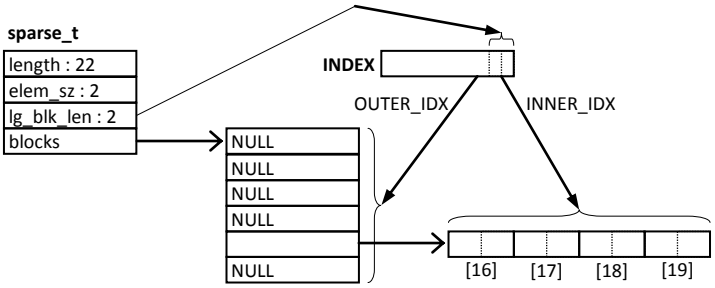
1. [10] O tipo `sparse_t` representa *arrays* cujo espaço de armazenamento interno é alocado apenas para blocos em que há índices efectivamente utilizados. Os blocos correspondem a sequências de índices consecutivos, todas da mesma dimensão, que é sempre uma potência inteira de 2. Cada instância contém o comprimento total do *array* representado (`length`), o tamanho em *bytes* de cada elemento (`elem_sz`), o log_2 do comprimento dos blocos internos (`lg_blk_len`) e o *array* de blocos (`blocks`). A imagem apresenta o exemplo de uma instância de `sparse_t` acedida apenas no índice 17.



Analise as definições em C apresentadas ao lado, que serão a base para as respostas às alíneas seguintes.

sparse.h

```
#ifndef SPARSE_H
#define SPARSE_H

typedef struct sparse {
   unsigned length;     /* capacidade máxima */
   unsigned elem_sz;    /* sizeof de cada entrada */
   unsigned lg_blk_len; /* núm. bits p/a indexar blocos */
   void * * blocks;
} sparse_t;

/* Inicializa instância. */
void init_sparse(sparse_t * obj, unsigned length,
                 unsigned elem_sz, unsigned lg_blk_len);

/* Liberta recursos internos. */
void cleanup_sparse(sparse_t * obj);

/* Calcula endereço da posição index. */
void * sparse_ptr_to(sparse_t * obj, unsigned index);

/* Invoca func para cada índice dos blocos existentes. */
int sparse_foreach(sparse_t * obj, void * context,
    void (*func)(int index, void * item, void * context));

#endif
```

sparse.c

```
#include "sparse.h"

#define OUTER_IDX(SP_PTR, IDX) ...
#define INNER_IDX(SP_PTR, IDX) ...
#define NUM_BLOCKS(SP_PTR) \
    (OUTER_IDX(SP_PTR, (SP_PTR)->length - 1) + 1)

...
void cleanup_sparse(sparse_t * obj) {
  unsigned i, blocks_len = NUM_BLOCKS(obj);
  for (i = 0; i < blocks_len; ++i) { ... }
  ...
}
void * sparse_ptr_to(sparse_t * obj, unsigned index) {
  unsigned oidx = OUTER_IDX(obj, index);
  unsigned iidx = INNER_IDX(obj, index);
  if (obj->blocks[oidx] == NULL) alloc_block(obj, oidx);
  return ((char *)(obj->blocks[oidx])) + iidx * obj->elem_sz;
}
```

a) [1.25] Implemente a operação `init_sparse`, que inicializa uma instância de `sparse_t`, deixando o *array* `blocks` criado, embora vazio.

b) [1.25] Em `sparse_ptr_to` utiliza-se `alloc_block`, que aloca um novo bloco e o adiciona a `blocks`. Construa essa função, deixando-a inacessível a outros módulos.

c) [1.25] Apresente a versão completa da operação de libertação de recursos, `cleanup_sparse`.

d) [1] Defina a macro `INNER_IDX` para retornar o valor correspondente aos `lg_blk_len` *bits* de menor peso de `IDX`, que servirá para indexar dentro de um bloco.

