1 parte: modularização, tempo de amarração
2 parte: polimorfismo, exceção, concorrência

ESPECIFICAÇÕES DA LP

O léxico da LP corresponde ao vocabulário que pode ser utilizado para formar sentenças na linguagem. A sintaxe da LP corresponde ao conjunto de regras que determinam quais sentenças podem ser formadas a partir da combinação dos itens léxicos. O léxico e sintaxe estão relacionados com a forma dos programas, isto é, como expressões, comandos, declarações e outros elementos da LP podem ser combinados para formar programas válidos. A semântica da LP descreve como as construções sintaticamente corretas são interpretadas ou executadas. A semântica está relacionada com o significado dos programas, isto é, como eles se comportam quando executados por computadores

a = b;

Léxico: a, = , b, ; fazem parte do vocabulário da LP
Sintaxe: a sentença designa um comando válido de atribuição
Semântica: comando que substitui o valor de a pelo atual de b

IDENTIFICADOR	ENTIDADE	TEMPO DE AMARRAÇÃO
*	operação de multiplicação	projeto da LP
int	intervalo de inteiros	implementação do compilador(C) projeto da lp (java)
variável	associação de uma variável a um tipo de dado	compilação(c)
função	código correspondente à função	ligação
variável global	variável em memória	carga do programa
variável local	variável em memória	execução

Tempo de Compilação: Um grande número de amarrações ocorre no momento em que o programa é compilado. São exemplos desse tipo de amarração em C a associação de uma variável a um tipo de dados e a associação, em uma expressão do programa, do operador * à operação que denota.

TEMPOS DE AMARRAÇÃO
estática: ocorre antes da execução do programa e permanece
dinâmica: ocorre ou é alterada durante a execução do programa

AMBIENTES DE AMARRAÇÃO
ESCOPO: estático ou dinâmico

DEFINIÇÕES E DECLARAÇÕES
Definições produzem amarrações entre identificadores e entidades criadas na própria definição
Declarações produzem amarrações entre identificadores e entidades já criadas ou que ainda o serão

Definições Tipos em C
```
struct data {
        int d, m, a;
};
```
Declarações Tipos em C
```
struct data;
```

Definições de Variáveis em C
```
int k;
union angulo ang;
int *p, i, j, k, v[10];
```

Definições com Inicialização
```
int i = 0;
float f, g = 3.59;
int j, k, l = 0, m=23;
```

Definição de Subprogramas em C (
```
int soma (int a, int b) {
        return a + b;
}
```

Declaração de Subprogramas em C (Declaração: apenas cabeçalho A definição está em outro trecho do programa)
```
int incr (int);
```

VALORES E TIPOS DE DADOS

PRODUTO CARTESEANO (struct em c)
```
struct nome {
        char primeiro [20];
        char meio [10];
        char sobrenome [20];
 }
```

```
struct empregado {
        int id;
        struct nome nfunc;
        float salario;
} emp;
```

Cardinalidade (quantos elementos existem)
#(char)^20*#(char)^10*#(char)^20

E qual cardinalidade da estrutra empregado?
#int * ((#char)20 * (#char)10 * (#char)20) * #float

UNIÕES

```
union medida {
         int centimetros;
        float metros;
 }
```

MAPEAMENTOS;
mapeamento finito: vetores -> int v[7]

MODULARIZAÇÃO
Parâmetro formal:Identificadores listados no cabeçalho do subprograma e usados no seu corpo
Parâmetro real: Valores, identificadores ou expressões utilizados na chamada do subprograma
Argumento: Valor passado do parâmetro real para o parâmetro formal

Direção de Passagem:
unidirecional de entrada com parâmetro formal variável
unidirecional de entrada com parâmetro formal constante - não é possível fazer atribuição ao parâmetro formal
unidirecional de saída - Na passagem unidirecional de saída o fluxo vai do parâmetro formal para o parâmetro real
bidirecional de entrada e saída

Passagem de parâmetro
Cópia
Referência

Uma contribuição do mecanismo de cópia é viabilizar a passagem unidirecional de entrada variável de parâmetros. Essa direção de passagem de parâmetros só pode ser feita através do mecanismo de cópia, uma vez que o uso do mecanismo de referência implicaria na modificação do parâmetro real.

