CPD - INFORMA

UNIVERSIDADE DE SAO PAULO - ESCOLA DE ENGENHARIA DE SAO CARLOS

ANO 2 NUMERO 9 OUTUBRO DE 1989

S U M A R I O

NOTICIAS..... 01

INFORMES TECNICOS..... 01

ESTATISTICAS 13

D E S T A Q U E S

.PROGRAMACAO COM CORRECAO E CONFIABILIDADE

NOTICIAS

* REDEUSP

Realizou-se no dia 24/11/89 junto ao CCE/USP, um seminário sobre utilização da REDEUSP. Participaram os funcionários João Roberto de Almeida e Luiz Carlos Dotta do CPD/EESC/USP.

INFORMES TÉCNICOS

* PROGRAMAÇAO COM CORREÇAO E CONFIABILIDADE : Eliminando Efeitos Prejudiciais

Edson Walmir Cazarini
Analista de Sistemas do CPD

INTRODUÇAO

Na primeira parte deste artigo foram enfatizados os fatores que prejudicam a comprovação da correção de programas (uso de GO TO, os efeitos colaterais e o uso de sinônimos) e citados alguns recursos para obter confiabilidade, como por exemplo a estruturação de programas, a modularidade de programas e o exame do código dos programas (walk through).

Nesta segunda parte serão enfatizadas formas de eliminar os efeitos prejudiciais.

A utilização do comando GO TO na programação de algoritmos, comando bastante utilizado antes do aparecimento de metodologias e de técnicas de programação (quando programar era uma arte), é um dos principais fatores responsáveis por erros em programas. Seu uso quebra o seqüenciamento na leitura de um programa, violando sua estrutura e dificultando o seu entendimento. As linguagens mais modernas simplesmente eliminaram esse comando, não colocando essa

possibilidade à disposição do programador. Modernas técnicas de programação, por sua vez, mostram que este comando é totalmente dispensável. Atualmente o comando GO TO só aparece na forma de escape de estruturas de repetição (geralmente recebe o nome de exit).

Dois outros fatores que dificultam a produção de programas corretos são: os efeitos colaterais; resultado de modificações provocadas no ambiente externo de uma unidade de programa, e os sinônimos, onde duas variáveis se referenciam ao mesmo objeto durante a execução de uma unidade de programa. Esses dois fatores dificultam a leitura do programa por não serem facilmente visíveis quando os comandos do programa são examinados isoladamente.

FORMAS DE CONTROLE DE EFEITOS COLATERAIS

Uma forma de controlar os efeitos colaterais é através da utilização de passagem de parâmetros, como meio de comunicação entre unidades de programa. A passagem de parâmetros obriga o programador a declarar as variáveis-parâmetros no local da chamada, especificando assim as variáveis que potencialmente sofrerão um efeito colateral, e que portanto, deverão ser controladas.

A passagem de parâmetros é utilizada para comunicação entre unidades de programa. Através de parâmetros é permitida a transferência de diferentes objetos a cada chamada do subprograma ou procedimento; este é o meio mais adequado para tal finalidade, considerando-se os aspectos de legibilidade e manutenção dos programas. Duas classes de objetos podem ser passados como parâmetros: dados e subprogramas.

Passagem de dados como parâmetro: Os parâmetros são declarados em dois instantes diferentes: os parâmetros formais - são declarados quando da definição do subprograma (procedimento, função ou sub-rotina) e são válidos no interior do subprograma. Os parâmetros reais são declarados no programa chamador (em geral o programa principal) e são utilizados na hora e no local da chamada do subprograma. No momento da

chamada eles são passados para o subprograma através dos parâmetros formais, que devem ser compatíveis em relação ao tipo.

Existem três convenções distintas para passagem de dados através de parâmetros a um subprograma: por referência, por cópia e por nome. É importante conhecer quais dessas convenções são adotadas pela linguagem em uso, uma vez que cada uma possui sua semântica própria.

Passagem de dados através de parâmetros por referência: A unidade chamadora passa à unidade chamada o endereço do parâmetro real, possibilitando a alteração direta do valor do parâmetro real pelo subprograma chamado.

Exemplo, em FORTRAN:

```
      SUBROUTINE SOMAVT(NE, A, B, C, IR)
      DIMENSION A(10),B(10),C(10)
      IF NE .LE. 10  THEN
         IR = 0
         DO 100 I = 1, NE
         C(I) = A(I) + B(I)
100      CONTINUE
      ELSE
         IR = 1
      ENDIF
```

Na chamada, no programa principal:

```
      CALL SOMAVT(N, X, Y, Z, IOK)
      ...
      CALL SOMAVT(M, D, E, F, IOK)
      ...
```

Parâmetros FORMAIS: NE, A, B, C, IR
Parâmetros REAIS : N, M, X, Y, Z, D, E, F e IOK

Sofrerão efeitos colaterais, as variáveis Z, F e IOK, uma vez que foram alteradas através dos parâmetros formais C e IR.

Passagem de dados através de parâmetros por cópia: É feita uma cópia do valor do parâmetro real para o parâmetro formal, no momento da chamada. Os parâmetros formais são variáveis locais. Esse tipo de passagem de parâmetro tem a vantagem de proteger os parâmetros reais de modificações acidentais (efeitos colaterais indesejáveis). Existem três maneiras de passar parâmetro por cópia: por valor, por resultado e por valor-resultado.

