How to Pass Structures to Functions in C

Introduction

Passing structures to functions in C is a fundamental concept that helps group related variables and simplifies parameter passing. This approach is especially useful when functions need multiple related arguments, as it reduces complexity and ensures logical grouping.

```c
// Define a basic structure

struct Employee {

  int id;

  char name[50];

  float salary;

  char department[30];

};


// Pass by value

void displayEmployee(struct Employee emp) {

  printf("ID: %d\n", emp.id);

  printf("Name: %s\n", emp.name);

  printf("Salary: %.2f\n", emp.salary);

  printf("Department: %s\n", emp.department);

}


// Pass by reference

void updateSalary(struct Employee *emp, float newSalary) {

  emp->salary = newSalary;   // Using arrow operator for pointer

}
```

Advantages of Using Structures

1. Efficient Parameter Passing

- Passing structures by reference avoids copying large amounts of data
- Only the pointer (typically 4 or 8 bytes) is passed
- Example:

```
void processEmployeeData(const struct Employee *emp) {
  // Using const ensures data cannot be modified
  printf("Processing employee: %s\n", emp->name);
}
```

2. Reducing Function Parameters

```
// Instead of this:
void updateEmployee(int id, char* name, float salary, char* dept);

// Use this:
void updateEmployee(struct Employee *emp);
```

3. Improved Code Readability

```
struct Rectangle {
  float length;
  float width;
};

float calculateArea(const struct Rectangle *rect) {
  return rect->length * rect->width;
}
```

4. Pass-by-Reference for Efficiency

```
// Example of efficient memory usage
void modifyEmployee(struct Employee *emp) {
  // Direct modification of original data
  strcpy(emp->department, "New Department");
  emp->salary *= 1.1; // 10% raise
}
```

5. Data Integrity

```
struct BankAccount {
  long accountNumber;
  double balance;
  char accountType[20];
  char holderName[50];
};

void processTransaction(struct BankAccount *account, double amount) {
  // All related data is guaranteed to be present
  account->balance += amount;
}
```

6. Scalability Example

```
// Original structure
struct Employee {
  int id;
  char name[50];
  float salary;
};

// Updated structure - function signatures remain same
struct Employee {
  int id;
  char name[50];
  float salary;
  char department[30];   // New field added
  char designation[30];  // New field added
  struct Date joinDate;  // New field added
};
```

Best Practices

1. Using Const for Read-only Access:

```
void printEmployeeDetails(const struct Employee *emp) {
  // Cannot modify emp's data
  printf("Employee: %s, ID: %d\n", emp->name, emp->id);
}
```

2. Error Handling:

```
int updateEmployee(struct Employee *emp, float newSalary) {
  if (emp == NULL) {
      return -1; // Error handling
  }
  if (newSalary < 0) {
      return -2; // Validation
  }
  emp->salary = newSalary;
  return 0; // Success
}
```

3. Complete Working Example:

```
#include <stdio.h>
#include <string.h>

struct Employee {
  int id;
  char name[50];
  float salary;
  char department[30];
};

// Function prototypes
void initEmployee(struct Employee *emp, int id, const char *name,
          float salary, const char *dept);
```

```
void displayEmployee(const struct Employee *emp);
int updateSalary(struct Employee *emp, float newSalary);

int main() {
  struct Employee emp1;

  // Initialize employee
  initEmployee(&emp1, 1001, "John Doe", 50000.0, "IT");

  // Display original data
  printf("Original Employee Data:\n");
  displayEmployee(&emp1);

  // Update salary
  if (updateSalary(&emp1, 55000.0) == 0) {
    printf("\nAfter Salary Update:\n");
    displayEmployee(&emp1);
  }

  return 0;
}

void initEmployee(struct Employee *emp, int id, const char *name,
        float salary, const char *dept) {
  emp->id = id;
  strncpy(emp->name, name, sizeof(emp->name) - 1);
  emp->salary = salary;
  strncpy(emp->department, dept, sizeof(emp->department) - 1);
}

void displayEmployee(const struct Employee *emp) {
  printf("ID: %d\n", emp->id);
  printf("Name: %s\n", emp->name);
  printf("Salary: %.2f\n", emp->salary);
  printf("Department: %s\n", emp->department);
}

int updateSalary(struct Employee *emp, float newSalary) {
  if (emp == NULL || newSalary < 0) {
    return -1;
  }
  emp->salary = newSalary;
  return 0;
}
```

Key Points to Remember:

1. Use pass-by-reference for large structures to improve efficiency
2. Always validate pointer parameters
3. Use const when the function shouldn't modify the structure
4. Consider memory alignment and padding
5. Maintain consistent naming conventions