ডাটাবেজ ল্যাংগুয়েজ হিসাবে ক্লিপারের ব্যবহার

এরিক ডি সিলভা (রবিন)

কোন কম্পাইলার ল্যাংগুয়েজের সর্বোত্তম ব্যবহার নির্ভর করে প্রোগ্রামার ঐ কম্পাইলারের অভ্যন্তরীণ বিষয়গুলো কতটুকু জানেন তার উপর।

পূর্বেই বলেছি যে, ক্লিপার C-তে লেখা হয়েছে। তাই অধিকাংশ ANSI সাপোর্টেড সি কম্পাইলারের সাথে ক্লিপারের অভ্যন্তরীণ ব্যাপারে মিল পাওয়া যাবে। অবশ্য যদিও ডাটাবেজ প্রোগ্রামিংয়ের খাতিরে ক্লিপার সি-এর অনেক স্ট্রাকচারই লুকিয়ে ব্যবহার করে, তথাপি ভার্সন ৫.১ হতে একে কিছুটা সমনে আনা হয়েছে।

সত্যিকার অর্থে ক্লিপারের স্ক্রীন ইন্টারফেস ও প্রেজেনটেশন ম্যানেজার পুরোপুরি সি-কে ব্যবহার করে তৈরি হয়েছে। আবার নিকট ভবিষ্যতে ক্লিপারের জন্য যে গ্রাফিক্যাল লাইব্রেরী ডেভেলপমেন্টের কাজ চলছে তাও সি নির্ভরশীল হবে বলে আশা করা যায়। অবশ্য এজন্য ব্রায়ান রাসেল, ইয়ালিক এবং রিবে রিক স্পেন্সদের মত ক্লিপারের লাইব্রেরী ডেভেলপারদের সি-এর BGI এবং CHR ফাইলগুলো নিয়ে কিছু নতুন চিন্তা করতে হতে পারে।

এখানে, ক্লিপারের সাথে সি-কে নিয়ে আলোচনায় আসার মূল উদ্দেশ্য হচ্ছে বর্তমানে সফটওয়্যার ডেভেলপমেন্টে সর্বাপেক্ষা জনপ্রিয় ও বহুল ব্যবহৃত ল্যাংগুয়েজ হচ্ছে সি। তাই যারা সি দিয়ে ডাটাবেজ ডেভেলপমেন্টের চিন্তাভাবনা করছেন তাদের নতুন করে হাজার হাজার লাইন কোড লেখার কোনপ্রয়োজন আছে বলে মনে করি না। বরং ক্লিপারকে এ কাজে ব্যবহার করে আপনি প্রয়োজনীয় উপযোগ পেতে পারেন।

এবার তাহলে মূল আলোচনা শুরু করা যাক। শুরুতে আমরা ক্লিপারের মেমোরী ম্যাপ নিয়ে কিছু আলোচনা করবো এবং এর পরেই আমরা ধীরে ধীরে এর আইডেন্টিফায়ার (ভেরিয়েবল) ও ফাংশন ইত্যাদি নিয়ে আলোচনা করবো।

ক্লিপারে এবং এতে লেখা প্রোগ্রাম সাধারণত ৮০৮৬ এমুলেশন মুডে রান করে। এর অর্থ হচ্ছে ১৬ বিটের XT কমপিউটার হতে আরম্ভ করে ৬৪ বিটের পিসি কিংবা সার্ভার পরিবেশে সমানভাবে ক্লিপার চালানো যাবে। সি-এর অটো কম্পাটিবিলিটির মুল রহস্যও এখানে।

অনেকেই বলেন যে ৮৬ ভিত্তিক প্রসেসরগুলোতে মেমরী এড্রেস করা হয় সেগমেন্টঃ অফসেট ভিত্তিতে। অর্থাৎ কোড এবং ডাটা লোড করার জন্য এবং ঐ ডাটা হতে নির্দিষ্ট বাইট ট্রেস করার জন্য সেগমেন্ট অফসেট প্রয়োজন হয়।

ছোট খাট প্রোগ্রাম যেমন com ফাইল বা TSR গুলো সাধারণতঃ একটি সেগমেন্ট কোড, ডাটা স্ট্যাক এবং এক্সট্রা সেগমেন্ট ধারণ করে। এ রকম প্রোগ্রামের আয়তন ৬৪ কি. বাইটের অধিক হতে পারে না। এই সমস্ত প্রোগ্রাম মেমোরী এড্রেসের জন্য near pointer ব্যবহার করে। Near pointer-এড্রেসিং পদ্ধতিতে রেজিস্টারে কেবল মাত্র এড্রেসের অফসেট লোড করতে হয় বলে এদের গতি অনেক দ্রুত হয়।

তবে ডাটাবেজ প্রোগ্রাম যেহেতু অধিক সংখ্যক ডাটা এবং কোড নিয়ে কাজ করে তাই এ ক্ষেত্রে পূর্বের নিয়ম ঘটবে না। ডাটাবেজ প্রোগ্রামগুলোর অধিকাংশই মেমোরী এড্রেসের জন্য Far pointer ব্যবহার করে, এই পদ্ধতিতে সেগমেন্ট প্রসেস করার জন্য CPV রেজিস্টারে এড্রেসের সেগমেন্ট ও অফসেট দুটোই লোড করা হয় এতে রান টাইম স্পীড কমে যায় তবে একটি Far পয়েন্টার ১ মেগাবাইট মেমোরী স্পেসের ভিতরে যে কোন এড্রেস লোকেট করতে পারে। এই নিয়মে আবার EMS, XMS মেমোরীও প্রসেস করা যায়– কিছু ১৬ বা বেশি কিলোবাইটের Page ব্যবহার করে।

