**Requisitos:**

**Especificação da linguagem Fluxo-Gráfica**

Pretende-se a criação de uma ferramenta que permita o desenvolvimento/aprendizagem de algoritmos através de fluxogramas. Um fluxograma é um conjunto de símbolos gráficos, que contêm texto e são unidos por linhas de fluxo. Para este tópico os requisitos serão identificados por “FLUX.xx.xx”.

**FLUX.00** – Requisitos globais da linguagem.

**FLUX.01** - Inicio/Fim.

**FLUX.02** - União.

**FLUX.03** - Leitura.

**FLUX.04** – Processo/Funções.

**FLUX.05** – Decisão.

**FLUX.06** – Escrita.

**FLUX.07** – Return.

**FLUX.08** – Ligações.

**FLUX.00.01** – As formas devem ser distintas entre si

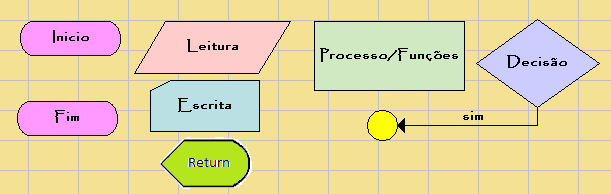


Figura 1 – Exemplo das formas

**FLUX.00.02** – Todas as formas devem ser manipuláveis graficamente, recorrendo ao uso de pontos extensíveis, arrasto com o rato.

**FLUX.00.02.01** – O duplo clique de rato serve para inicializar a introdução de dados.



Figura 2 – Pontos de extensão

**FLUX.00.03** – As seguintes formas devem suportar multi-texto:

* Leitura
* Processo/Funções
* Escrita

**FLUX.00.04** – As linguagens de programação diferem entre si e por isso os símbolos reservados são diferentes entre as mesmas, um utilizador deve poder escolher os símbolos com que está mais familiarizado.

**FLUX.00.04.01** – Os símbolos padrão são aqueles utilizados no Portugol.

**FLUX.01.01** - O início e o fim são definidos pela mesma forma:

Figura 3 – forma para início e fim.

**FLUX.01.02** - Deve ser possível atribuir apenas uma ligação de entrada.

**FLUX.01.03** - Deve ser possível atribuir apenas uma ligação de saída.

**FLUX.01.04** – O texto define o início ou fim e por isso deve estar protegido e não pode ser alterado.

**FLUX.01.04.01** – O duplo clique de rato altera o sentido da forma.

**FLUX.02.01** - A união reúne o fluxo do programa.

Figura 4 - União

**FLUX.02.02** - Deve ser possível atribuir uma ou mais ligações de entrada.

**FLUX.02.03** - Deve ser possível atribuir apenas uma ligação de saída.

**FLUX.03.01** – A leitura define a declaração de variáveis.

 FLUX.03.01.01 – Podem ser definidas várias variáveis separadas pelo símbolo “,”.

Figura 5 - Leitura

**FLUX.03.02** - O tipo de variáveis, vão ser deduzidas pelo código.

**FLUX.03.03** - As variáveis podem ser Inteiros, Reais, Booleans, Strings e Char.

**FLUX.03.04** - O valor da inicialização da variável tem de ser compatível com o tipo de variável definido.

**FLUX.03.05** - O valor da inicialização pode ser omitido, logo ela é inicializada por defeito.

**FLUX.03.06** - É possível omitir a palavra variável.

**FLUX.03.07** - Se a variável contém uma String, a mesma, começa por uma letra ou caracter e não pode ser nenhuma palavra reservada da linguagem.

**FLUX.03.08** – Podem ser definidos arrays vazios seguindo a forma:

“Nome\_array[tamanho\_array]”

**FLUX.03.09** - A forma da declaração da variável permite uma entrada de informação e uma saída de informação;

**FLUX.04.01** - O mesmo símbolo representa expressões e/ou funções

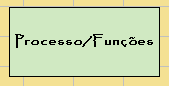


Figura 6 – Processo/Funções

**FLUX.04.02** - No mesmo bloco podem ser executadas várias tarefas



Figura 7 – Esquema de introdução de dados

**FLUX.04.03** - A forma tem suporte para multi-texto

**FLUX.04.04** - Pode ter vários pontos de entrada mas apenas tem uma entrada válida. Pode ter uma ou mais saídas



Figura 8 – Entradas e Saídas

**FLUX.04.05** - As saídas possíveis são, a decisão, o return e a escrita

**FLUX.04.06** - Deve ser possível obter resultados individuais



Figura 9 – Saída de dados separadamente

**FLUX.04.07** - Devem estar reservados símbolos para operações aritméticas

**FLUX.04.08** - Soma ' + '

Subtração ' - '

Divisão ' / '

Multiplicação ' \* '

Percentagem ' % '

Potência ' ^ '

**FLUX.04.09** - As funções devem ser definidas com o nome da função seguido dos seus parâmetros entre parêntesis.

**FLUX.04.10** - Uma função pode ter um ou mais parâmetros e os mesmos devem estar separados por vírgulas, exemplo: cos(x).