Momento da Passagem de Parâmetros

No modo normal (eager) a avaliação ocorre no momento da chamada do subprograma. No modo por nome (by name) a avaliação ocorre em todos os momentos em que o parâmetro formal é usado. No modo preguiçoso (lazy) a avaliação ocorre no primeiro momento em que o parâmetro formal é usado.

Tipos de Dados
Tipos Anônimos: Só pode ser usada uma única vez em virtude do tipo não possuir um nome. Se precisar de outra variável “igual” tem que repetir toda a definição

```
struct {
        int elem[100];
                int topo;
} pilhaNumeros;
```

Tipos Simples:
Agrupam dados relacionados em uma única entidade nomeada Aumentam reusabilidade, redigibilidade e legibilidade. Uma vez definido o tipo tPilha, pode-se criar quantas variáveis desejar desse tipo

Tipos Abstratos de Dados:
Conjuntos de valores com comportamento uniforme definido por operações Em LPs, TADs possuem representação e operações especificadas pelo programador. Em java: Classes String, Hash, Array, Map, Tree, etc..

Pacotes
Só subprogramas e tipos não são suficientes para sistemas de grande porte
Fontes de código são coleções de entidades reutilizáveis de computação

Modularização, Arquivos e Compilação Separada
Uso de arquivo único causa problemas Redação e modificação se tornam mais difíceis
Reutilização apenas com processo de copiar e colar

Compilação Separada de Arquivos
Ligação para gerar executável
Perda da verificação de tipos

Vantagens da Modularização
Melhoria da legibilidade: Divisão lógica do programa em unidades funcionais Separação do código de implementação do código de uso da abstração
Aprimoramento da redigibilidade: Mais fácil escrever código em vários módulos do que em um módulo único

Aumento da modificabilidade: Alteração no módulo é localizada e não impacta código usuário
Incremento da reusabilidade: Módulo pode ser usado sempre que sua funcionalidade é requerida
Aumento da produtividade de programação: Compilação separada Divisão em equipes
Maior confiabilidade: Verificação independente e extensiva dos módulos antes do uso
Suporte a técnicas de desenvolvimento de software

Polimorfismo
Verificação Estática de Tipos:
Todos os parâmetros e variáveis devem possuir um tipo fixo identificável a partir da análise do texto do programa. Tipo de cada expressão pode ser identificado e cada operação pode ser verificada em tempo de compilação

Verificação Dinâmica de Tipos
Em tempo de execução. Somente os valores dessas LPs têm tipo fixo. Cada valor tem associado a ele uma tag indicando o seu tipo. Uma variável ou parâmetro não possui um tipo associado. Pode designar valores de diferentes tipos em pontos distintos da execução. Verificação dos tipos de operandos imediatamente antes da execução da operação

Verificação de Tipos Mista
Maior parte das verificações de tipos em tempo de compilação, porém algumas verificações em tempo de execução Programas em LPs orientadas a objetos podem precisar verificar a conversão de tipos de objetos durante a execução

Sistemas de Tipos Monomórficos
Linguagens monomórficas exigem que se crie representação e operações distintas para cada tipo de elemento. Lista de inteiros, lista de cadeia de caracteres, etc

Sistemas de Tipos Polimórficos
Favorecem a construção e uso de estruturas de dados e algoritmos que atuam sobre elementos de tipos diversos Subprogramas polimórficos → parâmetros e tipo de retorno podem assumir valores de mais de um tipo. Tipos de dados polimórficos → as operações associadas são aplicáveis a valores de mais de um tipo

Tipos de Polimorfismo
Ad-hoc - Se aplica apenas a subprogramas
Coerção - Significa conversão implícita de tipos

```
void funcao (float i) { }
main() {
        long num;
        funcao (num);
}
```

funcao aparenta lidar com float e long -> Na verdade, funcao lida apenas com floa ->
Compilador se encarrega de embutir código para transformar long em float
Compiladores podem ter tabelas de conversões permitidas