Passagem de dados por cópia de valor: Os valores dos parâmetros reais são utilizados para inicializar os valores dos parâmetros formais correspondentes, que agem como variáveis locais na unidade chamada. Não é permitido qualquer fluxo de dados no sentido de retorno à unidade chamadora através dos parâmetros passados.

Passagem de dados por cópia de resultado: Os valores dos parâmetros formais são copiados para os parâmetros reais no término do processamento da unidade chamada. Não será possível o fluxo de informações no sentido da unidade chamadora para a unidade chamada.

Passagem de dados por cópia de valor-resultado: Corresponde à fusão dos dois casos anteriores; os parâmetros formais são variáveis locais que são inicializadas com os valores dos parâmetros reais e no término da execução da unidade chamada, os valores finais dos parâmetros formais são copiados para os parâmetros reais.

Exemplo em Pascal:

```
Procedure SomaVetores(NE: integer;
          A,B: array[1..10] of real;
          var C: array[1..10] of real;
          var IR: integer)
var
     i : integer;

begin
  If NE <= 10  then
     begin
     IR := 0;
     For i := 1 to NE  DO
        C[i] := A[i] + B[i]
     end
   Else IR := 1;
end;
```

Na chamada, no programa principal:

```
SomaVetores(N, X, Y, Z, IOK)
...
SomaVetores(M, D, E, F, IOK)
...
```

Parâmetros FORMAIS:- NE, A, B, (por cópia)
C, IR (por referência)
Parâmetros REAIS :- N, M, X, Y, Z, D, E, F e IOK

Sofrerão efeitos colaterais, as variáveis Z, F e IOK, uma vez que foram alterados através dos parâmetros formais C e IR. A variável "i", é uma variável local, tendo seu escopo, definido apenas no interior da procedure. Apenas os parâmetros passados por referência (prefixo VAR), poderão provocar efeito colateral. Os demais parâmetros, passados por cópia de valor, serão considerados variáveis locais.

Passagem de dados através de parâmetros por nome: Como na passagem de parâmetro por referência, os parâmetros formais NÃO são variáveis locais, pertencem ao ambiente da unidade chamadora. Entretanto, a associação entre os parâmetros reais e formais não é feita no momento da inicialização da unidade chamada,

mas sim cada vez que o parâmetro formal é usado. Como conseqüência, cada utilização de um parâmetro formal pode referenciar uma localização diferente. Se os parâmetros reais forem variáveis escalares, a passagem de parâmetro por nome será equivalente à passagem de parâmetro por referência.

Apesar de muito poderosa, esse tipo pode levar a erros em tarefas bastante simples, se não for utilizada com cuidado. A procedure TROCA, em ALGOL, é um exemplo clássico:

```
procedure TROCA (A,B);
     real A,B;
     begin real TEMP;
          TEMP := A;
          A := B;
          B := TEMP;
     end;
end;
```

Uma chamada TROCA(I,C[I]) produzirá um resultado diferente do desejado. Se antes da chamada tivermos I=2, C[2]=3, C[3]=4, após a chamada teremos : I=3, C[2]=3 e C[4]=2.

Quando a linguagem oferece mais que um tipo de passagem de parâmetro a escolha entre qual usar pode ser orientada por considerações sobre efeitos colaterais e eficiência de utilização. Passagem por referência possui a possibilidade de efeitos colaterais e pode ser ineficiente se os parâmetros formais são intensamente utilizados no subprograma, uma vez que cada acesso requer um endereçamento indireto. Por outro lado, se os objetos passados são grandes e pouco utilizados no subprograma, a passagem por cópia torna-se ineficiente em termos de memória e tempo para as cópias.

No FORTRAN 77 : A comunicação entre as unidades pode ser feita através da passagem de parâmetros por referência. A linguagem NÃO faz verificações de compatibilidade entre parâmetros reais e formais, ficando esse controle sob a responsabilidade do programador. O uso de variáveis globais é feito através da área de dados comum (comando COMMON).

No PASCAL : A comunicação é feita pela passagem de parâmetros e variáveis globais. A passagem de dados através de parâmetros pode ser feita por cópia (passagem por valor) ou por referência (na sintaxe, utiliza o prefixo VAR). A linguagem faz a verificação de compatibilidade entre os parâmetros formais e reais.

Passagem de Subprogramas como Parâmetros: Embora útil, e às vezes indispensável, a utilização de subprogramas como parâmetros podem facilmente levar à criação de programas obscuros, com pouca legibilidade. O programa passado como parâmetro, por exemplo, uma subrotina que calcula a integral de uma função, pode acessar variáveis não locais, sendo necessário a obtenção de informações sobre o ambiente não local pertencente a esse subprograma. A bem da qualidade, essa opção deve ser evitada.

No FORTRAN 77 : É permitido a passagem de parâmetro do tipo subprograma.

No PASCAL : É permitido a passagem de parâmetro do tipo procedures e funções.

FORMAS DE CONTROLAR OS SINONIMOS

Com relação aos sinônimos, as linguagens oferecem pouca contribuição. A solução está na disciplina do programador.

Os identificadores que serão utilizados na definição dos nomes de dados e procedimentos, devem identificar bem as entidades (classe homogênea de objetos que podem ser identificados de forma distinta em função de suas caracteristicas ou propriedades comuns) por eles representadas. Os nomes deverão ser formados levando-se em conta a documentação do algoritmo, evitando-se codificações excessivas.