**FLUX.04.11** - Deve ser possível a definição de arrays e a atribuição de valores a arrays vazios.



Figura 10 – Exemplo de inicialização de um array

**FLUX.04.11.01** - O sistema deve permitir que a inicialização do array seja feita com dados compreendidos entre chavetas e separados por “,”.

**FLUX.04.11.02** – Se as posições não estiverem todas ocupadas, são inicializadas a “0”.

**FLUX.05.01** – Decisão permite a criação de ciclos, comparações e decisões.

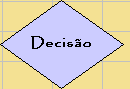


Figura 11 – Decisão

**FLUX.05.02** - Deve ser possível atribuir apenas uma conexão de entrada.

**FLUX.05.03** - Deve ser possível atribuir, no mínimo, uma conexão de saída e, no máximo, dois conexões de saída.

**FLUX.05.04** - Ao selecionar a primeira atribuição de saída, deve-lhe ser atribuída por defeito a condição “Sim”.

**FLUX.05.05** - Ao selecionar a segunda atribuição de saída, deve-lhe ser atribuída por defeito a condição “Não”.

Não

Sim

Figura 12 – Relação conectores/decisão

**FLUX.05.05** - Dentro da forma deve ser permitida a escrita de operadores relacionais e lógicos

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Operadores Relacionais: | < | > | = | != | <= | >= |

|  |  |  |  |
| --- | --- | --- | --- |
| Operadores Lógicos: | E | OU | NAO |
| Operadores Lógicos(Alternativo): | && | || | ! |

**FLUX.05.06** - Um ciclo deve ser representado na seguinte forma:

Figura 13 - Ciclo

**FLUX.06.01** – A escrita permite o retorno de dados/resultados de operações simples.

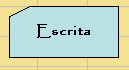


Figura 14 - Escrita

**FLUX.06.02** - Possui uma só entrada e saída de conetores

**FLUX.06.03** - É responsável pela escrita do valor de variáveis, funções, texto ou expressões

**FLUX.06.04** - Permite a escrita de um ou mais elementos, dos quais, variáveis, funções, textos ou expressões, separadas por uma vírgula “,”

**FLUX.06.05** – A escrita processa-se pela ordem indicada no “fluxograma” ou no código (Ex: a,b,c ; imprime a, seguido de b e for fim c)

**FLUX.06.06** - Para a escrita variáveis é escrito o valor correspondente dessa variável

**FLUX.06.07** - Para escrita de texto, é mostrado o texto colocado entre aspas(“”) (Ex: “123” imprime 123)

**FLUX.06.08** - Para o caso da escrita de uma função esta é caracterizada pelo valor de retorno, escrevendo esse valor

**FLUX.06.09** - Esta forma suporta caracteres reservados, tais como o “\n” para mudança de linha (enter), “\t” para a tabulação e \”para as aspas(“)

**FLUX.07.01** – o return represenal o resultado de uma função



Figura 15 - return

**FLUX.07.02** - Dentro do símbolo de retorno é permitido retornar:

* + Vários tipos normais de variáveis(char, int, reais, bolean, strings)
  + Números
  + Expressões matemáticas (cos, sin, tan, etc)
  + Operações matemáticas

**FLUX.07.03** - Deve ser possível atribuir apenas uma conexão de entrada e uma saída

**FLUX.07.04** - Não é obrigatório a inclusão de retorno a não ser que a função o especifique (não ser do tipo void ou conter parâmetros)

c <- Soma(a, b)

Descrição: Description: return.png

c

Figura 16 – Resultado de uma função

**FLUX.08.01** – Os conectores ligam as formas entre si, são unidirecionais e podem estabelecer condições de saída para a forma de decisão como o especificado em “**FLUX.05”.**

Figura 17 – Exemplo de conector

**Especificação da forma FOR**

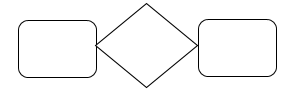
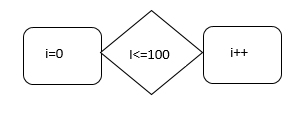
**FLUX.10.00 –** Esta forma, representa na sua essência o ciclo FOR, e tem o seguinte formato:

Figura 18

**FLUX.10.01 –** Tal como o próprio ciclo FOR, esta forma está dividida em 3 secções: definição(i=0), condição(i<=100) e o incremento(i++).

