این بخش کد که در زبان برنامه نویسی C++ نوشته شده، شامل تعریف یک تابع با نام `explorePhase` است که دو بردار (vector) از ساختارهای داده (که در اینجا به عنوان `VM` و `Server` تعریف شده‌اند) را به عنوان پارامتر دریافت می‌کند. این کد می‌تواند بخشی از یک الگوریتم بهینه‌سازی باشد که برای پیکربندی ماشین‌های مجازی (VMs) روی سرورها (servers) به کار می‌رود.

من در ادامه هر خط از کد را به زبان فارسی توضیح می‌دهم:

```plaintext

void explorePhase(const vector<VM>& vms, const vector<Server>& servers) {

```

- این خط شروع تعریف تابع `explorePhase` است. تابع هیچ مقداری باز نمی‌گرداند (`void`) و دو پارامتر به صورت آرایه‌های قابل تغییر (`const vector`) دریافت می‌کند: یکی از نوع `VM` و دیگری از نوع `Server`.

```plaintext

for(auto& solution : population) {

```

- این خط یک حلقه `for` را شروع می‌کند که روی هر عنصر از مجموعه‌ای به نام `population` اجرا می‌شود. `auto&` اینجا به معنای این است که برای هر عنصر از `population`، ما یک مرجع (reference) داریم که به نام `solution` است.

```plaintext

double fit = solution.fitness;

```

- در این خط، یک متغیر جدید به نام `fit` ایجاد می‌شود که مقدار `fitness` از هر `solution` (حل) فعلی را در خود نگه می‌دارد.

```plaintext

for(int i = 0; i < solution.placements.size(); ++i) {

```

- این خط شروع یک حلقه داخلی `for` است که بر روی بردار `placements` از هر `solution` اجرا می‌شود.

```plaintext

vector<int> neighbors;

```

- اینجا، یک بردار جدید از اعداد صحیح به نام `neighbors` ایجاد می‌شود که برای نگه داری شاخص‌های همسایه‌ها به کار می‌رود.

```plaintext

for(int j = 0; j < servers.size(); ++j) {

```

- این خط شروع یک حلقه `for` دیگر است که بر روی همه سرورها اجرا می‌شود.

```plaintext

if(j != solution.placements[i]) {

```

- این شرط بررسی می‌کند که آیا شاخص `j` متفاوت از شاخص جایگذاری فعلی در `solution.placements[i]` است یا خیر.

```plaintext

neighbors.push\_back(j);

```

- اگر شرط بالا درست باشد، شاخص `j` به بردار `neighbors` افزوده می‌شود.

```plaintext

}

```

- پایان بلوک شرطی.

```plaintext

if(!neighbors.empty()) {

```

- این شرط بررسی می‌کند که آیا بردار `neighbors` خالی نیست.

```plaintext

int index = generator() % neighbors.size();

```

- یک تابع `generator` فراخوانی می‌‌شود که یک شماره تصادفی تولید می‌کند و این شماره با اندازه `neighbors` تقسیم می‌شود تا یک `index` تصادفی از میان همسایه‌ها انتخاب شود.

```plaintext

int neighbor = neighbors[index];

```

- شاخص تصادفی انتخاب شده در خط بالا برای گرفتن یک عنصر از بردار `neighbors` به کار می‌رود که به `neighbor` اختصاص داده می‌شود.

```plaintext

solution.placements[i] = neighbor;

```

- جایگذاری فعلی در `solution` با `neighbor` جدید جایگزین می‌شود.

```plaintext

}

}

}

```

- این خطوط پایان حلقه‌ها و تابع را نشان می‌دهند.

این بخش کد شامل دو تابع در زبان برنامه‌نویسی C++ است که به نظر می‌رسد جزو یک الگوریتم بهینه‌سازی از نوع مشابه به الگوریتم‌های جمعیتی باشد که برای موقعیت‌یابی بهینه ماشین‌های مجازی (VMs) بر روی سرورها (Servers) استفاده می‌شود. در زیر هر خط کد به زبان فارسی توضیح داده شده است:

\*\*تابع اول: exploitPhase\*\*

```plaintext

void exploitPhase(const vector<VM>& vms, const vector<Server>& servers) {

```

این تابع، تابعی به نام `exploitPhase` است که دو پارامتر بردار ورودی غیرقابل تغییر از نوع `VM` و `Server` را دریافت می‌کند.

```plaintext

for (auto& solution : population) {

```

این دستور یک حلقه فوریچ است که روی هر عضو از مجموعه‌ای به نام `population` عملیات انجام می‌دهد.

```plaintext

for (int i = 0; i < solution.placements.size(); ++i) {

```

این حلقه بر روی تمامی مقادیر موجود در بردار `placements` از هر `solution` اجرا می‌شود.

```plaintext

auto& placement = solution.placements[i];

```

این خط یک مرجع (reference) به هر مقدار در بردار `placements` به نام `placement` ایجاد می‌کند.

```plaintext

if (placement >= 0 && placement < servers.size()) {

```

این شرط بررسی می‌کند که آیا `placement` معتبر است (یعنی در بازه صحیحی از اندیس‌های `servers` قرار دارد).

```plaintext

Server server = servers[placement];

```

این خط یک نسخه از سرور مشخص شده توسط `placement` ایجاد می‌کند.

```plaintext

vector<int> bestServers;

```

این خط یک بردار برای نگهداری بهترین سرورها (به لحاظ انتخابی که بعدا تعیین می‌شود) می‌سازد.

```plaintext

for (int j = 0; j < servers.size(); ++j) {

```

این حلقه برای بررسی تمامی سرورها به جهت پیدا کردن گزینه‌های بهتر استفاده می‌شود.

```plaintext

if (j != placement) {