Ampliação
Tipo de menor conjunto de valores para tipo de maior conjunto
Estreitamento
Tipo de maior conjunto de valores para tipo de menor conjunto
Nem sempre é ampliação ou estreitamento: int para unsigned em C

```
main() {
        int i;
        char c = ‘a’;
        float x;
        i = c;  1 - o valor char de c é convertido em int
        c = i + 1; 2 - : o valor int de i é convertido em char
        x = i; 3 - o valor int de i é convertido em float
        i = x / 7;  4 - o valor float de x é convertido em int
}
```

Menor confiabilidade pois podem impedir a detecção de certos tipos de erros por parte do compilador
Maior redigibilidade por não demandar chamada de funções de conversão

Sobrecarga - Quando identificador ou operador é usado para designar duas ou mais operações distintas. Sugere que determinada operação ou subprograma pode ser realizada com operandos de tipos diferentes mas não é isso que ocorre

Sobrecarga ocorre quando escrevemos o mesmo método ou função, porém usando tipos de parametros diferentes ou quantidade de parametros diferentes. Ex: int calcula(int x, int y) e int calcula(int x, int y, int z), porém repare que mudar o tipo de retorno não vale para esse polimorfismo

Programas podem ficar mais fáceis de serem lidos e redigidos com a sobrecarga
Aumenta a complexidade da LP
Pode ser mal utilizado, tendo efeito contrário a legibilidade
JAVA não inclui a sobrecarga de operadores por considerá-la capaz de gerar confusões e aumentar a complexidade da LP

Universal - Se aplica a subprogramas e estruturas de dados
Paramétrico - A principal característica desse polimorfismo é a parametrização das estruturas de dados e subprogramas com relação ao tipo do elemento sobre o qual operam. Pode-se dizer, então, que essas abstrações recebem um parâmetro implícito adicional especificando o tipo sobre o qual elas agem.

Em C:

```
T identidade (T x) {
        return x;
}
```

Se trata de ter métodos genéricos que podem lidar com vários tipos de dados

Inclusão
Polimorfismo - Característico de linguagens orientadas a objetos. Uso de hierarquia de tipos para criação de subprogramas e estruturas de dados polimórficos. Subclasses herdam os atributos e métodos de uma classe e, portanto, implementam subtipos do tipo definido por essa classe

Sobrescrição - Método herdado não é adequado para realizar a mesma operação nos objetos das subclasses. Pode usar super.método para chamar o método da superclasse

Concorrência
Termo usado em computação para designar situações nas quais diferentes processos competem pela utilização de algum recurso

Programas de computador são seqüências de instruções passíveis de serem executadas por um processador. Programas em execução são chamados de processos
Diferente de programas, processos são entidades ativas cujo estado é alterado durante a sua execução

Programa -: estático
Processo - dinâmico

Possíveis estados de um processo:
Novo
Executável
Em execução
Em espera
Encerrado

Time Slices
O sistema operacional estabelece fatias de tempo (time slices) para que cada processo tenha posse do processador por um certo período, dando a impressão de que eles estão sendo executados ao mesmo tempo.

E quando um mesmo processo que executar atividades em paralelo? Threads

Threads são fluxos de execução concorrentes que compartilham recursos do mesmo processo do qual são originários

Vantagens de Threads
São leves
Compartilham memória com o processo que o criou e com as demais threads
Possibilitam a utilização de mais de um método ou função de uma mesma aplicação, simultaneamente

Processos Concorrentes
Programas concorrentes são não-determinísticos
    Ordem de execução das instruções não é determinada previamente
Isso pode acarretar em alguns problemas
Lockout (trancamento) - Dois ou mais processos ficam esperando por um evento que nunca acontecerá
Deadlock (impasse) - Todos os processos estão bloqueados esperando por um evento, que só pode ser gerado por outro processo também bloqueado
Starvation (inanição) - Um processo tem a aquisição de um recurso postergada indefinidamente. Ex: Processos de baixa prioridade onde não há mecanismo de fila ou similares
Indeterminismo - O **indeterminismo** em processos concorrentes refere-se ao comportamento imprevisível que pode ocorrer em sistemas onde múltiplos processos ou threads são executados simultaneamente. Em um ambiente concorrente, o **resultado final** da execução de processos que interagem entre si pode **variar** de acordo com a ordem em que as operações são executadas, o que pode não ser controlável ou previsível devido à natureza do agendamento dos processos.

