Os programas em linguagem C possuem vários componentes
  - Diretivas de pré compilação
  - Definição de variáveis
  - Definição de funções



Diretivas de pré-compilação

- São mensagens que o programador envia ao compilador para que este execute alguma tarefa, antes ou no momento da compilação
  - as diretivas são iniciadas pelo caractere #
  - as diretivas mais comuns são #include e #define, ambas utilizadas para especificar bibliotecas de funções a serem incorporadas na compilação

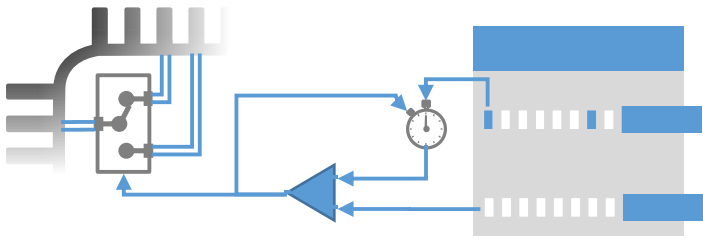
```
//Diretivas de compilação
#include <biblioteca.h>
#define macros
#define labels
```



Diretivas de pré-compilação

- A diretiva #define é utilizada para que o código fonte seja modificado antes de ser compilado.

```
#define CONST 15
void main(void){
	printf("%d", CONST * 3);
}

//depois de compilado
void main(void){
	printf("%d", 15 * 3);
	//é possível: printf("%d", 45);
}
```

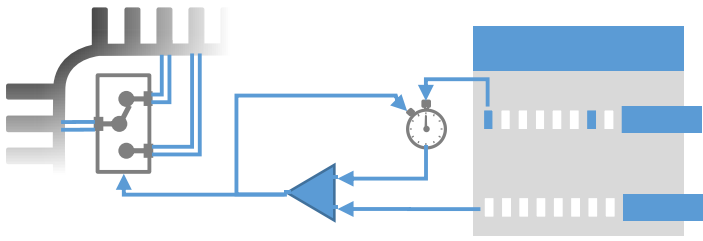


Compilação condicional

```
void MostraSaidaPadrao(){
#ifdef PADRAO Serial
	char * msg = "SERIAL";
#else
	char * msg = "LCD";
#endif
	printf(msg);
}
```

```
#include <stdio.h>
#define PADRAO Serial
void main(void){
	MostraSaidaPadrao();
}
```

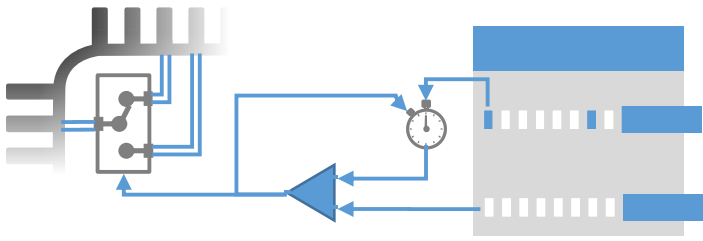
```
#include <stdio.h>
#define PADRAO LCD
void main(void){
	MostraSaidaPadrao();
}
```



Compilação condicional

- No momento da compilação o pré-compilador irá verificar se a "tag" LCD foi definida em algum lugar. Em caso positivo o pré-compilador irá deixar tudo que estiver entre o #ifdef e o #else e retirará tudo que está entre o #else e o #endif.}

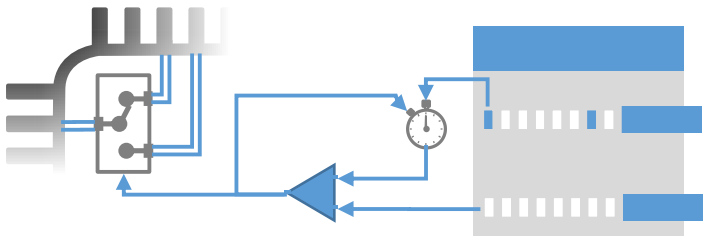
```
void ImprimirTemp(char valor){
#ifdef LCD
        Imprime_LCD(valor);
#else
        if (valor > 30){
                led = 1;
        }else{
                led = 0;
        }
#endif //LCD
}
```