Far পয়েন্টার এড্রেসিং পদ্ধতিতে ব্যবহার করে যে প্রোগ্রাম তাকে আমরা Large মডিউল এবং Huge মডিউল বলে থাকি। এর মধ্যে ক্লিপার হচ্ছে Large মডিউলের অন্তর্গত একটি কম্পাইলার এবং এতে লেখা সব প্রোগ্রামই এ মডিউল ব্যবহার করে। তবে এই মডিউলের একটি অসুবিধা হচ্ছে এর ডাটার কোন একক আইটেম ৬৪ কি. বাইটের বেশি হতে পারে না। (যেমন ভেরিয়েবল, এরে)

Far পয়েন্টার ব্যবহারের ফলে ক্লিপার যদিও কোড এবং ডাটার জন্য একাধিক সেগমেন্ট ব্যবহার করতে পারে তথাপি এজন্য একে দুটো খুব গুরুত্বপূর্ণ টার্বো পাওয়ার হারাতে হয়েছে। প্রথমতঃ ফ্লোটিং পয়েন্ট, কল জেনারেট করার জন্য এটিকে লাইব্রেরীর অলটারনেট ম্যাথ প্যাকেজ করতে হয় অর্থাৎ সরাসরি কাজ করতে পারে না। দ্বিতীয়তঃ ২০ বিটের ডাটা এড্রেস করার সময় এটি স্পীড বৃদ্ধির জন্য কোন স্ট্যাক প্রোব জেনারেট করতে পারে না।

অবশ্য অধিকাংশ ডাটাবেজ ল্যাংগুয়েজেই এই অসুবিধা রয়েছে। তবে এখন হতে ক্লিপারের এ সমস্ত অভ্যন্তরীন অথচ গুঢ় সমস্যা সমাধানের চেষ্টা চলছে। যেমন– কম্পাইলারের সাথে বর্তমানে /N সুইচটি ব্যবহার করে মেমোরীতে লোকাল ও স্ট্যাটিক (পরে আলোচিত হবে) ভেরিয়েবলের জন্য সিম্বল টেবিল তৈরি করা হয় না। এতে রান টাইমে প্রোগ্রাম কোডের ভাল কম্পাক্টনেস আসে। বিশেষতঃ ডিবেজের পাবলিক ভেরিয়েবলগুলোকে এক্সটারনাল (ফাইলওয়াইজ) স্ট্যাটিক এবং প্রাইভেট ভেরিয়েবলগুলোকে লোকাল ডিক্লেয়ার করে নিয়ে ভাল পরিমান অপটিমাইজেশন সম্ভব।

প্রোগ্রামিংয়ের কিছু মৌলিক বিষয় নিয়ে আমরা এখানে পর্যায়ক্রমে সংক্ষিপ্ত আলোচনা করবো যার ফলে ব্যবহারকারীরা ক্লিপারের প্রোগ্রাম স্ট্রাকচার এবং এলগরিদম তৈরির কিছু ধারণা পাবেন।

মেমোরী ভেরিয়েবলঃ ক্লিপারে সর্বমোট আট প্রকারের মেমোরী ভেরিয়েবল ডিক্লেয়ার করা যায় [ডিক্লেয়ার নিয়ে পরে আলোচনা করছি] এগুলো হল Array, Block, Object, Character, Date, Logical, Numeric এবং Memo. এর মধ্যে শেষোক্ত ৫টির সম্পর্কে ডিবেজ ব্যবহারকারীরা পরিচিত। এখানে আমরা Array, Block এবং Object ভেরিয়েবলের Type নিয়ে আলোচনা করব।

Array ঃ ক্লিপারে দুই ধরনের Array আছে। প্রথমটি ওয়ান ডাইমেনশনাল এবং দ্বিতীয়টি মাল্টি। কোন ভেরিয়েবলকে এরে হিসাবে ব্যবহার করতে চাইলে প্রথমে একে ডিক্লেয়ার করে নিতে হয়। এর ফলে ঐ ভেরিয়েবলটি একটি Array হিসাবে গণ্য হয় এবং এর প্রতিটি এলিমেন্টকে নিউমেরিক Value দিয়ে INDEX করতে হয়। Array ডিক্লেয়ার করার জন্য ক্লিপারে Declare নামক একটি কমাণ্ড ও Array () নামক একটি ফাংশন রয়েছে। নীচের উদাহরণ দুটি লক্ষ্য করুন–

```
*EX-1       Declare Array-Test [10]
*EX-2             Array-Test = Array (10)
```

