**LAB2 SB Anotacoes**

Um inteiro tem 4 bytes, a reposta do primeiro programa resulta em:

0x7ffc3c336db4 - 10

0x7ffc3c336db5 - 27

0x7ffc3c336db6 - 00

0x7ffc3c336db7 - 00

e o numero i = 10.000

isso signfica que o numero 10.000 é representado em hexademical por 0x2710, e como a arquitetura do computador é little endian, ele será reprensetado pelo byte menos signficativo para o mais, assim ficando na ordem mostrada. Como o numero 0x2710 só tem 2 bytes, porem é um inteiro logo sendo necessario "completar" esses bytes "sobrando" com 00 - os numeros antes 0x7ffc3c336db4 , .... , são os endereços de memória onde esses bytes estão sendo armazenados.

Quando muda o i para 250, vemos que o é:

0x7ffc2fa29ec4 - fa

0x7ffc2fa29ec5 - 00

0x7ffc2fa29ec6 - 00

0x7ffc2fa29ec7 - 00

Assim podemos ver que o numero tem 1 byte e teve que ser completado com 00. E que 250 pode ser representado com FA em hexadecimal

Para transformar um numero em hexadecimal basta dividir por 16 até o quociente ser 0, e pegar todos os restos do ultimo valor obtido ao primeiro

Substituindo o tipo de i por um long, vemos que é a mesmo resultado porem um long possui 8 bytes inves de 4, logo o output fica assim:

0x7ffd6eb9f780 - 10

0x7ffd6eb9f781 - 27

0x7ffd6eb9f782 - 00

0x7ffd6eb9f783 - 00

0x7ffd6eb9f784 - 00

0x7ffd6eb9f785 - 00

0x7ffd6eb9f786 - 00

0x7ffd6eb9f787 - 00

Já com o uso do short é usado 2 bytes:

0x7fff2ba04146 - 10

0x7fff2ba04147 - 27

Substituindo o tipo de i por um char e testando com 97 e 'a' vemos que ambos os casos o output é de 61, sendo o numero 97 representado em hexadecimal, pois 'a' é 97 na tabelas ASCII

E como um char só tem 1 byte:

0x7ffd33040217 - 61

Atualizando a main com o novo codigo

int main (void) {

char p[] = "7509";

dump(p, sizeof(p));

return 0;

}

E substituindo por "A" o p[], vemos que o numero encontrado é 41 que é 65 em hexadecimal, sendo 65 o valor da tabela ASCII do "A"

Sendo a mesma coisa para outros exemplos, como seus respectivos numeros da tabela ASCII em hexadecimal

Ex 2:

A função string2num percorre a string de entrada caractere por caractere e converte essa string, composta de dígitos, em um número inteiro correspondente.

O valor de (\*s - '0') é o valor numérico correspondente ao caractere atual na string. Por exemplo, se \*s é '1', (\*s - '0') será 1, etc...

O valor de a vai sendo atualizado em cada iteração, multiplicando o valor anterior de a por 10 e somando o valor de (\*s - '0').

Modificando a funcao e testando com os valores dados, temos:

int string2num(char \*s, int base) {

int a = 0;

for (; \*s; s++) {

a = a \* base + (\*s - '0');

}

return a;

}

int main (void) {

printf("%d\n", string2num("777", 8));

printf("%d\n", string2num("777", 10));

return 0;

}

Output:

511

777

Modificando novamente, temos:

int string2num(char \*s, int base) {

int a = 0;

int digit;

for (; \*s; s++) {

if (isdigit(\*s)) {

digit = \*s - '0';

} else if (isalpha(\*s)) {

digit = tolower(\*s) - 'a' + 10;

} else {

printf("Invalid character '%c' for base %d\n", \*s, base);

return -1;

}

if (digit >= base) {

printf("Invalid character '%c' for base %d\n", \*s, base);

return -1;

}

a = a \* base + digit;

}

return a;

}

int main(void) {

printf("%d\n", string2num("1a", 16));

printf("%d\n", string2num("a09b", 16));

printf("%d\n", string2num("z09b", 36));

return 0;

}

Output:

26

41115

1633295