Includes

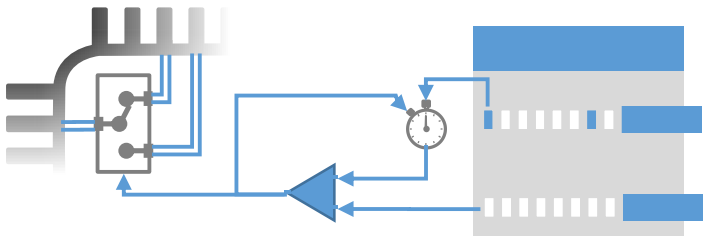
- Servem para especificar ao compilador que deseja-se usar novas funções, tipos e macros que estão disponíveis em outros arquivos
- Como a linguagem C tem uma grande variedade deste funções e definições é comum que elas sejam agrupadas em arquivos diferentes, de acordo com a natureza das tarefas que elas executam

```
#include <nome do arquivo>
#include "nome do arquivo"
```



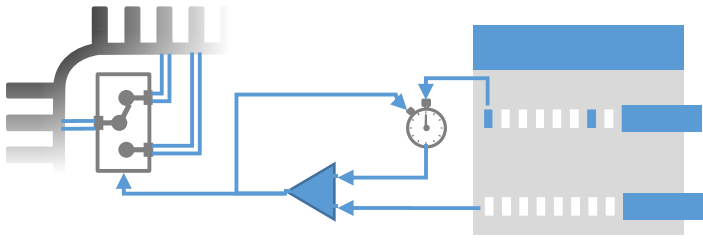
Includes

- A diferença entre as duas formas está no local onde o compilador vai procurar o arquivo no momento da compilação
  - #include<nome do arquivo>
    - o arquivo é procurado no diretório definido pelo compilador C, como sendo aquele que contém os header files
  - #include "nome do arquivo":
    - é usado quando deseja que o compilador busque o arquivo especificado no mesmo diretório do arquivo que está sendo criado
    - esta forma é usada quando deseja-se incorporar arquivos criados e salvos pelo programador no mesmo diretório atual
- Header files: são os arquivos com extensão .h que contém as definições de tipos, dados e várias funções já prontas



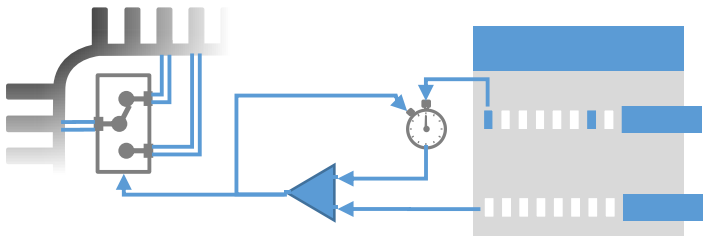
Arquivos .c e .h

- Arquivo de código (code)
  - terminado com a extensão .c
  - contém a implementação do código
  - é compilado gerando um arquivo .o
- Arquivo de cabeçalho (header)
  - terminado com a extensão .h
  - contém apenas defines e protótipos
  - não é compilado



Exemplo de arquivo .c

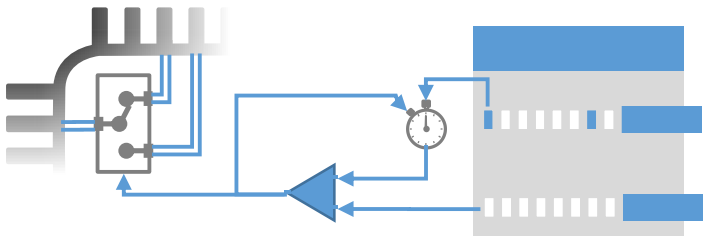
```
//variável usada apenas dentro deste arquivo
static char temp;
//variável que será usada também fora do arquivo
static char valor;
//funções usadas dentro e fora do arquivo
void MudaDigito(char val){
      valor = val;
}
char LerDigito(void){
      return valor;
}
void InicializaDisplays(void){
      //código da função
}
//função usada apenas dentro deste arquivo
void AtualizaDisplay(void){
      //código da função
}
```



Exemplo de arquivo .h

- Não existe a função AtualizaDisplay()
- A variável "digito" só pode ser lida ou gravada pelas funções MudaDigito() e LerDigito();