A única justificativa aceitável para a utilização de sinônimos é que se pode com isso economizar memória, às vezes vital para a aplicação.

OUTRAS FORMAS DE CONTROLE DA CORREÇAO E CONFIABILIDADE: UTILIZAÇAO DE TIPOS DE DADOS; TRATAMENTO DE ERROS; VERIFICAÇAO E TESTE

A possibilidade que as linguagens atuais oferecem na construção de tipos de dados, também deve ser explorada pelo programador. Conceituação de tipos de dados:

Um objeto de dado é dito elementar se ele contém somente um único valor de dado, sendo sempre manipulado como uma unidade. Essa classe de objetos adicionados às operações destinadas à sua definição e manipulação é chamada de tipo de dados elementares. Exemplo: REAL (classe de números, que contém um ponto decimal, juntamente com as operações de adição, subtração, multiplicação e divisão).

Um objeto de dado composto por um agregado de outros objetos de dados é chamado de estrutura de dados. Os objetos de dados que ele contém são chamados de componentes. Um componente pode ser elementar ou ainda uma outra estrutura de dados. Uma classe desses objetos associados às suas operações é chamada de tipo de dados estruturados. Exemplo: ARRAY[0..100] OF INTEGER (classe formada por um agregado de 101 (0..100) números sem o

ponto decimal, juntamente com as operações de acesso, incremento/decremento do índice).

Um tipo de dados recursivo T, contém em sua definição componentes do mesmo T. Este é um dos mecanismos mais fortes de estruturação implementados nas linguagens. A recursão permite a representação de estruturas dinâmicas complexas como por exemplo, árvore binária, lista encadeada, de forma bastante simples. Exemplo:

```
type MatrizEsparsa = ↑ElementoMatriz;

     ElementoMatriz = Record
                        indice  : integer;
                        valor   : real;
                        proximo : MatrizEsparsa
                          end;
```

Para definir o tipo MatrizEsparsa, utiliza-se o tipo ElementoMatriz. Para definir o tipo ElementoMatriz, utiliza-se o tipo MatrizEsparsa, através do componente próximo.

Tipos abstratos de dados: Um tipo abstrato de dados representa uma classe de dados definida pelo usuário em termos de suas propriedades abstratas. O uso de objetos desse tipo é feito unicamente através dessas propriedades, sem a preocupação de como essas propriedades abstratas foram implementadas. Horowitz, sugere que a especificação de um tipo abstrato de dados contenha três partes: uma especificação sintática, uma especificação semântica e uma especificação das restrições. A especificação sintática define o nome do tipo, suas operações e o tipo dos argumentos das operações; a especificação semântica contém um conjunto de equações algébricas que descreve as propriedades das operações independentemente da implementação; a especificação das restrições estabelece as condições que devem ser satisfeitas antes e depois da aplicação das operações. Exemplo: Uma lista linear, definida através de um ponteiro (POINTER) e de suas proriedades, tais como: Obter TOPO, Contar elementos, Colocar um elemento na lista, dentre outras. O usuário poderá utilizar a lista, através do tipo definido como

tipo abstrato, sem conhecer sua estrutura e como se processam suas operações abstratas.

Os efeitos dos tipos de dados na programação são:

Legibilidade: A escolha de nomes mais próximos da aplicação e a abstração representada pelos tipos de dados representam um papel fundamental na legibilidade dos programas. Quando a linguagem possibilita a definição de tipos enumerados, torna-se bastante evidente essa qualidade uma vez que se permite utilizar os mesmos nomes utilizados na aplicação.

Fatoração: A declaração de variáveis complexas repetidas vezes no programa é um gerador potencial de erros. O procedimento recomendado é o de definir esse tipo, dando-lhe um nome e utilizar esse nome na declaração das variáveis.

Correção: A utilização de tipos de dados especificados por intervalos bem definidos, garante uma correta modelagem das grandezas reais que tenham um intervalo bem definido no mundo real.

É também desejável que os programas tenham um comportamento previsivel e razoável: na presença de erros (como a divisão por zero, indices fora dos limites); em eventos que ocorrem com pouca freqüência (como fim de arquivo, overflow em uma operação aritmética); e na entrada de dados inválidos (através da digitação). Se o programa não souber tratar essas anomalias, ou o programa será interrompido e abortado, ou poderá criar inconsistências fornecendo resultados incorretos.

Quando a linguagem não fornece mecanismos para tratamento de exceções, essa preocupação fica a cargo do programador, que tratará essas exceções toda vez que elas ocorrerem, de uma forma geral complicando o algoritmo básico de solução do problema. Para manipular tais exceções é usual adicionar-se parâmetros de controle aos subprogramas, que informam a ocorrência ou não de exceções durante sua execução.

Se a linguagem permite invocar o tratamento de erros de forma automática, esse tratamento é feito em uma unidade chamada Tratador de Exceção. A ação de notificar a exceção, interromper a execução normal do programa e transferir o controle para o tratador de exceção é denominada sinalização da exceção (raise).

No FORTRAN 77 : Permite algum tratamento. Ex:

READ(.....,END = ‹r1›, ERR = ‹r2›)
...‹lista de variáveis›

No PASCAL : Não existe facilidade definida na linguagem.

