معناشناسی عملیاتی یک روش از معناشناسی زبان‌های برنامه‌نویسی است که به‌صورت قاعده‌مند نحوه اجرای یک برنامه را با استفاده از گام‌های محاسباتی توصیف می‌کند. در این روش، می‌توان از زبان دیگری مانند C برای نشان دادن گام‌های محاسباتی استفاده کرد.

- ساختار SELECT:

این ساختار برای انتخاب داده‌های موردنظر از یک یا چند جدول استفاده می‌شود. معناشناسی عملیاتی این ساختار را می‌توان در زبان C به‌صورت یک تابع که یک پارامتر ورودی به‌عنوان شرط و یک پارامتر خروجی به‌عنوان نتیجه دارد پیاده‌سازی کرد.

برای مثال، ساختار زیر را می‌توان به‌صورت زیر توصیف کرد:

```
SELECT * FROM students WHERE age > 20
```

```c
// Define a struct for student records
struct student
{
    int id; char name[50];
    int age;
};

// Define a function for selecting students with age greater than 20
void select_students (
    struct student *condition,
    struct student *result) {

    // Declare a variable for the نمایه of the result array
    int index = 0;
    // Loop through the condition array
    for (int i = 0; i < sizeof(condition) / sizeof(condition[0]); i++)
    {
    // Check if the age of the current student is greater than 20
        if (condition[i].age > 20)
            { // Copy the current student to the result array
                result[index] = condition[i];
                // Increment the index of the result array
                index++;
            }
    }
}
```

- ساختار INSERT:

این ساختار برای درج داده‌های جدید به یک جدول استفاده می‌شود. معناشناسی عملیاتی این ساختار را می‌توان در زبان C به‌صورت یک تابع که یک پارامتر ورودی به‌عنوان داده‌های جدید و یک پارامتر خروجی به‌عنوان جدول به‌روزرسانی شده دارد پیاده‌سازی کرد.

برای مثال، ساختار زیر را می‌توان به این صورت پیاده‌سازی کرد:

```
INSERT INTO students (id, name, age) VALUES (4, 'Ali',19)
```

```c
// Define a struct for student records
struct student {
    int id;
    char name[50];
    int age;
};

// Define a function for inserting a new student to the table
void insert_student(struct student *new_record, struct student *table) {
    // Declare a variable for the size of the table
    int size = sizeof(table) / sizeof(table[0]);

    // Append the new record to the end of the table
    table[size] = *new_record;
}
```

- ساختار UPDATE:

این ساختار برای به‌روزرسانی داده‌های موجود در یک جدول استفاده می‌شود. معناشناسی عملیاتی این ساختار را می‌توان در زبان C به‌صورت یک تابع که دو پارامتر ورودی به‌عنوان شرط و عمل به‌روزرسانی و یک پارامتر خروجی به‌عنوان جدول به‌روزرسانی شده دارد پیاده‌سازی کرد.

برای مثال، ساختار زیر را می‌توان به این صورت پیاده‌سازی کرد:

```
UPDATE students SET age = age + 1 WHERE id = 2
```

```c
// Define a struct for student records
struct student {
    int id;
    char name[50];
    int age;
};

// Define a function for updating the age of a student with id 2
void update_student( struct student *condition, struct student *action, struct
student *table) {
    // Loop through the table
    for (int i = 0; i < sizeof(table) / sizeof(table[0]); i++)
    {
        // Check if the id of the current student matches the condition
        if (table[i].id == condition->id)

        { // Perform the action on the age of the current student
        table[i].age = table[i].age + action->age;
        }
    }
}
```

- ساختار DELETE:

این ساختار برای حذف داده‌های موردنظر از یک جدول استفاده می‌شود. معناشناسی عملیاتی این ساختار را می‌توان در زبان C به‌صورتی نشان داد که یک پارامتر ورودی به‌عنوان شرط حذف و یک پارامتر خروجی به‌عنوان جدول به‌روزرسانی شده دارد. برای مثال، ساختار زیر را می‌توان به این صورت پیاده‌سازی کرد:

```
DELETE FROM students WHERE id = 3
```

```
  // Define a struct for student records
  struct student {
    int id;
    char name[50];
    int age;
};
// Define a function for deleting a student with id 3 from the table
void delete_student(struct student *condition, struct student *table) {
    // Declare a variable for the index of the table
    int index = 0;
    // Loop through the table
    for (int i = 0; i < sizeof(table) / sizeof(table[0]); i++) {
        // Check if the id of the current student matches the condition
        if (table[i].id == condition->id) {
            // Skip the current student and shift the remaining students to
the left
            continue;
        }
        // Copy the current student to the new index of the table
        table[index] = table[i];
        // Increment the index of the table
        index++;
    }
}
```