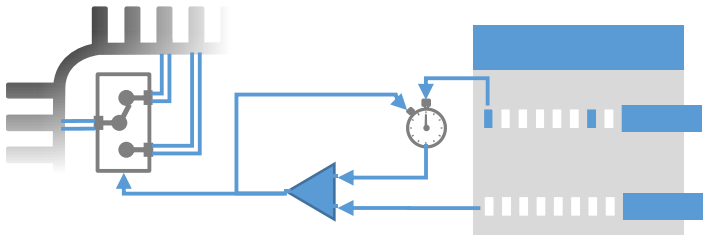
```
#ifndef VAR_H
#define VAR_H
	void MudaDigito(char val);
	char LerDigito(void);
	void InicializaDisplays(void);
#endif //VAR_H
```



Comentários

- Podem e devem estar em qualquer ponto do programa.
  - Entre os delimitadores /* e */ para um bloco de comandos
  - Após // para comentar até o final da linha

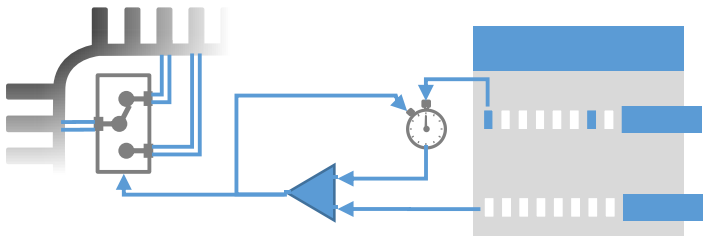
```
/* Programa 01 */
/* Função: descrição */
// Autor: nome
```



Definições Globais:

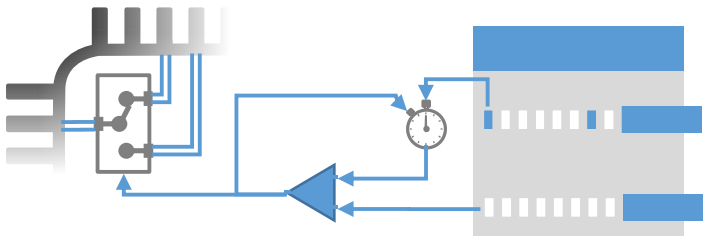
- São especificações de constantes, tipos e variáveis que serão válidas em todas as funções que formam o programa
  - Embora sejam de relativa utilidade, não é uma boa prática de programação definir muitas variáveis globais
  - Podem ser acessadas em qualquer parte do programa, um descuido na alteração dos seus valores, pode provocar problemas em muitos outros locais

```
// Seção de variáveis globais
char variavelGlobal;
```



Blocos de comando:

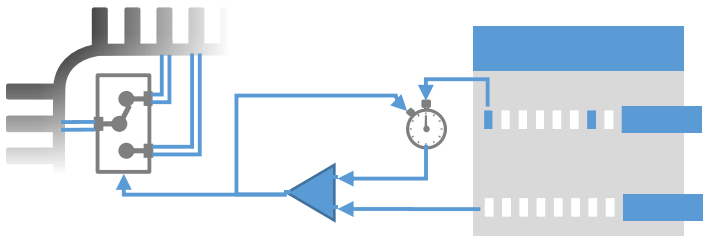
- São grupos de comandos que devem ser tratados como uma unidade lógica
  - O início de um bloco em C é marcado por um "{" e o término por um "\"
  - O bloco de comando serve para criar um grupo de comandos que devem ser executados juntos
- Usa-se o bloco de comandos quando se usa comandos de teste em que deve-se escolher entre executar dois blocos de comandos
- Um bloco de comandos pode ser utilizado em qualquer trecho de programa que se pode usar um comando C
- Os comandos de controle especificam a ordem em que a computação é feita no programa



Protótipos de funções

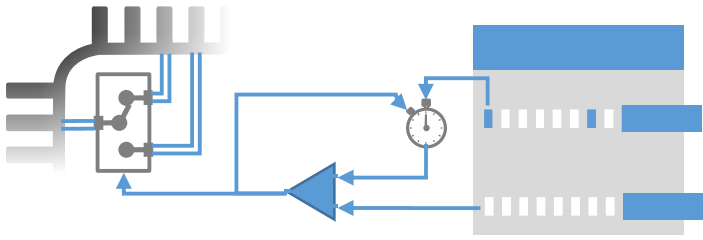
- Não são obrigatórios. São usados pelo compilador para fazer verificações durante a compilação
  - ver se as partes do programa que acionam as funções o fazem de modo correto, com o nome certo, com o número e tipo de parâmetros adequados

```
// Seção de protótipo de funções
void funcao01(char var);
int funcao02(void);
```