Apesar de todo esforço para a produção de um programa correto, sua qualidade ainda deverá ser verificada para se obter a validação. O próprio computador é uma ferramenta útil para verificar a correção de programas. As formas mais comuns de verificação são feitas pelo compilador (sintaxe e compatibilidade de tipos) e os testes no programa.

Verificações Estáticas: São feitas durante a compilação dos programas. Algumas das verificações feitas estaticamente pelo compilador são: compatibilidade entre os parâmetros formais e reais de um subprograma; a correspondência entre a importação de variáveis e o uso correto de variáveis só para leitura (read-only); a existência de duas atribuições seguidas para uma mesma variável; a existência de variáveis declaradas e não utilizadas; a utilização de variáveis para as quais valores ainda não foram atribuidos, partes de códigos nunca utilizados.

Testes de Programas: É a forma mais utilizada para verificar correção de programas. São utilizados dois esquemas para teste e correção de programas:

A execução do programa com entradas de teste, produzindo listagens de valores intermediários e finais conhecidos que podem ser analizados; e

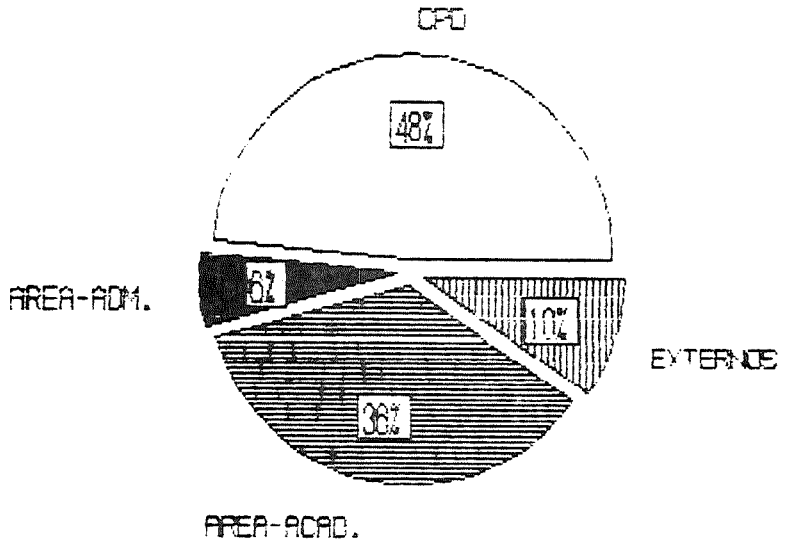
A utilização de depuradores simbólicos.

Além desses esquemas, o uso de ferramentas que possibilitam identificar trechos críticos em um programa, ou seja, trechos que são muito executados, contribuem sensivelmente no processo de otimização dos programas.

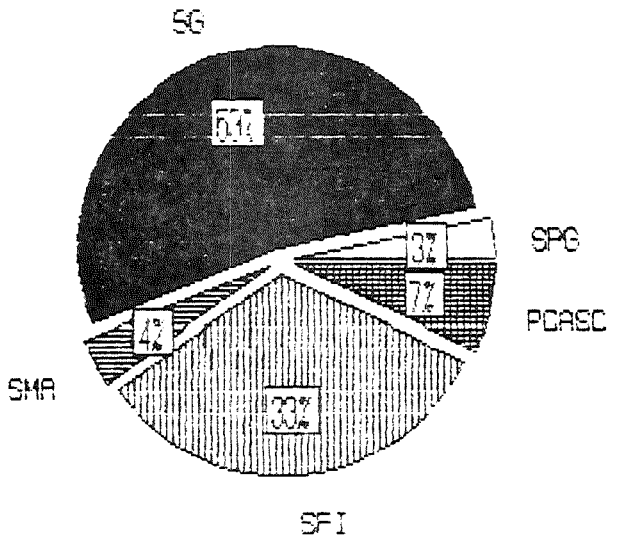
Espera-se ainda que o projeto de novas linguagens ou ferramentas associadas às atuais linguagens, leve em conta a possibilidade de se provar formalmente que um programa está correto. O teste de um programa só mostra que ele funciona corretamente para as condições e dados testados. A verificação formal de um programa demonstra que ele é correto para qualquer condição de entrada de dados.