উভয় ক্ষেত্রে Array-Test নামে ১০ এলিমেন্টের ১টি Array ডিক্লেয়ার করা হয়েছে। Array ডিক্লেয়ার করে নিয়ে আপনি এর প্রতিটি এলিমেন্টে আলাদা ভাবে মান সরবরাহ করতে পারেন। যেমন–

```
Array-Test [1] = "Finst Element"
Array-Test [10] = "At the End"
? array-Test [10] // Ans : at the End.
```

প্রকৃতপক্ষে ক্লিপারে Array খুবই গুরুত্বপূর্ণ একটি আইডেন্টিফায়ার। কেননা এর প্রায় লাইব্রেরী ফাংশনই Array ব্যবহার করে এবং পাওয়ার প্রোগ্রামিং ও এরে ব্যবহার করে সম্ভব।

Block ঃ ব্লকের ধারণাটি ক্লিপারে সম্পূর্ণ নতুন। এমনকি কোন ল্যাংগুয়েজেই এই বিষয়টি নেই, স্বভাবতই এর ধারনা কিছুটা জটিল। তবে কোড ব্লক মূলতঃ প্রোগ্রামে ম্যাক্রো ব্যবহারের বিকল্প পদ্ধতি হিসেবে যোগ করা হয়েছে। তাছাড়া এরে বা ডাটাবেজ রেকর্ড সবগুলো একসাথে প্রসেস করার ক্ষেত্রে DO WHILE এবং For-------NEXT লুপ স্ট্রাকচার অপেক্ষা Block ব্যবহার করে অনেক দ্রুত ফলাফল পাওয়া সম্ভব। এজন্য field block () এবং Fieldw block() নামক দুটি ফাংশন রয়েছে।

Block আসলে প্রোগ্রামের কিছু কোড, ফাংশন বা স্টেটমেন্ট ধারণ করে এবং যে কোন সময় তা EVULATE (প্রসেস) করতে পারে। তবে Block মূলত এডভান্স ক্লিপার প্রোগ্রামাররা ব্যবহার করেন।

OBJECT ঃ ক্লিপার ভার্সন ৫.০ হতে এতে যে অবজেক্ট ওরিয়েন্টেড প্রোগ্রামিং সাপোর্ট করার পদক্ষেপ নেয়া হয়েছে তার ফলশ্রুতিতে OBJECT টাইপের ভেরিয়েবল নামক একটি নতুন ভেরিয়েবল টাইপ উৎপন্ন হয়েছে।

যারা C++ প্রোগ্রামার তারা OBJECT-এর ধারণা জানেন। সংক্ষেপে OBJECT হল একটি ইউজার ডিফাইন্ড ভেরিয়েবল যা কিছু সম্পর্কযুক্ত ডাটা ধারণ করে এবং ঐ ডাটা প্রসেস করার জন্য কিছু কোড (সাধারণতঃ ফাংশন) ধারণ করে।

Tb column class, Tb browse class; Get class এবং Error class-ইত্যাদি বিল্টইন Class গুলোতে অবজেক্ট ওরিয়েন্টেড প্রোগ্রামিং ব্যবহার করা হয়েছে এবং এগুলো সাধারণত অবজেক্ট ব্যবহার করে। প্রোগ্রাম তৈরির ক্ষেত্রে Object-এর ধারণা এখনও দুর্বল। তাছাড়া অবজেক্ট ওরিয়েন্টেড প্রোগ্রামিংয়ের সাথে জড়িত POLYMORPHISM, INHERITANCE ইত্যাদি থিওরী এবং CLASS, INSTANCE VARIABLE এবং OBJECT ACCESS METHOD সংক্রান্ত কিছু অতিপ্রয়োজনীয় Tool এর অভাব রয়েছে। তবে এটুকু আশা করা যায় ভবিষ্যতে এ বিষয়গুলোর দিকে Nantucket দৃষ্টি দেবে। আপাততঃ OBJECT-এর ধারণাটি এডভান্স প্রোগ্রামিংয়ে প্রয়োগ করা হয়।

ভেরিয়েবল স্কোপ, লাইফটাইম এবং এসাইনমেন্ট ঃ ভেরিয়েবলের স্কোপ বলতে বুঝায় কোন ভেরিয়েবল প্রোগ্রামের কোন্ কোন্ অংশে Visible

(দৃশ্যমান) হবে বা ঐ ভেরিয়েবল প্রসেস করা যাবে। আর লাইফ টাইম বলতে বুঝায় কোন ভেরিয়েবলে স্টোর করা Value কত সময় (অর্থাৎ প্রোগ্রামের এক অংশ হতে অন্য অংশে কল করে যেমন DO কমান্ড বা ইউজার ডিফাইন্ড ফাংশন কল হবার পর আবার মেইন অংশে ফিরে আসা) স্থায়ী হবে। এ দুটি ধারণা ডিবেজ ব্যবহারকারীদের অনেকের কাছেই নতুন। কেননা তারা শুধু মাত্র ভেরিয়েবলটি ব্যবহার করেন, এর স্কোপ বা লাইফটাইমস্ নিয়ে কিছুই চিন্তা করেননা এবং এই চিন্তার ভার ছেড়ে দেন সিস্টেমের উপর। কিন্তু এ ধরণের প্রোগ্রামিং প্রকৃত অর্থে "Sloppy" এবং অনেক Error সৃষ্টির সহায়ক।