```

این شرط از مقایسه سرور فعلی با سروری که هم اکنون VM روی آن است جلوگیری می‌کند.

```plaintext

Server neighborServer = servers[j];

```

این خط یک نسخه از سرور جاری ایجاد می‌کند تا در مقایسه با دیگران استفاده شود.

```plaintext

if (neighborServer.cpu >= vms[i].cpu && neighborServer.memory >= vms[i].memory && neighborServer.storage >= vms[i].storage) {

```

این شرط بررسی می‌کند که آیا سرورهای مجاور منابع کافی برای نیازهای ماشین مجازی را دارند.

```plaintext

bestServers.push\_back(j);

```

این خط اندیس سروری که معیارها را برآورده می‌کند به بردار `bestServers` اضافه می‌کند.

```plaintext

}

```

این خط پایان بلوک شرطی داخلی است.

```plaintext

if (!bestServers.empty()) {

```

این شرط بررسی می‌کند که آیا انتخاب‌های بهتری وجود دارند.

```plaintext

int index = static\_cast<int>(generator() % bestServers.size());

```

این خط یک اندیس تصادفی از بردار `bestServers` انتخاب می‌کند.

```plaintext

placement = bestServers[index];

```

این خط `placement` فعلی را به بهترین گزینه موجود تغییر می‌دهد.

```plaintext

}

```

این خط پایان بلوک شرطی بالایی است.

```plaintext

solution.fitness = solution.calculateFitness(vms, servers);

```

این خط مقدار برازندگی (`fitness`) هر راه حل را بر اساس یک تابع محاسبه می‌کند که به احتمال زیاد منابع مصرفی و تطابق آن با سرورها را در نظر می‌گیرد.

```plaintext

}

```

این خط پایان تابع `exploitPhase` را نشان می‌دهد.

\*\*تابع دوم: attackPhase\*\*

```plaintext

void attackPhase(const vector<VM>& vms, const vector<Server>& servers) {

```

این تابع، تابعی به نام `attackPhase` است که مشابه تابع قبلی دو بردار ورودی را به عنوان پارامتر می‌پذیرد.

```plaintext

vector<PlacementSolution> sortedPopulation = population;

```

این خط یک نسخه از مجموعه `population` را می‌سازد تا به ترتیب ببرد.

```plaintext

sort(sortedPopulation.begin(), sortedPopulation.end(),

[](const PlacementSolution& a, const PlacementSolution& b) {

return a.fitness > b.fitness;

});

```

این خط بردار `sortedPopulation` را بر اساس مقدار `fitness` هر عضو از زیاد به کم مرتب می‌کند.

```plaintext

for (auto& solution : population) {

```

حلقه فوریچ برای هر عضو در `population` تکرار می‌شود.

```plaintext

uniform\_real\_distribution<double> dist(0.0, 1.0);

```

یک توزیع یکنواخت برای اعداد حقیقی بین 0 و 1 ایجاد می‌کند.

```plaintext

double moveProb = dist(generator);

```

این خط یک عدد تصادفی بر اساس توزیع بالا گرفته و به عنوان احتمال حرکت در نظر می‌گیرد.

```plaintext

if (moveProb > 0.5) {

```

این شرط بررسی می‌کند که آیا احتمال حرکت بیشتر از 0.5 است.

```plaintext

int currentVM = rand() % vms.size();

```

یک شماره تصادفی (اندیس VM) بر اساس تعداد VMها انتخاب می‌شود.

```plaintext

int currentServer = solution.placements[currentVM];

```

سرور فعلی برای VM انتخابی تعیین می‌شود.

```plaintext

double p = dist(generator);

```

یک عدد تصادفی دیگر برای تصمیم‌گیری بعدی ایجاد می‌کند.

```plaintext

if (p > 0.5) {

```

بررسی می‌کند آیا عدد تصادفی بیشتر از 0.5 است.

```plaintext

// Move towards the global best

solution.placements[currentVM] = bestSolution.placements[currentVM];

```

اگر بالاتر از 0.5 بود، جایگاه VM را به جایگاه بهترین راه حل جهانی حرکت می‌دهد.

```plaintext

} else {

```

در غیر این صورت، یک جایگاه شخصی بهتر را جستجو می‌کند.

```plaintext

// Move towards the personal best

int personalBestServer = -1;

double personalBestFitness = 0.0;

```

متغیرهایی برای ذخیره بهترین جایگاه شخصی و برازندگی آن تعریف می‌شود.

```plaintext

for (int j = 0; j < sortedPopulation.size(); ++j) {

```

برای تعیین بهترین جایگاه شخصی، یک حلقه از جمعیت مرتب شده اجرا می‌شود.

```plaintext

if (sortedPopulation[j].fitness > personalBestFitness) {

```

در صورتی که فیتنس یکی از افراد جمعیت از فیتنس شخصی بهتر باشد، بررسی می‌شود.

```plaintext

personalBestFitness = sortedPopulation[j].fitness;

personalBestServer = sortedPopulation[j].placements[currentVM];

```

فیتنس شخصی بهتر و جایگاه مربوطه به‌روزرسانی می‌شود.

```plaintext

}

```

پایان بلوک شرطی.

```plaintext

solution.placements[currentVM] = personalBestServer;

```

جایگاه VM به بهترین جایگاه شخصی تغییر پیدا می‌کند.

```plaintext

}

```

پایان بلوک شرطی.

```plaintext

solution.fitness = solution.calculateFitness(vms, servers);

```

فیتنس راه‌حل بعد از تغییر جایگاه به‌روزرسانی می‌شود.

```plaintext

}

```

پایان حلقه فوریچ.

```plaintext

}

```

پایان تابع `attackPhase`.