UTILIZACAO DO COMPUTADOR IBM-4341

UNIDADES DE USO GASTAS - GERAL - NOV/88

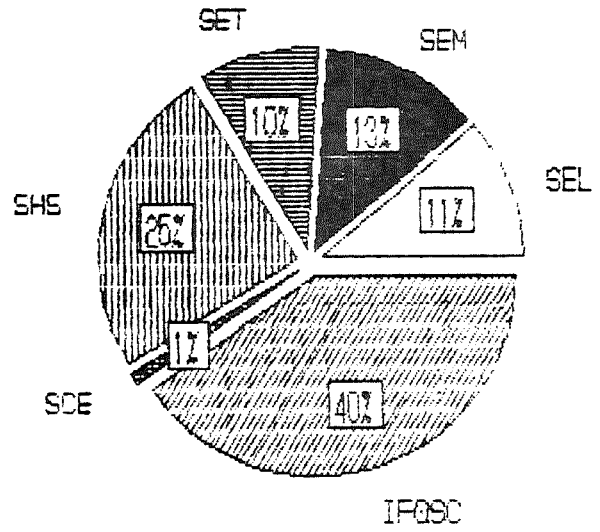


UNIDADES DE USO GASTAS
AREA ADMINISTRATIVA - NOV/88

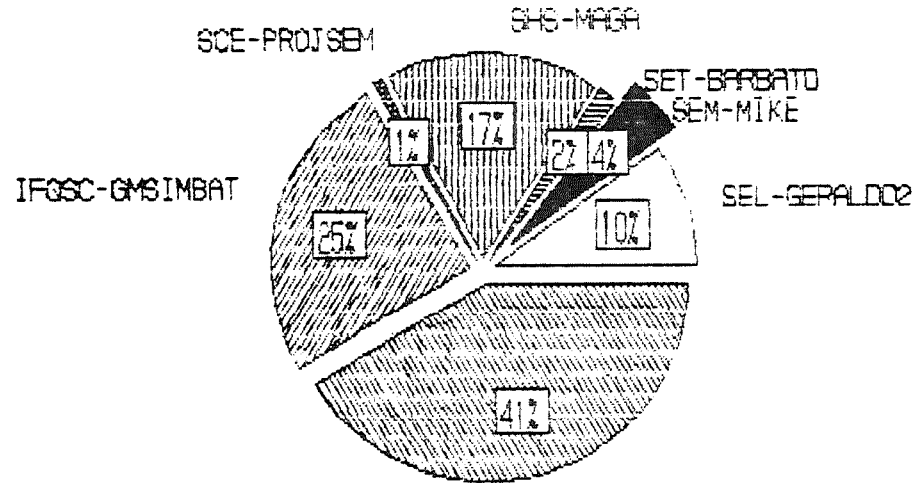


UTILIZACAO DO COMPUTADOR IBM-4341

UNIDADES DE USO GASTAS
AREA ACADEMICA - NOV/89



UNIDADES DE USO GASTAS
AREA ACADEMICA MAIORES USUARIOS-NOV/89



UTILIZACAO DO COMPUTADOR IBM-4341

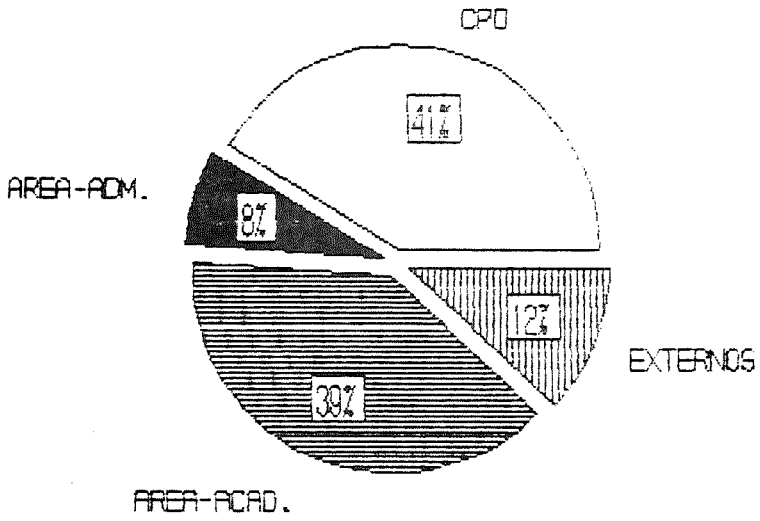
UNIDADES DE USO GASTAS POR GRUPOS DE USUARIOS

TIPO DE USUARIO	GERAL	AREA ADMINIST.	AREA ACADEMICA	MAIORES USUARIO
CPD	268768.4			
AREA ADM.	32042.4			
AREA ACAD.	200365.2			
EXTERNOS	53296.6			
SPG		1099.5		
SG		18120.1		
SMA		1283.6		
SFI		11353.2		
PCASC		2398.5		
SEL			22415.3	
SEM			25280.6	
SET			19586.3	
SHS			49650.9	
IFQSC			81519.7	
SCE			930.8	
SEL GERALDO2				19544.7
SEM MIKE				8609.1
SET BARBATO				3157.9
SHS MAGA				33611.7
IFQSC GMSIMBAT				49368.8
SCE PROJSEM				1771.4
OUTROS USUARIOS				85143.8

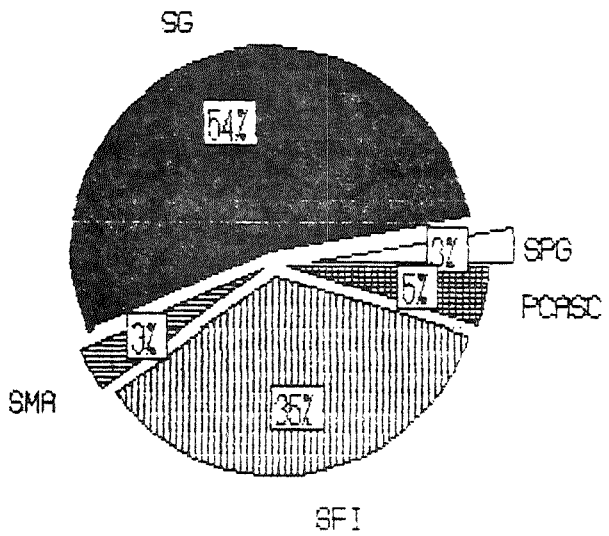
VALOR DA UNIDADE DE USO PARA NOVEMBRO = NCZ$ 0,10226

