Linguagens e Ambientes de Programação (2018/2019)

Prática 6

Primeiros exercícios sobre a linguagem C. Exercícios de 31 a 35.

Guião para usar a linguagem C na consola

1. Usar um editor de texto qualquer para escrever um ficheiro "test.c"
 2. Gravar
 3. `cc -o test test.c`
 4. `./test`
-
-

Guião para usar a linguagem C dentro do Eclipse

1. Ativar Eclipse, escrevendo "eclipse" num consola.
2. Window > Open Perspective... mudar perspetiva para C/C++
3. File > New > Project...
4. Selecionar "C Project"
5. Escrever "Test" e selecionar "Hello World ANSI C Project" e "Linux GCC"
6. Fazer "Finish" (ignorar o "Next")
7. File > New > Source File
8. Dar o nome "Test.c" ao ficheiro
9. Escrever o programa no ficheiro "Test.c"
10. Ctrl-S (save) para gravar
11. Carregar seta branca e verde da barra de tools para compilar e correr o programa.
12. Se o programa não correr logo, ir ao menu Project, forçar um build manual, e depois mandar o programa correr por duas vezes na seta verde. Da segunda vez, o programa já corre.

Instalação: Já está instalado nos laboratórios. Para instalar no seu computador pessoal, use o comando do Eclipse "Help>Install New Software...". Na caixa "Work with:" insira "http://download.eclipse.org/tools/cdt/releases/_versao_" e depois escolha apenas "CDT Main features".

- 31 - Considere a seguinte variável global `x` e as seguintes funções `f` e `g`:

```
int x;  
  
int f(int a) { x = a + 1; return x; }  
int g(int a) { x = a - 1; return x; }
```

Suponha que `f` e `g` são usadas num segmento de programa da seguinte forma:

```
...  
x = 10;  
s1 = f(x) + g(x);  
x = 10;  
s2 = g(x) + f(x);
```

Quais os valores que ficam em `s1` e `s2`, assumindo que os operandos nas expressões são avaliados da esquerda para a direita. Note que, em geral, os compiladores têm a liberdade de definir a ordem de avaliação ou seja a ordem de avaliação das sub-expressões está indefinida ao nível da linguagem. Isso recomenda que evitemos escrever código do género do que está contido neste exemplo.

- 32 - Analise o programa incompleto abaixo. No início, define-se um tipo `Byte` para representar bytes de memória. Também se define um tipo `Block` para representar sequências de bytes em memória. A função `blockZero` inicializa um bloco com zeros, e a função `blockPrint` mostra o conteúdo dum bloco.

Pedimos-lhe para completar as funções `blockFill_char_uint_double` e `blockPrint_char_uint_double`.

A função `blockFill_char_uint_double` guarda no início dum bloco três valores de diferentes tipos, sem qualquer alinhamento: um `char`, um `unsigned int` e um `double`.

A função `blockPrint_char_uint_double` extrai e imprime (um por linha) três valores do início dum bloco, com os tipos: `char`, um `unsigned int` e um `double`.

A função `main` já está escrita e serve para testar se você escreveu bem as funções anteriores.

```
#include <stdio.h>

#define BLOCK_SIZE 32

typedef unsigned char Byte;
typedef Byte Block[BLOCK_SIZE];

void blockZero(Block b) {
    int i;
    for( i = 0 ; i < BLOCK_SIZE ; i++ )
        b[i] = 0;
}

void blockPrint(Block b) {
    int i;
    for( i = 0 ; i < BLOCK_SIZE ; i++ )
        printf("%02x ", b[i] & 0xFF);
    printf("\n");
}

void blockFill_char_uint_double(Block b, char c, unsigned int i, double d) {
    Byte *pt = b;
    // fazer
}

void blockPrint_char_uint_double(Block b) {
    Byte *pt = b;
    // fazer
    printf("\n");
}

int main(void) {
    Block b;
    blockZero(b); blockPrint(b);
    blockFill_char_uint_double(b, 'a', ~0, 667.6e-22); blockPrint(b);
    blockPrint_char_uint_double(b);
    return 0;
}
```

- 33 - Escreva em C um programa introspectivo que olhe para a sua própria pilha de execução e escreva os dados necessários para poder responder às seguintes perguntas:
 - Em que sentido cresce a pilha de execução na implementação de C que está a usar.
 - Por que ordem são empilhados os argumentos das funções.

- Por que ordem são arrumadas as variáveis locais nos registos de activação.

