Aula Prática
Operações Bit a Bit

Instruções para Submissão

Na aula prática de hoje, você terá que elaborar programas para resolver problemas diversos, conforme descrito abaixo. Cada uma das soluções deverá ser implementada em seu próprio arquivo com extensão `.c`. Por exemplo, a solução para o problema 1 deverá ser implementada em um arquivo chamado `problema1.c`, a solução para o problema 2 deverá ser implementada no arquivo `problema2.c` e assim por diante. Para os problemas que solicitarem a criação de um módulo, você deverá fornecer dois arquivos: um arquivo contendo a declaração das funções (arquivo com extensão `.h`), e um arquivo contendo a definição das funções (com extensão `.c`). Finalmente, submeta cada um dos arquivos pelo Moodle.

Problema 1 [problema1.h e problema1.c]

Escreva uma função chamada `fast_pow_2` que recebe como **parâmetro** um número (n) (tipo `int`) e **retorna** o valor de 2^n (tipo de retorno `unsigned long long`). Por exemplo, se o valor do expoente for 7, a função deve retornar o valor de 2^7. Neste exercício, você deve usar a operação de deslocamento de bits (<<), ou seja, não utilize a função `pow`.

Dica: O tipo `unsigned long long` é um inteiro sem sinal de 64 bits. Você pode usá-lo como um `unsigned int`, mas com um alcance muito maior. Diferente do tipo `long`, que nem sempre utiliza 64 bits, o tipo `long long` garante que serão utilizados pelo menos 64 bits na representação do número inteiro.

Protótipo da função:

```
unsigned long long fast_pow_2(int);
```

Observação 1: O cabeçalho da função deve ser exatamente como especificado acima (tipo de retorno, nome da função, tipos e quantidade de parâmetros).

Observação 2: Você terá que implementar um módulo com a função descrita acima. Logo, você deverá submeter dois arquivos: um arquivo contendo a declaração das funções (arquivo `problema1.h`), e um arquivo contendo a definição das funções (arquivo `problema1.c`)

Observação 3: Você não precisa realizar a entrada e saída de dados (não precisa usar as funções `scanf()` e `printf()`).

O programa que irá testar o seu módulo no Moodle é apresentado abaixo:

```c
#include "problema1.h"
#include <stdio.h>

int main() {
    int expoente;

    printf("Digite o expoente: \n");
    scanf("%d", &expoente);

    printf("%llu \n", fast_pow_2(expoente));

    return 0;
}
```

Exemplo de execução do programa:

Digite o expoente:
10
1024

Problema 2 [problema2.h e problema2.c]

Escreva uma função chamada `par_ou_impar` que recebe como **parâmetro** um número (n) (tipo `unsigned int`) e **retorna** 1 se o número for par, ou 0 caso contrário (tipo de retorno `int`). Tente solucionar o problema utilizando o operador bit a bit AND (&), ou seja, a função não deve usar operadores condicionais, como `if, ?:, switch`, etc.

Protótipo da função:

```
int par_ou_impar(unsigned int);
```

Observação 1: O cabeçalho da função deve ser exatamente como especificado acima (tipo de retorno, nome da função, tipos e quantidade de parâmetros).

Observação 2: Você terá que implementar um módulo com a função descrita acima. Logo, você deverá submeter dois arquivos: um arquivo contendo a declaração das funções (arquivo `problema2.h`), e um arquivo contendo a definição das funções (arquivo `problema2.c`)

Observação 3: Você não precisa realizar a entrada e saída de dados (não precisa usar as funções `scanf()` e `printf()`).

O programa que irá testar o seu módulo no Moodle é apresentado abaixo:

```
#include "problema2.h"
#include <stdio.h>

int main() {
    unsigned int x;

    printf("Digite o numero: \n");
    scanf("%d", &x);

    printf("%d \n", par_ou_impar(x));

    return 0;
}
```

Exemplo de execução do programa:

Digite o numero:
60
1

Problema 3 [problema3.h e problema3.c]

Escreva uma função chamada `verifica_bit` que recebe como **parâmetros** um número (n) (tipo `unsigned int`) e a posição de um bit (tipo `unsigned short`). Pode assumir que o valor da posição de bit sempre será um número entre 0 e 31. A função deve então **retornar** o valor inteiro 1 se o bit da posição informada for 1 no número (o bit está ligado), e 0 caso contrário (o bit está desligado) (tipo de retorno `int`). Por exemplo: Se o número for 42 (`101010` em binário) e a posição informada for 2, a função deve retornar 0. Por outro lado, se a posição informada for 3, a função deve retornar 1.