UTILIZACAO DO COMPUTADOR IBM-4341

UNIDADES DE USO GASTAS - GERAL - DEZ/89

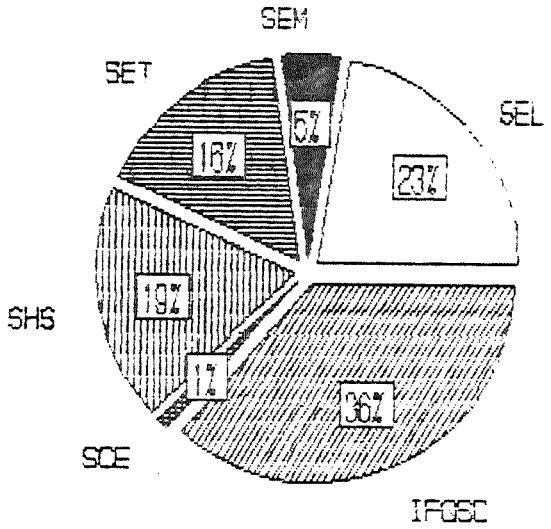


UNIDADES DE USO GASTAS
AREA ADMINISTRATIVA - DEZ/89

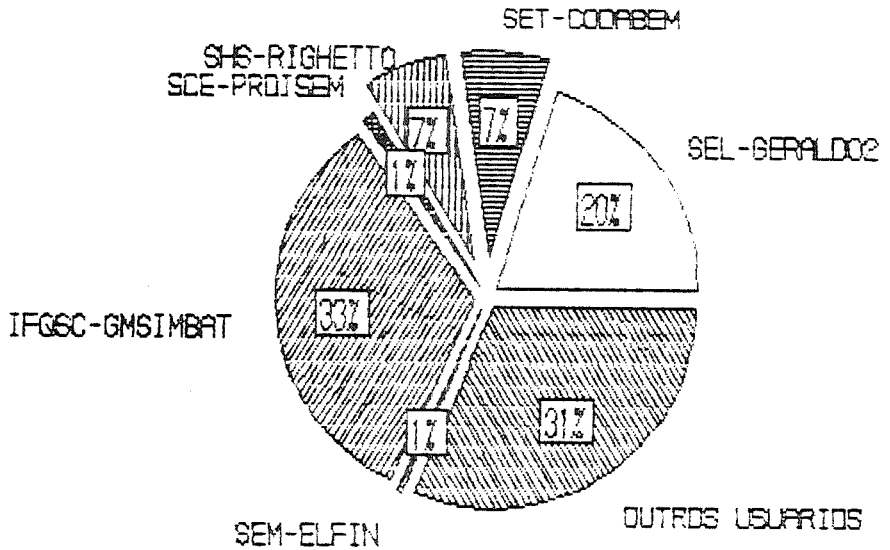


UTILIZACAO DO COMPUTADOR IBM-4341

UNIDADES DE USO GASTAS
AREA ACADEMICA - DEZ/89



UNIDADES DE USO GASTAS
AREA ACADEMICA MAIORES USUARIOS DEZ/89



UTILIZACAO DO COMPUTADOR IBM-4341

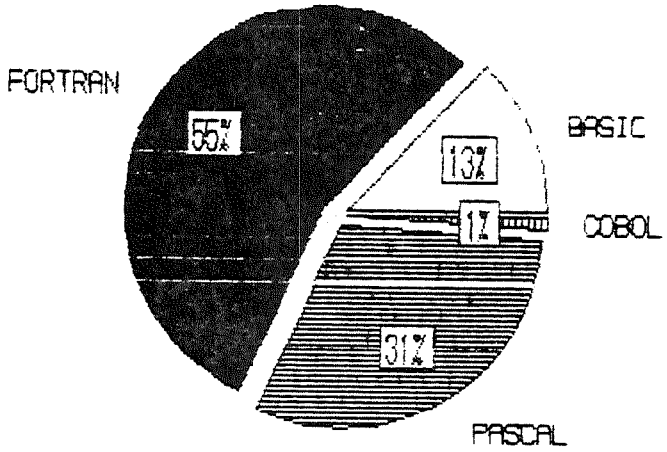
UNIDADES DE USO GASTAS POR GRUPOS DE USUARIOS

TIPO DE USUARIO	GERAL	AREA ADMINIST.	AREA ACADEMICA	MAIORES USUARIO
CPD	167466.7			
AREA ADM.	30610.5			
AREA ACAD.	157631.1			
EXTERNOS	48345.2			
SPG		854.8		
SG		17423.1		
SMA		1057.7		
SFI		11088.9		
PCASC		1485.0		
SEL			35523.2	
SEM			7712.4	
SET			25183.8	
SHS			29864.6	
IFQSC			57487.7	
SCE			1771.4	
SEL GERALDO2				32215.0
SEM ELFIN				898.4
SET CODABEM				11265.9
SHS RIGHETTO				10351.2
IFQSC GMSIMBAT				51857.5
SCE PROJSEM				1771.4
OUTROS USUARIOS				49271.7