ক্লিপারের সকল ভেরিয়েবল ব্যবহারের পূর্বে অর্থাৎ এতে Value এসাইন করার ক্ষেত্রে বা Retrive করার পূর্বে আমাদের এর স্কোপ এবং লাইফটাইম স্থির করে নিতে হবে। এ কাজটি আমরা ভেরিয়েবল ডিক্লেয়ার করার সময় নির্ধারণ করে দেই। ক্লিপারে মোট চার প্রকারের ভেরিয়েবল ডিক্লেয়ার করা যায়। LOCAL STATIC, PUBLIC এবং PRIVATE. এখানে আমরা শুধু LOCAL এবং STATIC ডিক্লারেশন নিয়ে আলোচনা করব।

LOCAL টাইপ মডিফায়ার দ্বারা কোন ভেরিয়েবলকে এমন রূপে স্কোপ দেয়া যায় যার ফলে ঐ ভেরিয়েবলটি কেবলমাত্র যে প্রোগ্রামে বা যে ফাংশনে ডিক্লেয়ার করা হয়েছে সেখানে এক্সেস করা যায়। ঐ প্রোগ্রাম বা ফাংশনের বাইরে অন্য কোন ফাংশন বা প্রোগ্রাম ঐ ভেরিয়েবল এক্সেস করতে পারবেনা। আবার একই ভাবে যখনই ঐ PRG বা ফাংশন তার কাজ শেষ করে মূল ফাংশন/প্রোগ্রাম কন্ট্রোল রিটার্ন করবে তৎক্ষণাৎ ঐ ভেরিয়েবেল লাইফটাইম শেষ হয়ে যাবে অর্থাৎ এতে STORE করা সমস্ত ডাটা NIL হয়ে যাবে। নীচের উদাহরণটি দেখুন–

```
* File :- Variable PRG
* Note : Test Variable Scope and life time
LOCAL Name              // Declare Name.
name : = "My name" // Assign Character
Value
Waiting g()          // Call function waiting
?Name
Function Waiting ()
Local i
For i = 1 to 10
? i
NEXT
Return NIL
** End of PRG.
```

এখানে name ভেরিয়েবলটি Waiting() ফাংশন থেকে এক্সেস করা যাবে না। আবার Waiting() ফাংশনে ডিক্লেয়ার করা ভেরিয়েবল i মূল প্রোগ্রাম ফাইল হতে এক্সেস করা যাবে না।

LOCAL ভেরিয়েবল ব্যবহার করার সুবিধা হল যখন আপনার প্রোগ্রাম অনেক জটিল ও দীর্ঘ হবে এবং এতে ব্যবহৃত প্রসিডিউর বা UDF অনেক হবে তখনও আপনি নিশ্চিন্তে ভেরিয়েবল ব্যবহার করতে পারেন। এতে করে ঐ ভেরিয়েবল কেবলমাত্র নির্দিষ্ট ফাংশনেই সীমাবদ্ধ থাকবে। আবার মেইন ফাংশন বা প্রোগ্রামের এমন কিছু ডাটা যা আপনি এর কোন প্রসিডিউর বা ফাংশনকে এক্সেস করতে দিতে চাননা তাও LOCAL হিসেবে ডিক্লেয়ার করে নিতে পারেন। পরিশেষে এপ্লিকেশনটি এমন কম্প্যাক্ট হবে যে ভুলবশতঃ একটি ভেরিয়েবলের Assignment দ্বারা অন্য ভেরিয়েবলগুলো ওভাররাইট হবার ভয় থাকবে না। দ্বিতীয়তঃ প্রোগ্রামের ঠিক যে অংশে ভেরিয়েবলের প্রয়োজন ও ব্যবহার সীমাবদ্ধ থাকবে ভেরিয়েবলটি ঠিক যেখানেই Visible এবং স্থায়ী হবে। এতে মেমোরী ব্যবহারও সম্পূর্ণ উপযোগী হবে। উপরন্তু একই নাম আপনি একাধিক স্থানে ব্যবহার করতে পারবেন।

STATIC ভেরিয়েবল ঃ STATIC ভেরিয়েবলের প্রকৃতিতে একই সাথে PUBLIC, PRIVATE এবং LOCAL ভেরিয়েবলের স্বাভাব অর্থাৎ স্কোপ ও লাইফটাইম জড়িত রয়েছে। এর তিন ধরনের ব্যবহার রয়েছে।

i) EXTERNAL STATIC ঃ এই পদ্ধতিতে ভেরিয়েবলের স্কোপ এবং লাইফটাইম অনেকটা PBULIC ভেরিয়েবলের মত হয়। প্রোগ্রামের শুরুতে (কোন কোড শুরুর পূর্বে) STATIC লিখে এই শ্রেণীর ভেরিয়েবলের ডিক্লেয়ার করে নিতে হয়।

ii) Internal STATIC ঃ এই ধরনের STATIC ভেরিয়েবলে ব্যবহার খুব কম করা হয়।

iii) যখন প্রসিডিউর বা ফাংশনের জন্য আলাদা PRG ফাইল তৈরি করা হয় এবং ঐ PRG ফাইলের শুরুতে STATIC ভেরিয়েবল ডিক্লেয়ার করা হয় তখন তাকে Module STATIC বলা হয়। এর SCOPE LOCAL ভেরিয়েবলের মত তবে Lifetime পাবলিক ভেরিয়েবলের মত।