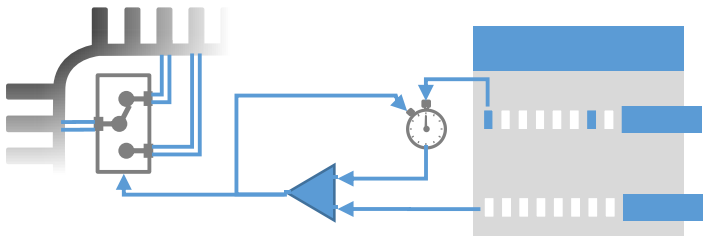
Definições de funções

- São os blocos do programa onde são definidos os comandos a serem executados em cada função
  - A função pode, ou não, receber valores que serão manipulados em seu interior
  - Após o processamento, as funções podem, se necessário retornar um valor
  - É obrigatório a presença de pelo menos uma função com o nome main, e esta será a função por onde começa a execução do programa
  - Não há ordem obrigatória para codificar as funções



Definições de funções

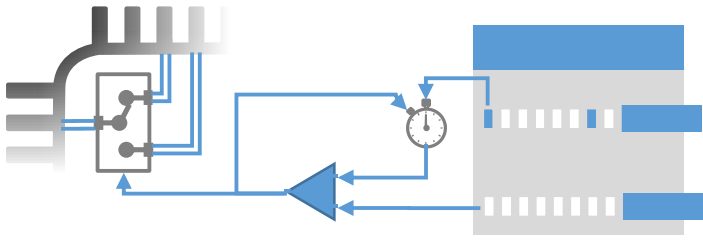
```
// Seção de definição de funções
int main(int argc, char *argv[]){
        return 0;
}//end main

void funcao01(char var){
        ...
}//end funcao01

void funcao02(void){
        ...
}//end funcao02
```



Função main

Função main

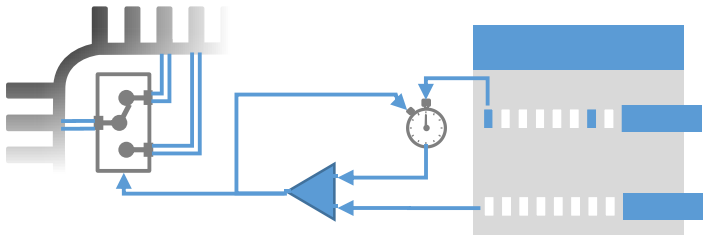
- O meio de indicar o ponto de início de um programa depende do compilador.
  - Geralmente apenas a criação da função main já é suficiente para o compilador
- O linker aloca a função main() em algum lugar disponível na memória
- O linker atualiza o vetor de reset colocando um pulo para a função main



Função main

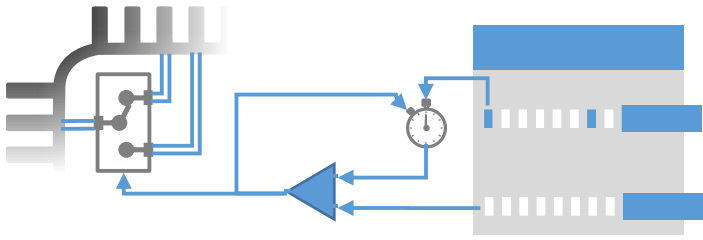
- Para sistemas embarcados a função main é a primeira a ser executada, portanto não pode receber parâmetro nenhum. Como ela não é chamada por nenhuma outra função, ela é a última a ser executada. Deste modo não existe a possibilidade de retornar nenhum valor. Deve ser declarada como:

```
void main (void){
        //aqui entra o código do programa
}
```



Função main

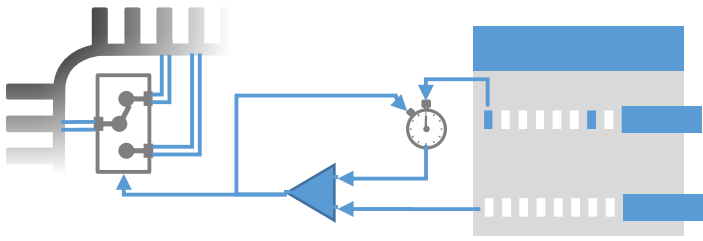
- Geralmente os sistemas embarcados são feitos para serem continuamente executados
- As rotinas devem ser cíclicas
- O sistema só para quando desligado
- Atenção!
  - Apenas nesta situação devemos utilizar loops infinitos.