Use o seguinte código como ponto de partida. Vai precisar de usar um apontador.

```
#include <stdio.h>
#include <stdint.h>

#if defined(__LP64__)
    typedef uint64_t Word;
#else
    typedef uint32_t Word;
#endif
typedef Word *Ptr;

void f(Word a1, Word a2, Word a3) {
    Word local1 = 55555;
    Word local2 = 66666;
    Word local3 = 77777;
    // ...
}

int main(void) {
    f(11111, 22222, 33333);
    return 0;
}
```

- 34 - Considere novamente o tipo soma Shape apresentado na teórica 11. Desta vez é também oferecido um construtor de pontos e um construtor de linhas.

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

typedef enum { LINE, CIRCLE, RECTANGLE } ShapeKind ;
typedef struct { double x, y; } Point;
typedef struct { Point p1, p2; } Line;
typedef struct { Point centre; int radius; } Circle;
typedef struct { Point top_left; double width, height; } Rectangle;

typedef struct { // Isto é um tipo SOMA, programado com a ajuda duma UNION
    ShapeKind kind;
    int color;
    union {
        Line line;
        Circle circle;
        Rectangle rectangle;
    } u;
} Shape;

Point point(double x, double y) // construtor de pontos
{
    Point p = {x, y};
    return p;
}

Shape line(Point p1, Point p2, int color) // construtor de linhas
{
    Line l = {p1, p2};
    Shape s = {LINE, color};
    s.u.line = l;
    return s;
}

int main(void)
{
    Shape c = circle(point(0,0), 1, 99);
    printf("%lf\n", area(c));
}
```

```

Shape t = triangle(point(0,0), point(5,10), point(10,5), 99);
printf("%lf\n", area(t));
return 0;
}

```

- Escreva os construtores que faltam.
- Escreva uma função chamada `area` para calcular a área de uma `Shape`.
- Faça todas as modificações necessárias para adicionar triângulos representados por três vértices. ([área do triângulo](#))

- 35 - Escreva em C um programa avaliador de expressões aritméticas inteiras onde podem ocorrer os operadores `+` e `*`. O programa lê uma linha completa da consola, avalia a expressão nela contida e escreve o resultado na consola. Quando testar, não introduza espaços em branco, pois o programa não está preparado para lidar com eles.

Para processar as fórmulas, escreva uma coleção de funções, seguindo a estrutura da gramática LL1 abaixo. Gramáticas LL1, vão ser estudadas em TC. Têm a particularidade de admitirem uma tradução simples à mão para código que implementa um reconhecedor da linguagem em causa. [Este exercício é um simples pretexto para programar em C algo de útil e interessante, num momento em que a matéria sobre esta linguagem apenas começou a ser dada nas teóricas.]

```

S -> E\n
E -> TX
X -> +TX | λ
T -> FY
Y -> *FY | λ
F -> I | (E)
I -> DZ
Z -> DZ | λ
D -> 0 | 1 | ... | 8 | 9

```

O programa já está 80% escrito. Estude o que já está feito, comparando com a gramática, e escreva as três funções que faltam: as funções `T`, `Y` e `F`.

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>

#define MAX_LINE    1000

char line[MAX_LINE];
int curr;

void error(char *mesg) {
    fprintf(stderr, "ERROR: %s\n", mesg);
    exit(1);
}

bool isDigit(char c) {
    return '0' <= c && c <= '9';
}

int D(void) {
    if( isDigit(line[curr]) ) {
        int val = line[curr] - '0';
        curr++;
        return val;
    }
    else error("Expected digit");
}

```

```

int Z(int val) {
    if( isDigit(line[curr]) )
        return Z(val * 10 + D());
    else
        return val;
}

int I(void) {
    return Z(D());
}

// Faltam T, Y, F

int X(int val) {
    if( line[curr] == '+' ) {
        curr++;
        return X(val + T());
    }
    else
        return val;
}

int E(void) {
    return X(T());
}

int S(void) {
    int val = E();
    if( line[curr] != '\0' )
        error("Texto a mais depois do final da expressão");
    return val;
}

int main(void) {
    for(;;) {
        printf("> ");
        fflush(stdout);
        fgets(line, MAX_LINE, stdin); // faça "man fgets" para aprender...
        line[strlen(line)-1] = '\0';
        curr = 0;
        printf("%s = %d\n", line, S());
    }
}

```

- 36 - Enriqueça o programa anterior com 26 variáveis mutáveis (use os nomes 'a'..'z') e uma operação de atribuição denotada "=". As variáveis podem receber valores de expressões e as expressões podem conter variáveis. Quando o programa arranca, todas as variáveis começam com o valor zero.

```

S -> E\n | =VE\n                // =VE\n adicionado
E -> TX
X -> +TX | λ
T -> FY
Y -> *FY | λ
F -> I | V | (E)                // V adicionado
I -> DZ
Z -> DZ | λ
D -> 0 | 1 | ... | 8 | 9
V -> a | b | ... | y | z        // linha adicionada

```