Tipos de Interação
Sem Interferência / Competição
Processos independentes que apenas concorrem com outros processos pela utilização de um mesmo recurso. Sincronização para utilização dos recursos é feita pelo sistema operacional, não provocando qualquer dificuldade para os programadores

Cooperação
Processo afeta ou é afetado pela execução de outro processo em prol da realização de uma atividade.Para que ocorra a cooperação entre processos é necessário que exista uma forma de comunicação entre eles.
    Troca de mensagens
    Compartilhamento de memória

Troca de Mensagens
Características
    Processos podem estar em máquinas diferentes

Não compartilham a mesma memória física
Troca intensa de mensagens pode gerar sobrecarga (overhead)

Duas chamadas de sistema básicas
send (envio)
receive (recepçã

Troca de mensagens Direta
Envia mensagem diretamente para outro processo
Necessário saber o id do processo para o qual está enviando a mensagem

Troca de mensagens Indireta
Envia mensagem para caixas postais
Necessário saber o id da caixa postal do destinatário e ter permissão para escrita nessa caixa
Cada processo retira as mensagens de suas caixas postais

A comunicação de processos através de troca de mensagens
Bloqueante → envia a mensagem e fica aguardando a confirmação do destinatário
Não-bloqueante → não espera. Envia a mensagem e continua sua execução

Rendezvous (encontro)
Situações em que não há recurso para armazenar mensagens, como buffer inexistente ou cheio
Processo que envia a mensagem é bloqueado para entregar a mensagem diretamente ao destinatário

Produtor e Consumidor
No problema do “Produtor e do Consumidor” há um processo que produz algo e um processo que utiliza o que foi produzido. O início da execução do consumidor dependente do término da execução do produtor. Utilizar processos concorrentes seria uma abordagem mais apropriada. Para evitar o problema de indeterminismo deve haver algum tipo de sincronização entre os processos, para garantir acesso exclusivo a sua região crítica

Semáforos
Mecanismo de sincronização entre processos. Um semáforo é um tipo abstrato de dados que possui um valor inteiro, uma lista de processos em espera e duas operações
P (do holandês Proberen, testar)
V (do holandês Verhogen, incrementar)

Para garantir exclusão mútua a uma determinada região (região crítica), cada processo deve chamar a operação P antes de acessar tal região e chamar a operação V após sair dessa região

Operação P sobre um semáforo S

Testa se o processo que executou P(S) pode ou não entrar na região crítica

Operação V sobre um semáforo S

Sinaliza ao semáforo que o processo não está mais na região crítica e retira outro processo da fila de espera

Programação Concorrente Estruturada

Outra solução: mecanismos que deixam para o compilador a responsabilidade de garantir o acesso exclusivo à região crítica

Regiões críticas condicionais

Monitores

Regiões críticas condicionais

Toda variável compartilhada necessita ser declarada como tal

Compilador pode inserir automaticamente as operações P(S) e V(S) nas posições apropriadas

Monitores

Mecanismos de sincronização compostos por um conjunto de variáveis, procedimentos e estruturas de dados dentro de um módulo. Finalidade: sincronização automática, garantindo exclusão mútua entre seus procedimentos

Nenhum procedimento dentro do monitor pode ser chamado por mais de um processo simultaneamente

Fila de espera

Ausência de Mecanismos em LPs(C)

Chamadas de sistema

Bibliotecas de funções (específicas da plataforma de execução)

Chamadas fork

Permite a duplicação de um processo

SO cria um processo idêntico (filho) ao processo original (pai)

Ambos os processos continuam a execução a partir do ponto onde o fork foi chamado

A diferença está no valor de retorno

No processo pai, fork retorna o PID do processo filho

No processo filho, fork retorna 0

Recursos compartilhados

Após o fork, os processos pai e filho podem ser executados simultaneamente

Sincronização: Necessidade de mecanismos para coordenar os processos

Threads em LPs OO

POSIX threads

Padrão adotado para manipular threads em C

API padrão possui aproximadamente 60 funções

Classes Threads

- Padrão adotado pelo JAVA
- Herdando da classe java.lang.Thread
- Implementando a interface java.lang.Runnable

Quando métodos de um objeto em JAVA são declarados como synchronized

- public synchronized void inserir(int x) { …
- JVM faz com que threads chamem esses métodos em exclusão mútua