অর্থাৎ যে প্রোগ্রাম ফাইলে (এই ফাইলে এপ্লিকেশনের কোন সোর্স কোড থাকবে না, কেবল এপ্লিকেশনে ব্যবহার করা হয়েছে এমন UDF বা প্রসিডিউর থাকবে) এই STATIC ভেরিয়েবলটি ডিক্লেয়ার করা হয়েছে, ঐ ফাইলের অন্তর্গত ফাংশন বা প্রসিডিউর ছাড়া অন্য কোন ফাংশন, প্রসিডিউর এমনকি মেইন রুটিনও প্রসেস করতে পারবে না।

আর Public (যাকে আমরা Global বলে থাকি) লাইফ টাইমের অর্থ হল MODULE ভেরিয়েবলে যদি কোন Value এসাইন করা হয়, তা পরিবর্তন না করলে প্রোগ্রামের Quit হওয়া পর্যন্ত এই Value টিকে থাকে। নীচের উদাহরণটি লক্ষ্য করুন–

```
*File
*File 1 :- STATTEST. PRG
*Note : - Test static variable's scope and
lifetime.
*    compile with:- CLIPPER STATTEST
W/A/M/N
*    Link with :- RTLINK F1 stattest,
module
STATIC prg variable: = 1 // Declare an
external
? prg-variable // (Some times called :- file
wide)
    // variable
?"static test-----calling stat_test()"
stat_test()   //Call stat-test() for 1st time
?prg-variable          //See          the
difference?
stat-test()   // See the diffrence of i
    // (in module. PRG File)
* End of PRG File-1 <STATTEST. PRG>
*FILE 2 : - MODULE, PRG
*Compile with /W/A/M/N switch.
*Note that any program using
*Static variable (S) must be compiled
with
*/N switch.
STATIC i : = 1 //  Module static - i
Function stat-test () // UDF stat _test ()
i++; PRG-variable++ // same as i = i + 1
?i   // prg-variable = prg +1
RETURN NIL           // Returns no value
(NIL)
*End of prg File 2 <MODULE, PRG>
```

UDF ঃ ইউ, ডি, এফ বা ইউজার ডিফাইন্ড ফাংশন হচ্ছে ক্লিপারের এমন একটি ফীচার যার মাধ্যমে অন্যান্য হাইলেভেল ল্যাংগুয়েজের মত এতে স্ট্রাকচার্ড ল্যাংগুয়েজের ধারণা সুস্পষ্ট করা হয়েছে। তাত্ত্বিকভাবে একটি ফাংশনের তিনটি অংশ রয়েছে। ১) ফাংশন ডিক্লেয়ারেশন ও প্যারামিটার লিস্ট ২) ফাংশন কোড ৩) রিটার্ন স্টেটমেন্ট (রিটার্ন Value সহ, যদি থাকে, না হলে NIL)।

ফাংশন ডিক্লেয়ারেশন করা যায় FUNCTION-এ কমান্ডটি ও ফাংশনের নাম দ্বারা এবং প্রথম বন্ধনীর ভিতর যে সকল প্যারামিটার ফাংশনে প্রয়োজন হবে তার উল্লেখ করতে হয়। প্যারামিটার বলতে বুঝায় এমন একটি বা কিছু ভেরিয়েবল বা এরে যার উপর ভিত্তি করে ফাংশনটি কাজ করবে। অনেক ল্যাংগুয়েজে একে ফাংশন প্যারামিটার বলা হয়। উদাহরণটি লক্ষ্য করুন–

```
FUNCTION my function (name, city)
@ 10,10 say name
@ 11,10 say City
RETURN NIL
```

উদাহরণের প্রথম লাইন দ্বারা My function নামে একটি UDF ডিক্লেয়ার করা হয়েছে। এর দুইটি প্যারামিটার হচ্ছে name ও city নামক দুইটি ভেরিয়েবল যা ফাংশনটির নিজস্ব।

ফাংশন কোড হচ্ছে ফাংশনের ডিক্লেয়ারেশন ও রিটার্ন স্টেটমেন্টের পূর্ব পর্যন্ত যে সমস্ত কাজ ফাংশনটি করবে তা অর্থাৎ ফাংশনের ACTION. রিটার্ন স্টেটমেন্টটি দ্বারা ফাংশনের একই সাথে দুইটি কাজ হয়। প্রথমতঃ এটি ফাংশনের শেষ সীমা নির্দেশ করে এবং দ্বিতীয়তঃ যদি কোন Value রিটার্ন করার প্রয়োজন হয় তবে ফাংশনের শেষের সাথে সাথে তা রিটার্ন করে। উপরের উদাহরণের my-function() ফাংশনটি কোন Value রিটার্ন করেনা বলে আমরা এখানে NIL শব্দটি দিয়েছি।

