Consider a linear Data Structure (DS), where the insertion of homogeneous data should occur according to the characteristics indicated below. The DS is generic, with the data size provided at the time of DS creation. Upon inserting the first data, a node of type **node_t** is allocated. This node has a reference to a contiguous region of the main memory, a field called **data** with a capacity for 4 data elements. We say that this node has a total capacity of 4 data elements, and the current total capacity of the DS is 4 data elements. When the fifth data element is added, a new node with a total capacity of 8 data elements is allocated, and the fifth data element is stored in this new node. The new node has a total capacity for 8 data elements, and the current total capacity of the DS becomes 12 data elements. Subsequently, the DS grows whenever necessary during insertion, allocating a new node with twice the total capacity of the previously allocated node.

Consider the descriptor, **desc_s** (see Figure), of this structure, which stores the data size in **size_data**. This value is received as an argument in the **create_array** creation function. The descriptor also has a reference, **head**, to the first node. In the node structure, **node_s** (see Figure), **capacity** indicates the node's capacity (4, 8, 16, 32, 64, ...), **qtd** indicates the number of inserted data elements in the node, and **next** is a pointer to the next node.

Implement the following functions:

desc_t* array_create(int size_data) which initialises (creates) this DS, allocating and initialising the descriptor.

int array_insert(array_t *p, void *data) which inserts a new item into this DS as described, and inserts new nodes as necessary.

The implementation should be in the C programming language. You can use the functions malloc, free, and memcpy. Any other auxiliary functions should also be implemented. The test code provided above (see Figure 4) should produce the following output:

```
0 1 2 3
4 5 6 7 8 9 10 11
12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27
```

```c
typedef struct node_s {
    int capacity;
    int qtd;
    struct node_s *next;
    void *data;
} no_t;

typedef struct desc_s {
    int size_data;
    no_t *head;
} desc_t;
```

```c
void printdata(desc_t *p) {
    no_t* node = p->head;
    while (node != NULL) {
        for (int i = 0; i < node->qtd; i++)
            printf("%i ", ((int*) no->data)[i]);
        printf("\n");
        node = node->nexr;
    }
}

int main() {
    desc_t *p = array_create(sizeof(int));
    for (int i = 0; i < 28; i++)
        array_insert(p, &i);
    printdata(p);
    return 0;
}
```