- ساختار ALTER:

این ساختار برای تغییر ساختار یک جدول استفاده می‌شود. معناشناسی عملیاتی این ساختار را می‌توان در زبان C به‌صورت یک تابع نشان داد که یک پارامتر ورودی به‌عنوان عمل تغییر و یک پارامتر خروجی به‌عنوان جدول به‌روزرسانی شده دارد. برای مثال، ساختار زیر در زبان C به این صورت پیاده‌سازی می‌شود:

```
ALTER TABLE students ADD email VARCHAR(50)
```

```
// Define a struct for student records
struct student {
    int id;
    char name[50];
    int age;
};

// Define a function for adding an email column to the table
void alter_table(struct student *action, struct student *table) {
    // Declare a variable for the size of the table
    int size = sizeof(table) / sizeof(table[0]);
    // Loop through the table
    for (int i = 0; i < size; i++) {
        // Allocate memory for the new column
        table[i].email = malloc(action->email);
        // Assign a default value to the new column
        strcpy(table[i].email, "N/A");
    }
}
```

- ساختار CREATE:

این ساختار یک جدول جدید با نام و ستون‌های مشخص‌شده ایجاد می‌کند. این دستور را می‌توان با تعریف یک ساختار داده‌ای در C شبیه‌سازی کرد. برای مثال، دستور sql زیر را می‌توان به این شکل در زبان C پیاده‌سازی کرد:

```
creat table student (id int, name varchar(20), age int);
```

```
struct student {
    int id;
    char name[20];
    int age;
};
```

- ساختار JOIN:

دستور join دو یا چند جدول را بر اساس یک شرط اتصال باهم ترکیب می‌کند. این دستور را می‌توان با استفاده از حلقه‌های تودرتو در C پیاده‌سازی کرد. برای مثال، دستور sql زیر را می‌توان به شکل زیر در زبان C پیاده‌سازی کرد:

```
select * from student join course on student.id = course.student_id;
```

```
for (int i = 0; i < student_count; i++) {

    for (int j = 0; j < course_count; j++) {

        if (student[i].id == course[j].student_id) {
            printf("%d %s %d %s %d\n", student[i].id, student[i].name,
student[i].age, course[j].name, course[j].grade);
        }
    }
}
```

- ساختار GROUP BY:

دستور group by یک جدول را بر اساس یک یا چند ستون گروه‌بندی می‌کند و امکان اجرای توابع تجمیعی را بر روی هر گروه فراهم می‌کند. این دستور را می‌توان با استفاده از یک آرایه از ساختارهای داده‌ای در C شبیه‌سازی کرد. برای مثال، دستور sql زیر در زبان C به‌صورت زیر پیاده‌سازی می‌شود:

```
select name, avg(grade) from student join course on student.id = course.student_id group
by name;
```

```
struct group {
    char name[20];
    int grade_sum;
    int grade_count;
};

struct group groups[student_count];

// initialize the groups array
for (int i = 0; i < student_count; i++) {
    strcpy(groups[i].name, student[i].name);
    groups[i].grade_sum = 0;
    groups[i].grade_count = 0;
}

// iterate over the joined table and update the groups array
for (int i = 0; i < student_count; i++) {

    for (int j = 0; j < course_count; j++) {
        if (student[i].id == course[j].student_id) {
            groups[i].grade_sum += course[j].grade; groups[i].grade_count++;
        }
    }
}

// print the groups array with the average grade
for (int i = 0; i < student_count; i++) {
    if (groups[i].grade_count > 0) {
        printf("%s %f\n", groups[i].name, (float)groups[i].grade_sum /
groups[i].grade_count);
    }
}
```