ফাংশন হচ্ছে ক্লিপারের সর্বাপেক্ষা গুরুত্বপূর্ণ একটি এলিমেন্ট। ক্লিপারের সমস্ত শক্তি তার ফাংশন সেটে এবং প্রোগ্রামারদের স্বাচ্ছন্দ্যও এখানেই। তাই ফাংশন ব্যবহারের সাথে প্রত্যেক ক্লিপার প্রোগ্রামারকে সতর্ক থাকতে হয়। এখানে আমরা UDF এর ডিক্লেয়ারেশন-এর অভ্যন্তরীণ প্রক্রিয়া এবং ব্যবহার নিয়ে কিছু গুরুত্বপূর্ণ তথ্য এবং উদাহরণ দেব।

শুরুতে আপনাকে এটা জানতে হবে যে, আপনি আপনার প্রোগ্রামে পুরো কোডই ফাংশনের আওতায় নিয়ে আসতে পারেন। এ ক্ষেত্রে আপনার এপ্লিকেশনটি একটি বড় UDF-এ পরিণত হবে। আবার প্রোগ্রামে একাধিকবার প্রয়োজন হয় এমন কিছু কোড আপনি ফাংশনের আওতায় নিয়ে যেতে পারেন। অনেকটা ডিবেজের প্রসিডিউরের মত। তবে UDF এর মূল সুবিধা হচ্ছে এটি প্রোগ্রামের যে কোন স্থান হতে কল করা যায়। কোন DO কমান্ডের প্রয়োজন হয় না। তাছাড়া প্রতিটি UDF আলাদা PRG ফাইলে তৈরি করে এবং কম্পাইল করে তাদের OBJ মডিউল একাধিকবার অন্যান্য প্রোগ্রামের সাথে ব্যবহার করা

যায়। (দ্বিতীয় পর্বে একই সাথে অনেকগুলো OBJ লিংক করার পদ্ধতি আলোচিত হয়েছে) এমনকি কোন লাইব্রেরী ম্যানেজার (যেমন মাইক্রোসফটের LIB. EXE) ব্যবহার করে লাইব্রেরী ফাইলে UDF সমূহ ইনক্লুড করে ক্লিপারের অন্যান্য ইন্টারনাল ফাংশনের মত ব্যবহার করা যায়।

ফাংশন প্যারামিটার ও রিটার্ন ভেল্যু ঃ UDF-এ কিভাবে প্যারামিটার ভেরিয়েবল ডিক্লেয়ার করতে হয় তা আমরা দেখেছি। এখন আমরা কিভাবে প্যারামিটার সাপ্লাই করতে হয় এবং ক্লিপারের প্যারামিটার হ্যান্ডলিং টেকনিক সম্বন্ধে জানব। এটি বেশ সহজ, ডিবেজে যে ভাবে আপনি লাইব্রেরী ফাংশন কল করেন ঠিক সে ভাবেই আপনি UDF কল করতে পারেন, যেমন–

```
LOCAL number:=100
?
LOCAL multi:=5
?multiple (number, multi)
Function multiple (P1, P2)
LOCAL return_Val:=P 1* P2
Return return_Val
```

আবার রিটার্ন Value'র ক্ষেত্রে ফাংশনের অভ্যন্তরে কোন LOCAL ভেরিয়েবলে ফাংশনের ফলাফল জমা করে RETURN স্টেটমেন্টে ঐ ভেরিয়েবল রিটার্ন করতে পারেন।

বাড়তি যা আপনার জানতে হবে তা হল ক্লিপার দুইভাবে UDF–এ প্যারামিটার সরবরাহ করতে দেয়। প্রথমটি হচ্ছে PASS BY VALUE (ডিফল্ট নিয়ম) পরেরটি PASS BY REFERENCE. প্রথমোক্ত পদ্ধতিতে ফাংশনে যে প্যারামিটার ডিক্লেয়ার করা হয় ক্লিপার প্রতিটি প্যারামিটারের জন্য আলাদা LOCAL ভেরিয়েবল তৈরি করে। পরবর্তীতে আপনি যখন Real value সরবরাহ করে ফাংশনটি কল করেন তখন ক্লিপার Real variable -এর Value গুলোর একটি করে প্রতিলিপি (Copy) ফাংশনটিকে সরবরাহ করে। এতে করে আপনার অরিজিনাল ভেরিয়েবল পরিবর্তিত হবার কোন ভয় থাকে না। দ্বিতীয় পদ্ধতিতে প্যারামিটারে সরবরাহকৃত real variable (আলোচ্য উদাহরণের number ও multi) গুলো ফাংশনে সরবরাহ করা হলে ফাংশন তা পরিবর্তন করতে পারে। যেমন ঃ

```
LOCAL number, multi // Declare only,
donot assign
number :=100 // Assign 100 to number
multi:= 10    // Assign 10 to multi
multiple (@number, multi) // Call function
multi()
?number    // Ans:- 1000
Function multiple (P1, P2)
P1=P1 * P2
Return P1 // It would also do if we write
//Return nil
```