e) [0.5] Defina a macro `SP_AT` para simplificar o uso de `sparse_ptr_to`, permitindo usos como:

```
/* pseudo-código: double value = psp1[3]; */
double value = SP_AT(psp1, 3, double);
/* pseudo-código: psp2[37] = 8; */
SP_AT(psp2, 37, int) = 8;
```

f) [1.25] Sejam `nb`, `ni` e `psp` variáveis globais, em que `nb` e `ni` são inteiros e `psp` é um ponteiro para um `sparse_t` que armazena inteiros em blocos de 32, apresente *assembly* IA-32 correspondente à instrução:

```
((int *)(psp->blocks[nb]))[ni] = 42;
```

NOTA: admita que todos os registos genéricos estão disponíveis

g) [3.5] Implemente, em *assembly* IA-32, a função `sparse_foreach`, que invoca `func` para cada item de cada bloco existente, passando como argumento o índice global correspondente à posição corrente, o endereço da posição corrente e o ponteiro recebido em `context`. A função deve retornar o número de invocações que fez a `func`.

2. [2] Considere que da compilação de `f1.c` e `f2.c` resultam, respectivamente, `f1.o` e `f2.o`.

f1.c

```
extern char id[];
extern int val;

void show_id() { puts(id); }

void show_val() {
   printf("%d\n", val);
}
```

f2.c

```
int id = 0x00435350;
int val = 88;

int oper(int v) {
   int val = v * 10;
   show_val();
   return val;
}
```

a) [1] `id` impede a ligação de `f1.o` com `f2.o`? Se sim, porquê? Se não, o que mostra `show_id()`?

b) [1] `val` impede a ligação de `f1.o` com `f2.o`? Se sim, porquê? Se não, o que mostra `oper(5)`?

3. [1.5] Considere um sistema com arquitectura IA-32 e uma cache de dados *4-way set-associative*, sendo 5 *bits* do endereço usados como *offset*, e responda às seguintes questões justificando devidamente as respostas.

```
struct addr {
   int  zip;
   int  number;
   char street[120];
};
```

a) [0.5] Sabendo que a *cache* armazena no máximo 512 instâncias distintas de `struct addr`, quantos *bits* são utilizados para determinar o *set*?

b) [1] Considere o cenário em que o objecto `addrObj`, do tipo `struct addr`, reside num endereço múltiplo de 1024 e **não** está presente em *cache*. Indique, para o código ao lado, que acessos aos campos de `addrObj` não encontrarão os dados na *cache* (*misses*).

```
int  z = addrObj.zip;
int  n = addrObj.number;
char x = addrObj.street[23];
char y = addrObj.street[24];
```

4. [5] Considere os seguintes tipos e os respectivos campos e métodos:

Tipo:	**Object**	**Number**	**Integer**
Deriva de:	--	Object	Number
Campos:	--	--	int value;

Métodos:	typename:
	  return this->vptr->name;
	print:
	  printf("%s@0x%08x",
	  typename(), this);
		typename:
		  (herdado)
		print:
		  (herdado)
		intValue:
		  (abstracto – não implementa)
		byteValue:
		  return (char)intValue();
			typename:
			  (herdado)
			print
			  printf("%d", this->value);
			intValue
			  return value;
			byteValue
			  (herdado)

Apresente:

- definições de estruturas correspondentes aos três tipos e às respectivas tabelas de métodos virtuais
- declarações (sem implementação) das funções estritamente necessárias para o preenchimento das tabelas de métodos
- definições (criação das instâncias) das tabelas de métodos virtuais dos três tipos
- funções de iniciação para os três tipos

```
void init_Object(Object * this);
void init_Number(Number * this);
void init_Integer(Integer * this, int value);
```

- função de instanciação dinâmica de Integer:

```
Integer * new_Integer(int value);
```

5. [1.5] No caso de se ter tomado a decisão de colocar parte do código e dados de um programa numa biblioteca de ligação dinâmica, indique e justifique que critérios genéricos utilizaria para optar por carregamento implícito (via *linker*) ou explícito (via *dlopen*) dessa biblioteca.

Duração: 2 horas e 30 minutos

Bom teste!