Lab05: Inteiros com e sem sinal

1. Analisando o programa abaixo, sem executá-lo, fazendo as contas para determinar o que será impresso por ele. Só depois de determinar todos os valores, execute o programa e verifique se você acertou.

```
#include <stdio.h>

void dump(void *p, int n) {
  unsigned char *p1 = p;
  while (n--) {
    printf("%p - 0x%02X\n", p1, *p1);
    p1++;
  }
}

int main(void) {
  char c = 150;
  short s = -3;
  int i = -151;
  printf("dump de c: \n");
  dump(&c, sizeof(c));
  printf("dump de s: \n");
  dump(&s, sizeof(s));
  printf("dump de i: \n");
  dump(&i, sizeof(i));
  return 0;
}
```

2. Mais uma vez, analise o programa abaixo e faça as contas para determinar o que será mostrado. Depois execute o programa para conferir.

```
#include <stdio.h>

void dump(void *p, int n) {
  unsigned char *p1 = p;
  while (n--) {
    printf("%p - 0x%02X\n", p1, *p1);
    p1++;
  }
}

int main(void) {
  short l = -32765;
  unsigned short k = 32771;
  printf("l = %d, k = %u\n", l, k);
  printf("dump de l:\n");
  dump(&l, sizeof(l));
  printf("dump de k:\n");
  dump(&k, sizeof(k));
  return 0;
}
```

3. Imagine que, para economizar espaço de armazenamento, utilizemos uma estrutura de dados que "empacota" quatro valores inteiros com sinal, de 1 byte cada, em um inteiro *unsigned* de 32 bits. Os bytes dentro desse inteiro são numerados, da direita para a esquerda, de 0 (byte menos significativo) a 3 (byte mais significativo), conforme mostrado a seguir:

Bits: 31 – 24	Bits: 23 – 16	Bits: 15 – 8	Bits: 7 – 0
Byte 3	Byte 2	Byte 1	Byte 0

Sua tarefa é implementar uma função com o protótipo abaixo:

```
typedef unsigned packed_t;

/*
 * Extrai o byte indicado e retorna valor inteiro
 * correspondente (32 bits) com sinal.
 */
int xbyte (packed_t word, int bytenum);
```

Use seus conhecimentos de aritmética de complemento para 2 e operações de manipulação de bits para implementar essa função. Use o *template* abaixo para implementar e testar a função:

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>

typedef unsigned packed_t;

int string2num (char *s, int base) {
  int a = 0;
  for (; *s; s++) {
    if(isdigit(*s))
      a = a*base + (*s - '0');
    else if((*s >= 'A') && (*s < (base-10+'A')))
      a = a*base + ((*s - 'A') + 10);
    else if((*s >= 'a') && (*s < (base-10+'a')))
      a = a*base + ((*s - 'a') + 10);
    else {
      printf("[ERRO] Número inválido!\n");
      exit(1);
    }
  }
  return a;
}

int xbyte (packed_t word, int bytenum) {
  /* implementar aqui */
  return 1;
}

int main (int argc, char **argv) {
  int x;
  if (argc != 3) {
    printf ("Uso: %s <word (em hexadecimal)> <bytenum>\n", argv[0]);
    exit(1);
  }

  x = xbyte(string2num(argv[1], 16), atoi(argv[2]));
  printf ("0x%08x  %d\n", x, x);
  return 0;
}
```

Para entender como deve funcionar o programa depois que você implementar a função xbyte, veja os exemplos abaixo:

```
$ ./xbyte 01abcd02 0
00000002  2

$ ./xbyte 11a032b5 0
ffffffb5  -75

$ ./xbyte 11a032b5 1
00000032  50

$ ./xbyte zzzz 1
[ERRO] Número inválido!
```

```
$ ./xbyte 11a032b5 2
ffffffa0  -96

$ ./xbyte 11a032b5 3
00000011  17

$ ./xbyte abcd 3
00000000  0

$ ./xbyte abcd 1
ffffffab  -85
```

4. Execute os programas abaixo (separadamente) e explique os resultados. Pense na representação interna dos valores de “x” e “y”, e na declaração de seus tipos.
Dica: Experimente trocar o “%d” por “%u” na string passada para printf() e veja o que os programas imprimem.

prog-01.c

```
#include <stdio.h>

int main (void) {
  int x = 0xffffffff;
  int y = 2;
  printf("x = %d, y = %d\n", x, y);
  printf("x é menor do que y? %s\n", (x<y) ? "sim":"nao");
  return 0;
}
```

prog-02.c

```
#include <stdio.h>

int main (void) {
  unsigned int x = 0xffffffff;
  unsigned int y = 2;
  printf("x = %d, y = %d\n", x, y);
  printf("x é menor do que y? %s\n", (x<y) ? "sim":"nao");
  return 0;
}
```

4.1. Agora execute o programa abaixo e explique o resultado comparando com os casos dos programas do exercício anterior:

prog-03.c

```
#include <stdio.h>

int main (void) {
  int x = 0xffffffff;
  unsigned int y = 2;
  printf("x = %d, y = %d\n", x, y);
  printf("x é menor do que y? %s\n", (x<y) ? "sim":"nao");
  return 0;
}
```

5. Em C, o programador pode atribuir um *signed char*, que ocupa 8 bits, a um *unsigned int*, que ocupa 32 bits:

```
signed  char sc = -1;
unsigned int ui = sc;
```

Ao atribuir um valor a “ui”, o que o código gerado tem que fazer?

Observe que há duas mudanças envolvidas, o tamanho da representação e o tipo (o tipo, nesse caso, é importante para saber como fazer a mudança de tamanho).

Faça a conversão na mão e diga como ficará a representação interna de “ui”. Em seguida, escreva um programa em C para conferir sua resposta. (Escreva as linhas acima e depois use a função *dump*() do exercício (1) para mostrar “ui”).