উপরের উদাহরণে multiple() ফাংশনটির number প্যারামিটারকে BY REFERENCE-এ কল করা হয়েছে বিধায় multiple() ফাংশনের অভ্যন্তরে যখন P1=P1 * P2 কমান্ড দেয়া হয়েছে তখন আপনা আপনি Number ভেরিয়েবলটিও পরিবর্তিত হয়ে গেছে। ফাংশনের যে প্যারামিটার BY REFERENCE-এ কল করতে হয় তার পূর্বে "@"–এই চিহ্ন (At the rate) দিয়ে নিতে হয়। আসলে যখন কোন ফাংশন প্যারামিটার BY REFERENCE-এ কল করা হয় তখন ক্লিপার ফাংশনটিতে প্রকৃত ভেরিয়েবলের জন্য যে মেমোরী এড্রেস ব্যবহৃত হয়েছে তার প্রতি একটা পয়েন্টার রিটার্ন করে। এতে ফাংশনটি ঐ প্যারামিটারটি মডিফাই করতে পারে।

এখানে পাঠকরা লক্ষ্য করবেন যে, আমরা LOCAL স্টেটমেন্টটি দ্বারা ভেরিয়েবলকে ডিক্লেয়ার করে একই সময়ে তাতে মান সরবরাহ করেছি আবার এসাইনমেন্টের জন্য সাধারণ "=" চিহ্ন ব্যবহার না করে ":=" ব্যবহার করেছি। একে বলে Inline Assignement operator. ক্লিপারে সাধারণত "=" চিহ্ন ব্যবহার করা উচিত নয়। তবে কন্ডিশন্যাল চেক (যেমন :- IF, ELSE, DO WHILE) করার সময় অবশ্যই ":=" ব্যবহার করা যাবে না।

ম্যাক্রো, ফাংশন কোড ব্লক এবং কমান্ডঃ

ম্যাক্রো ব্যবহারের সাথে ডিবেজ ব্যবহারকারীরা বহুল পরিচিত। সরাসরি SET FILTER TO এর এক্সপ্রেশনে হতে শুরু করে ক্ষুদ্র RUN কমান্ড পর্যন্ত সর্বদাই ম্যাক্রো অপারেটর "&"-কে দেখা যায়। ম্যাক্রো ব্যবহার করা যত সহজ এর ইন্টারন্যাল ইমপ্লিমেন্ট কিন্তু অনেক জটিল। আমরা সাধারণতঃ ম্যাক্রো ব্যবহার করি যে সমস্ত ক্ষেত্রে (কমান্ডে) literal এক্সপ্রেশন সরবরাহ করতে হয়। দেখা যায় প্রায় কমান্ডেই literal ব্যবহার করতে হয় (যেমনঃ- বহুল ব্যবহৃত USE, SET INDEX TO, COPY ইত্যাদি)। সাধারণ দৃষ্টিতে এটি কোন সমস্যা নয়। কিন্তু একজন এপ্লিকেশন ব্যবহারকারী যখন নিজ পছন্দ মত ডাটাবেজ ব্যবহার করতে চাইবেন কিংবা ফাইল creat করতে চাইবেন তখন প্রোগ্রামারকে ম্যাক্রো ব্যবহার করে সে সুযোগ সৃষ্টি করে দিতে হবে।

অধিকাংশ প্রোগ্রামার এ ক্ষেত্রে যা করেন তা হল Get ইস্যু বা অন্য কোন পদ্ধতিতে ব্যবহারকারী হতে কোন স্ট্রীং গ্রহণ করেন এবং কমাণ্ডে ব্যবহারের সময় ঐ স্ট্রীং (সাধারণতঃ বেরিয়েবল) এর পূর্বে "&" ম্যাক্রো চিহ্নটি বসিয়ে দেন, ম্যাক্রো তখন ঐ ভেরিয়েবলের ক্যারেক্টার স্ট্রীংকে (যাকে কোটেশন স্ট্রীং বলা যায়) লিটারাল স্ট্রীং (যাকে কোটেশন বিহীন স্ট্রীং বলা যুক্তিযুক্ত) এ পরিণত করে এবং কমাণ্ড লাইনে সরবরাহ করে। যেমন "Karim"-এর পূর্বে "&" ম্যাক্রো ব্যবহৃত হলে এটি হয়ে যাবে Karim .

ম্যাক্রো হ্যান্ডেলিং করার জন্য ক্লিপারে যে রুটিন রয়েছে তা আয়তনে বেশ দীর্ঘ। তাছাড়া যে এক্সপ্রেশন এটি ধারণ করে তা মোটেও কম্পাইলকৃত এক্সপ্রেশন নয় এবং ক্লিপারের কম্পাইলার একে অপটিমাইজ করতে পারে না। ম্যাক্রো এক্সপ্রেশনের সমস্তটুকু ম্যানেজমেন্ট সম্পন্ন করা হয় রান টাইমে, আপনার প্রোগ্রাম চলাকালীন।

সুতরাং দেখা যাচ্ছে, ম্যাক্রো ইমপ্লিমেন্ট করার খাতিরে আপনার SAEF-এর সাথে বাড়তি কিছু কোড যোগ হবে, তাতে একটি মিনি কম্পাইলার এবং লিংকারের আচরণক্ষম একটি ছোট (!) এনালাইজারও থাকতে হবে সুতরাং বুঝতেই পারছেন একটি EXE ফাইলের কম্পাক্টনেস্ কতটুকু নষ্ট হয়ে যেতে পারে একটি মাত্র ম্যাক্রো অপারেটর ব্যবহারের কারণে।