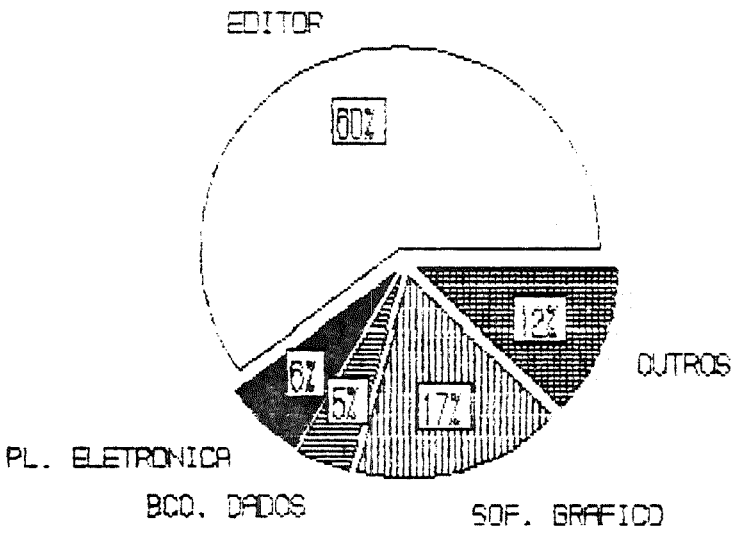
VALOR DA UNIDADE DE USO PARA DEZEMBRO = NCZ$ 0,14034