Função main

```
void main (void){
        for(;;){
                //aqui entra o
                //código principal
        }
}
```

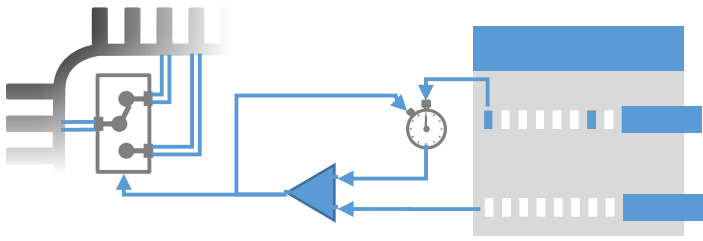
```
void main (void){
        while(1){
                //aqui entra o
                //código principal
        }
}
```



Wiring/Main

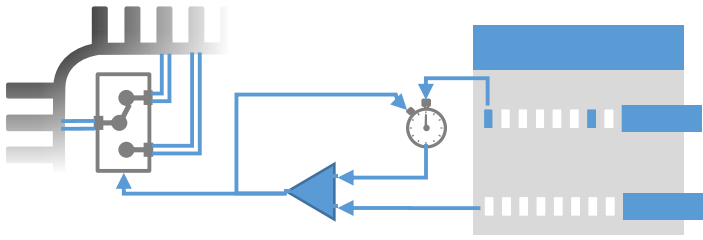
- As plataformas baseadas em Wiring não apresentam a função main, ao invés disso elas funcionam com a função setup e loop.
- Estas funções possuem uma relação com a função main e com o loop infinito como no código abaixo

```
void main(void){
        setup();
        for (;;) {
                loop();
        }
}
```



Entrada/saída de dados

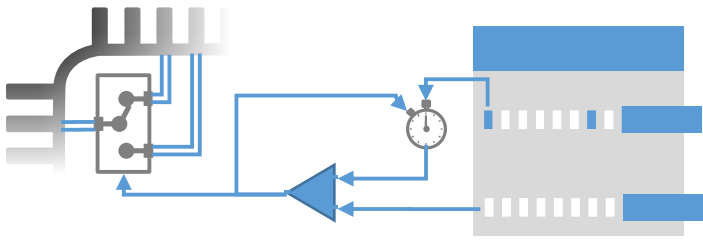
- A placa de desenvolvimento utilizada possui um caminho de comunicação serial que pode ser utilizado para receber ou enviar informações para o computador.
- Além da comunicação serial a placa de desenvolvimento possui um LCD e um display de 7 segmentos com quatro dígitos para exibir informações e um teclado com 10 botões para receber informações. Todos os exemplos apresentados utilizarão algum destes dispositivos de exibição.



Entrada/saída de dados

- O código a seguir realiza a leitura das teclas do teclado e imprime no LCD, sempre na posição inicial.

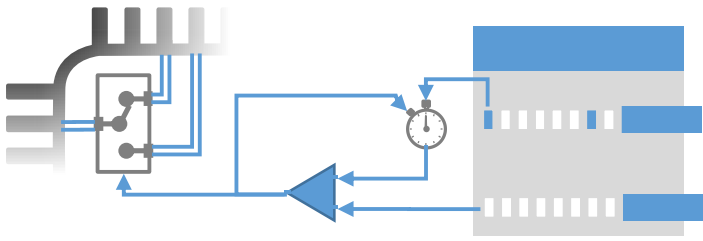
```
#include "lcd.h"
#include "keypad.h"

void main (void){
        int tecla;
        kpInit();
        lcdInit();
        for(;;){
                kpDebounce();

                tecla = kpReadKey();
                lcdPosition(0,0);
                lcdNumber(tecla);
        }
}
```



Entrada/saída de dados

- Utilizando as bibliotecas com arduino

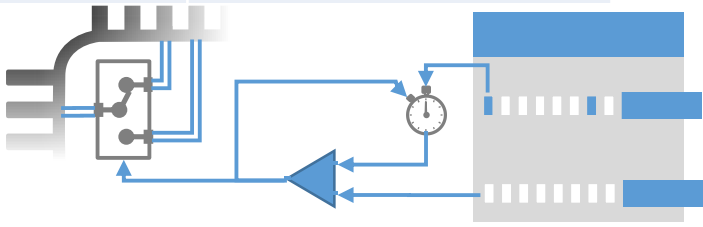
```
#include "lcd.h"
#include "keypad.h"

void setup (void){
        kpInit();
        lcdInit();
}

void loop(void){
        int tecla;
        kpDebounce();

        tecla = kpReadKey();
        lcdPosition(0,0);
        lcdNumber(tecla);
}
```