ক্লিপারে ম্যাক্রো ব্যবহার দূর করার জন্য কোড ব্লক নামক নূতন একটি ডাটা টাইপ যুক্ত করা হয়েছে যদিও এর ধারণা কিছুটা জটিল, বাকি যে সহজ এবং বেশ আকর্ষণীয় পদ্ধতি রয়েছে তা হল কমাণ্ডের পরিবর্তে ফাংশন ব্যবহার। শুনলে হয়ত অবাক হবেন যে ক্লিপারের যাবতীয় কমান্ডই আসলে এক ধরনের কম্পাইলার ডাইরেকটিভ এবং কিছু Translation স্টেটমেন্ট। ঐ কমাণ্ডগুলো আসলে নির্দিষ্ট ফাংশনকে কল করে। যেমন SET INDEX ON কমাণ্ডটি - dbcreatIndex ( ) নামক একটি ফাংশন' কল করে নির্দিষ্ট প্যারামিটারসহ। এই সমস্ত কমান্ড যখন কম্পাইল করা হয় তখন Parser দ্বারা ফাংশন কলে পরিণত হয়। এই পরিবর্তন আপনি ইচ্ছা করলে দেখতে পারেন কম্পাইলারে /P সুইচটির ব্যবহার করে। এই সুইচের ব্যবহারের ফলে আপনার PRG ফাইলের একটি OBJ কপি তৈরী হওয়ার পাশাপাশি PPO (যার অর্থ PRE-PROCESSOR'S OUTPUT) এক্সটেনশনের একটি ফাইল তৈরী হবে। ঐ ফাইলটি VIEW করে আপনি বুঝতে পারবেন কোন্ কমাণ্ড কোন্ ফাংশন কল করে। এটি যখন আপনি বুঝতে পারবেন তখন প্রোগ্রামে কমাণ্ড ব্যবহার না করে সরাসরি ফাংশন ব্যবহার করতে পারেন।

এখন দেখা যাক কিভাবে ফাংশন ব্যবহার করে ম্যাক্রো ব্যবহার দূর করা যায়। ক্লিপারের কমান্ডগুলোর জন্যই ম্যাক্রো দরকার পড়ে। এবং এই কমান্ডগুলো STD.CH নামক ক্লিপারের হেডার ফাইলে ডিফাইন করা হয়েছে। প্রকৃতপক্ষে এ সমস্ত কমাণ্ড যদিও literal value প্যারামিটার হিসাবে গ্রহণ করে তথাপি Rightside Match মার্কারের সাহায্যে ক্লিপার ঐ literal কে character স্ট্রীং-এ পরিণত করে এবং নির্ধারিত ফাংশনকে ঐ স্ট্রীং প্যারামিটার হিসাবে সরবরাহ করে। তাই আপনিও ঐ ফাংশন কল করার সময় literal ব্যবহার না করে সরাসরি স্ট্রীং কিংবা মেমোরী ভেরিয়েবল ব্যবহার করতে পারেন। নীচের উদাহরণ দুটি দেখুন ঃ -

```
fname : = "Prince"        fname = "Prince"
Run & fname               — — Run (fname)
"Use of macro             * Direct library function call
                          for "Run" command.
Example-1                 Example-2
```

(সমাপ্ত)

হিউলেট প্যাকার্ড

(২৬ নং পৃষ্ঠার পর)

এর ব্যবস্থাপনার দর্শন এবং মূল্যবোধ। সিলিকন ভ্যালী যেখানে দম্ভ আর ঔদ্ধত্য সাধারণ ঘটনা সেখানে কাজের প্রতি নিবেদিত প্রান এইচপির কর্মীরা অসাধারণভাবে ভদ্র, নম্র আর বিনয়ী। লোকে এদেরকে ডাকে 'বয় স্কাউট' নামে। অবসর গ্রহণ করলেও এখনও উইলিয়াম হিউলেট এবং ডেভিড প্যাকার্ড কোম্পানীতে আসেন কর্মীদের খোঁজখবর নেন। কোম্পানীর প্রতিটি কর্মীর মধ্যে রয়েছে সৌহার্দাপূর্ণ সম্পর্ক যা কোম্পানীর কাজের গতিতে এনেছে সমৃদ্ধি। এখানে কাজের জন্য কাউকে বলতে হয় না। কর্মীরা কোম্পানীকে নিজের মনে করে কাজ করেন। এ প্রসঙ্গে প্রধান নির্বাহী লুইস প্লাট বলেন, 'প্রধান নির্বাহী হিসেবে আমার কাজ হলো সৌহার্দাপূণ পরিবেশে কাজ করতে সকলকে উৎসাহিত করা। এখানে কাজের জন্য বা কি করতে হবে বলতে হয় না কারণ যারা এখানে আসে কিংবা আছে তারা নিজেরাই কাজ ঠিক করে নেয় এবং কোম্পানীর সমৃদ্ধির লক্ষ্যে কঠোর পরিশ্রম করে। কারণ তারা জানে এবং বোঝে প্রতিষ্ঠানের সমৃদ্ধিই তাদের সমৃদ্ধি।' এভাবেই এইচপি আপন গতিতে সমৃদ্ধির পথে এগুচ্ছে এবং দিনে দিনে অন্যদের ছাড়িয়ে যাচ্ছে। ●