UTILIZACAO DE MICROCOMPUTADORES

LINGUAGENS UTILIZADAS - NOVEMBRO/89

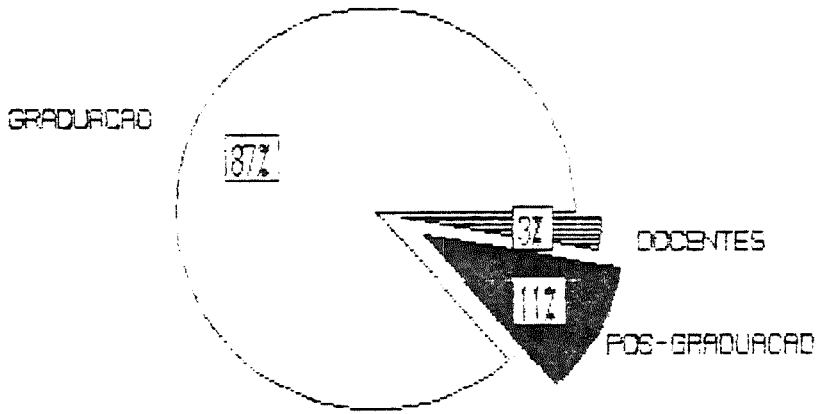


UTILITARIOS UTILIZADOS - NOVEMBRO/89



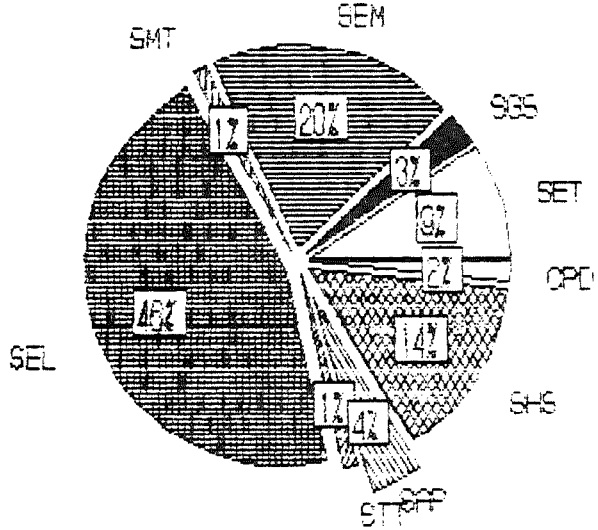
UTILIZACAO DE MICROCOMPUTADORES

ATIVIDADES CADASTRADAS - NOVEMBRO/89



DEPARTAMENTOS DA EESC CADASTRADOS

NOVEMBRO /89



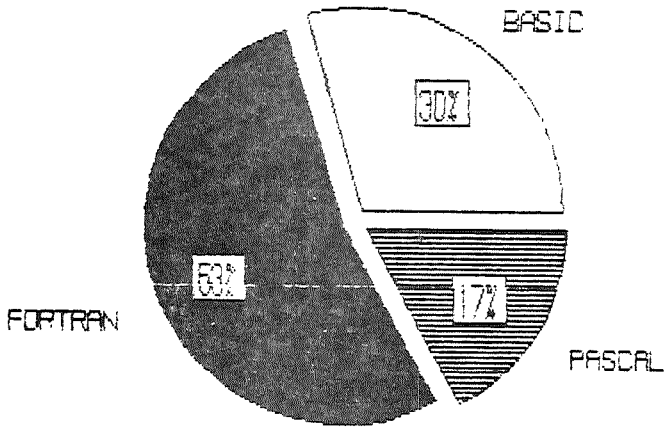
UTILIZACAO DOS MICROCOMPUTADORES

NOVEMBRO DE 1989

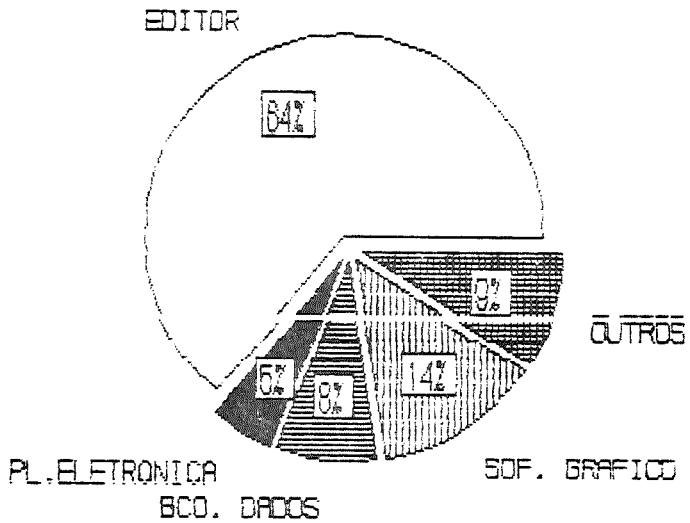
UTILIZACAO MENSAL POR SOFTWARE	
Linguagens	Usuarios
BASIC	66
FORTRAN	299
PASCAL	157
COBOL	2
Total	524
Utilitarios	Usuarios
EDITOR	218
BANCO DE DADOS	17
PLANIL.ELETRONICA	23
SOFTWARE GRAFICO	61
OUTROS	44
Total	363
CADASTRAMENTO MENSAL POR DEPARTAMENTOS	
Departamentos	Usuarios
SET	40
SGS	12
SEM	85
SMT	4
SEL	195
STI	05
SAP	18
SHS	62
CPD	7
Total	428
CADASTRAMENTO SEMESTRAL POR ATIVIDADES	
Atividades	Usuarios
GRADUACAO	985
POS-GRADUACAO	122
DOCENTES/ALUNOS	30
Total	1137

UTILIZACAO DE MICROCOMPUTADORES

LINGUAGENS UTILIZADAS - DEZEMBRO/89

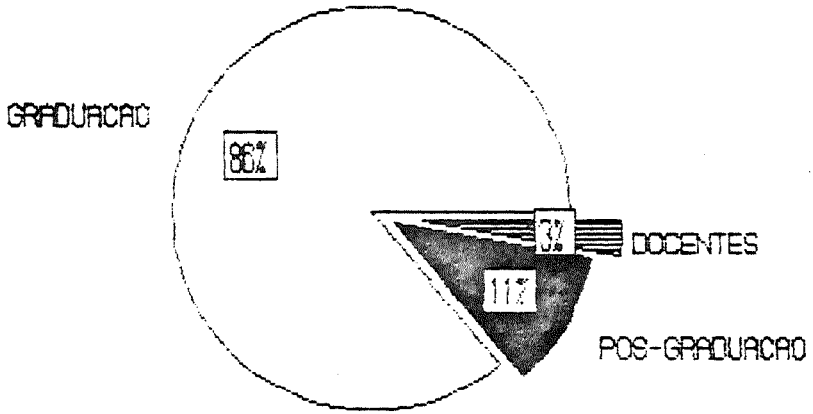


UTILITARIOS UTILIZADOS - DEZEMBRO/89



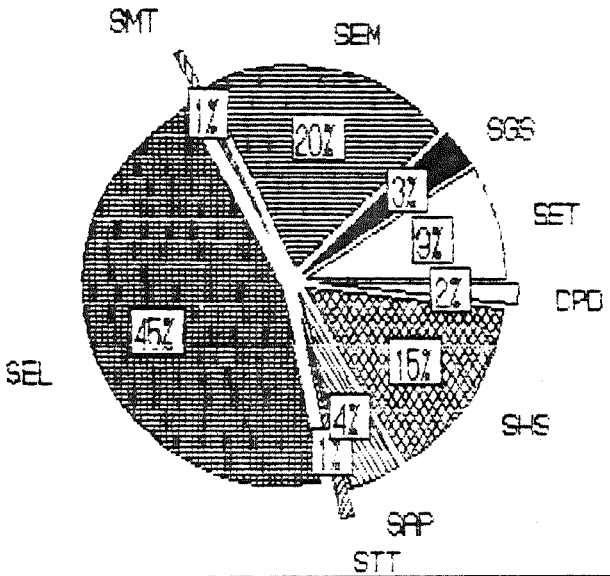
UTILIZACAO DE MICROCOMPUTADORES

ATIVIDADES CADASTRADAS - DEZEMBRO/89



DEPARTAMENTOS DA EESC CADASTRADOS

DEZEMBRO/89



UTILIZACAO DOS MICROCOMPUTADORES

DEZEMBRO DE 1989

UTILIZACAO MENSAL POR SOFTWARE	
Linguagens	Usuarios
BASIC	52
FORTRAN	93
PASCAL	30
COBOL	0
Total	175
Utilitarios	Usuarios
EDITOR	85
BANCO DE DADOS	11
PLANIL.ELETRONICA	7
SOFTWARE GRAFICO	18
OUTROS	12
Total	133
CADASTRAMENTO MENSAL POR DEPARTAMENTOS	
Departamentos	Usuarios
SET	40
SGS	13
SEM	86
SMT	4
SEL	195
STI	05
SAP	18
SHS	65
CPD	7
Total	433
CADASTRAMENTO SEMESTRAL POR ATIVIDADES	
Atividades	Usuarios
GRADUACAO	986
POS-GRADUACAO	126
DOCENTES/ALUNOS	30
Total	1142