Operadores

- A linguagem C tem uma grande quantidade de operadores
- Os operadores podem ser divididos em grupos como: aritméticos, aritméticos de atribuição, incremento, decremento, relacionais e lógicos
  - Operadores aritméticos: soma e subtração têm a mesma prioridade, que é menor do que a multiplicação, divisão e resto da divisão

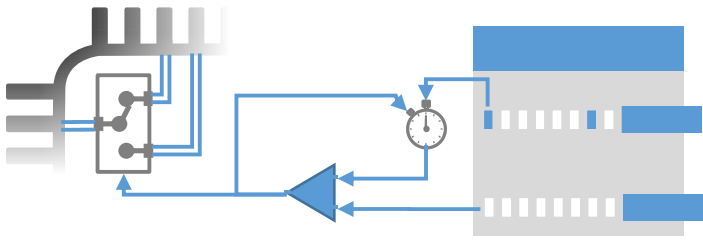
Operador	Descrição	Operador	Descrição
=	Atribuição	+	Soma
-	Subtração	*	Multiplicação
/	Divisão	%	Resto da divisão



Operadores

```
int a = 1, b = 3, c = 5;
int l, m, n, o, p;
```

Código	Interpretação	Resultado
l = 17 % c;	l = 17 % 5;	l = 2;
m = 15 + b;	m = 15 + 3;	m = 18;
n = 23 - a;	n = 23 - 1;	n = 22;
o = 18 / b;	o = 18 / 3;	o = 6;
p = 4 * c;	p = 4 * 5;	p = 20;



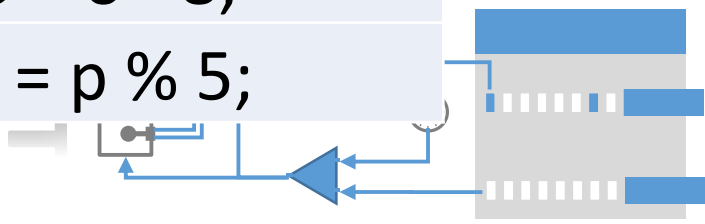
Operadores

- Pode-se combinar os operadores aritméticos (+, - ,*, /, %) com o operador de atribuição da seguinte forma:

```
int a = 1, b = 3, c = 5;
int l, m, n, o, p;
```

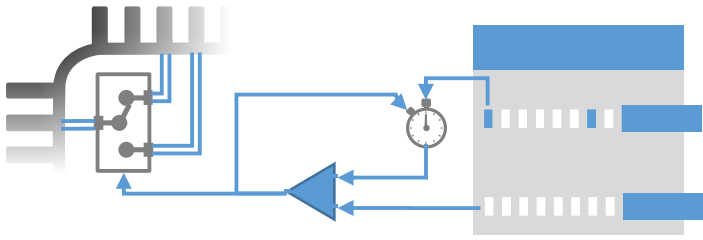
- Este tipo de contração é muito utilizado na linguagem C, pois facilita a escrita

Contração	Expandido
l *= 4;	l = l * 4;
m /= 2;	m = m / 2;
n += 5;	n = n + 5;
o -= 8;	o = o - 8;
p %= 5;	p = p % 5;



Operadores

- Em C existem dois operadores específicos para incremento/decremento de varíaveis:
  - ++ incrementa de 1 seu operando
  - -- decrementa de 1 seu operando
- São utilizados para realizar contagens progressivas ou regressivas
- Trabalham de dois modos:
  - pré-fixado: o operador aparece antes do nome da variável. Exemplo: ++n; onde n é incrementado antes de seu valor ser usado
  - pós-fixado: o operador aparece após o nome da variável. Exemplo: n++, onde n é incrementado depois de seu valor ser usado.



Operadores

```
int a = 4;
```

Comando	Interpretação	Resultado
a++;	a = a + 1;	a = 5;
a-- --;	a = a - 1;	a = 3;
l = 2 * a++;	l = 2 * 4;	l = 8;
m = 2 * ++a;	m = 2 * 5;	m = 10;