Protótipo da função:

```
int verifica_bit(unsigned int, unsigned short);
```

Observação 1: O cabeçalho da função deve ser exatamente como especificado acima (tipo de retorno, nome da função, tipos e quantidade de parâmetros).

Observação 2: Você terá que implementar um módulo com a função descrita acima. Logo, você deverá submeter dois arquivos: um arquivo contendo a declaração das funções (arquivo `problema3.h`),

e um arquivo contendo a definição das funções (arquivo `problema3.c`)

Observação 3: Você não precisa realizar a entrada e saída de dados (não precisa usar as funções `scanf()` e `printf()`).

O programa que irá testar o seu módulo no Moodle é apresentado abaixo:

```
#include "problema3.h"
#include <stdio.h>

int main() {
    unsigned int num;
    unsigned short pos;

    printf("Digite: num, pos \n");
    scanf("%u, %hu", &num, &pos);

    printf("%d \n", verifica_bit(num, pos));

    return 0;
}
```

Exemplo de execução do programa:

Digite: num, pos
170, 2
0

Problema 4 [problema4.h e problema4.c]

Escreva uma função chamada `ativa_bit` que recebe como **parâmetros** um número (n) (tipo `unsigned int`) e a posição de um bit (tipo `unsigned short`). Pode assumir que o valor da posição de bit sempre será um número entre 0 e 31. A função deve então `retornar` o novo número com o bit naquela posição ligado, ou seja, o bit deve ter valor 1 (tipo de retorno `unsigned int`). Por exemplo: Se o número for 42 (`101010` em binário) e a posição informada for 4, a função deve retornar 58 (`111010` em binário). Por outro lado, se a posição informada for 3, a função deve retornar o próprio 42.

Protótipo da função:

```
unsigned int ativa_bit(unsigned int, unsigned short);
```

Observação 1: O cabeçalho da função deve ser exatamente como especificado acima (tipo de retorno, nome da função, tipos e quantidade de parâmetros).

Observação 2: Você terá que implementar um módulo com a função descrita acima. Logo, você deverá submeter dois arquivos: um arquivo contendo a declaração das funções (arquivo `problema4.h`), e um arquivo contendo a definição das funções (arquivo `problema4.c`)

Observação 3: Você não precisa realizar a entrada e saída de dados (não precisa usar as funções `scanf()` e `printf()`).

O programa que irá testar o seu módulo no Moodle é apresentado abaixo:

```
#include "problema4.h"
#include <stdio.h>

int main() {
    unsigned int num;
    unsigned short pos;

    printf("Digite: num, pos \n");
    scanf("%u, %hu", &num, &pos);

    printf("%u \n", ativa_bit(num, pos));

    return 0;
}
```

Exemplo de execução do programa:

Digite: num, pos
8, 2
12

Problema 5 [problema5.h e problema5.c]

Escreva uma função chamada `desliga_bit` que recebe como **parâmetros** um número (n) (tipo `unsigned int`) e a posição de um bit (tipo `unsigned short`). Pode assumir que o valor da posição de bit sempre será um número entre 0 e 31. A função deve então `retornar` o novo número com o bit naquela posição desligado, ou seja, o bit deve ter valor 0 (tipo de retorno `unsigned int`). Por exemplo: Se o número for 42 (`101010` em binário) e a posição informada for 3, a função deve retornar 34 (`100010` em binário). Por outro lado, se a posição informada for 2, a função deve retornar o próprio 42.

Protótipo da função:

```
unsigned int desliga_bit(unsigned int, unsigned short);
```

Observação 1: O cabeçalho da função deve ser exatamente como especificado acima (tipo de retorno, nome da função, tipos e quantidade de parâmetros).

Observação 2: Você terá que implementar um módulo com a função descrita acima. Logo, você deverá submeter dois arquivos: um arquivo contendo a declaração das funções (arquivo `problema5.h`), e um arquivo contendo a definição das funções (arquivo `problema5.c`)

Observação 3: Você não precisa realizar a entrada e saída de dados (não precisa usar as funções `scanf()` e `printf()`).

O programa que irá testar o seu módulo no Moodle é apresentado abaixo:

```c
#include "problema5.h"
#include <stdio.h>

int main() {
    unsigned int num;
    unsigned short pos;

    printf("Digite: num, pos \n");
    scanf("%u, %hu", &num, &pos);

    printf("%u \n", desliga_bit(num, pos));

    return 0;
}
```

Exemplo de execução do programa:

Digite: num, pos
10, 1
8