Exemplo:

Figura 19

**FLUX.10.02 -** A forma FOR dispõe de duas conexões de entrada: uma proveniente de outras formas e outra da instrução que executa. As conexões de entrada apenas podem ser encaixadas na secção de definição.

A forma suporta tantas condições de entrada quanto o número de instruções.

**FLUX.10.03 -** A forma FOR dispõe de duas conexões de saída: uma que saí do ciclo para outras formas e outra que indica que instruções o ciclo vai executar.

As conexões de saída apenas podem ser encaixadas na secção de incremento. A forma suporta tantas condições de saída quanto o número de instruções.

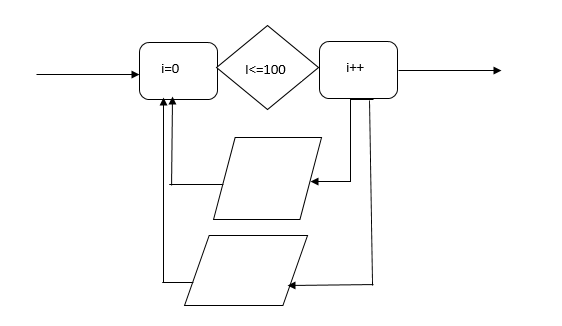


Figura 20

**FLUX.10.04** - Dentro da secção condição deve ser permitida a escrita de operadores relacionais e lógicos

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Operadores Relacionais: | < | > | = | != | <= | >= |

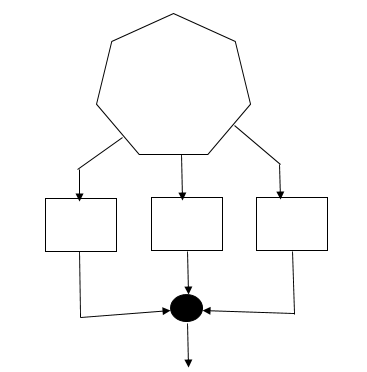
|  |  |  |  |
| --- | --- | --- | --- |
| Operadores Lógicos: | E | OU | NAO |
| Operadores Lógicos(Alternativo): | && | || | ! |

**Especificação da forma Switch Case**

Figura 22 - SWITCH CASE

**FLUX.09.00** – A forma Switch é, essencialmente, um conjunto de ciclos *if’s* encadeados

**FLUX.09.01** – Tem somente uma conexão de entrada.

**FLUX.09.02** – Tem obrigatoriamente uma conexão de saída, proveniente da forma União, sendo que entre o Switch e a União pode ter várias conexões tanto de entrada como de saída.

Condição

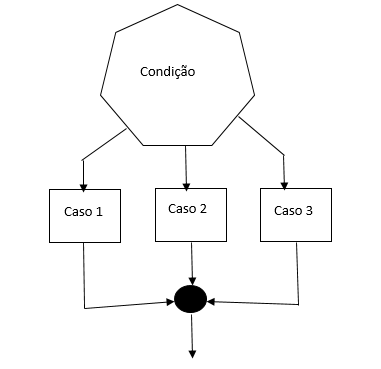
Figura 23 -

Caso 3

Caso 2

Caso 1

**FLUX.09.03** – Cada conexão entre a condição e o caso, deve conter uma label editável que especifica de que o caso se trata (Ex: 1,2,3;Janeiro,Fevereiro,Março)



Janeiro

Fevereiro

Março

Figura 24 -

**FLUX.09.04** - Dentro da forma deve ser permitida a escrita de operadores relacionais e lógicos

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Operadores Relacionais: | < | > | = | != | <= | >= |

|  |  |  |  |
| --- | --- | --- | --- |
| Operadores Lógicos: | E | OU | NAO |
| Operadores Lógicos(Alternativo): | && | || | ! |

**FLUX.09.05** – A condição deve ser especificada na forma Switch mas o resultado deve ser especificado em cada caso.

