Linguagem C
Ficheiros – Compilação Separada

Struct, union e typedef

Compilação separada e módulos

Estruturas

- Uma estrutura (**struct**) em C corresponde, em grosso modo, a uma **classe sem métodos** em Java
- Permite agregar dados de diferentes tipos
- Os **campos** da estrutura são acedidos como os membros de uma classe

```
struct data { int dia, mes, ano; };
struct funcionario {
   unsigned long num_interno;
   char nome[100];
   struct data data_admissao;
};
```

Estruturas - Exemplo de uso

```
struct data d;
struct funcionario f;

d.dia = 5; d.mes = 10; d.ano = 1910;

f.num_interno = 1024;
f.data_admissao = d;
strcpy(f.nome, "Belmiro");

f.data_admissao.dia = 6;

d = f.data_admissao;
```

Campos de bits (bitfields)

- Podemos ter membros de uma estrutura com um número de definido de bits
- Exemplo – representação de um número real:

```
struct IEEE754
{ unsigned int mantissa : 23;
  unsigned int expoente : 8;
  unsigned int sinal : 1;
};
```

- A ordem tem a ver com a representação interna dos números (little-endian)

Union

- Uma union é uma struct em que os campos estão sobrepostos na memória
- Exemplo – ver a representação interna de um número real:

```
union conv
{   struct IEEE754 bits;
    float num;
} n;
...
scanf( "%f", &n.num );
printf( "%x\n%x\n%x\n", n.bits.sinal,
        n.bits.expoente, n.bits.mantissa );
```

sizeof – struct e union

- Relembrar – sizeof(X) é o número de bytes ocupado por X
- Para uma struct S com campos S_1, S_2 ... S_n:

$$sizeof(S) \geq sizeof(S_1) + sizeof(S_2) + \ldots + sizeof(S_n)$$

- Pode haver "espaço em branco" dentro das structs!
- Para uma union U com campos U_1, U_2 ... U_n:

$$sizeof(U) \geq \max(sizeof(U_1), sizeof(U_2), \ldots sizeof(U_n))$$

typedef

- Podemos definir novos tipos em C!
- Exemplo – tipo booleano:

```
typedef boolean int;

// definimos um novo tipo – boolean – que
// pode ser usado como os tipos nativos

boolean terminar;
boolean e_par( int num );
```

typedef – mais exemplos

- Definir um tipo como uma struct anónima:

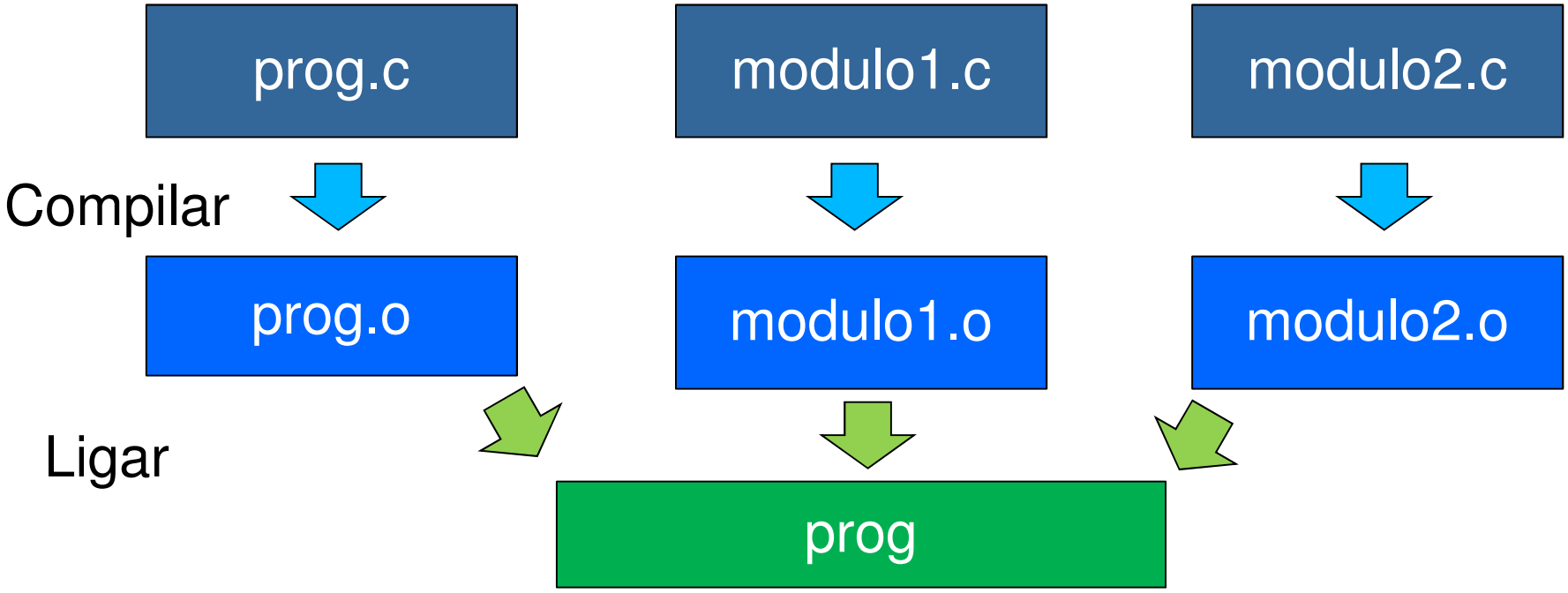
```
typedef data
struct { int dia, mes, ano; };
...
data d;
d.dia = 5;
```

- FILE é definido com typedef! Em stdio.h!

```
typedef FILE ???
```

Compilação Separada

- Podemos combinar vários módulos em C num único programa executável:



- Apenas o programa principal pode ter função main!

Ficheiros .h

- Servem como “interfaces” para os módulos
- Devem conter:
  - ✓ Protótipos de funções
  - ✓ Definições de tipos
  - ✓ Declarações extern de variáveis globais
- Usados na directiva include

```
#include "modulo.h"
```

  - ✓ Para incluir as definições dos nossos módulos

```
#include <modulo_sys.h>
```

  - ✓ Para incluir as definições de módulos do sistema
  - ✓ Como: stdio.h, stdlib.h, string.h, etc
- **Não são compilados de forma independente!**

Static

- Denota variáveis locais às funções que devem ser estáticas (mantém o valor entre chamadas da função)
- Também serve para denotar funções ou variáveis globais privadas do módulo
  - ✓ são invisíveis fora do ficheiro onde está a declaração

- Variáveis globais são variáveis declaradas fora das funções
  - ✓ Visíveis por todas as funções
  - ✓ Não devem ser usadas, a menos que sejam realmente necessárias

Exemplo: Módulo contador

contador.h

```
void init_contador(void);
void incr_contador(void);
int  ler_contador(void);
```

contador.c

```
#include "contador.h"
static int val;

void init_contador() { val = 0; }
void incr_contador() { val++; }
int  ler_contador()  { return val; }
```

Módulo Contador – programa principal

prog.c:

```
#include <stdio.h>
#include "contador.h"

int main()
{  int n, i;
   scanf( "%d", &n );
   init_contador();
   while( (i = ler_contador()) < n )
   {   printf( "%d\n", i );
       incr_contador();
   }
   return 0;
}
```

Compilação separada - comandos

- Podemos compilar todos os módulos num único comando:

```
gcc –Wall –g –o prog main.c contador.c
```

- Ou separadamente:
  - ✓ Compilar:

```
gcc –Wall –g -c main.c
gcc –Wall –g -c contador.c
```

  - ✓ Opção -c:
    - ✓ compilar o ficheiro mas não linkar
    - ✓ gera ficheiro objecto (.o)
  - ✓ "Linkar":

```
gcc –o prog main.o contador.o
```