Złożone typy danych

Struktury

Struktury w języku C to zaawansowany typ danych, który pozwala grupować zmienne różnych typów pod jednym identyfikatorem. Struktury są szczególnie przydatne w sytuacjach, gdy trzeba reprezentować złożone obiekty, takie jak punkty w przestrzeni trójwymiarowej, daty, czas, osoby czy produkty.

Definiowanie struktury: Struktury definiuje się za pomocą słowa kluczowego `struct`, po którym następuje nazwa struktury oraz lista zmiennych w nawiasach klamrowych:

```
struct nazwa_struktury {
    typ_danej1 zmienna1;
    typ_danej2 zmienna2;
    // ...
    typ_danejN zmiennaN;
};
```

Przykład 1 - struktura reprezentująca punkt w przestrzeni 2D:

```
struct Punkt2D {
    float x;
    float y;
};
```

Inicjalizacja struktury: Aby zainicjować zmienną typu struktury, należy użyć jej nazwy, po której umieścić nazwę zmiennej:

```
struct Punkt2D punkt;
```

Przykład 2 - inicjalizacja struktury z wartościami:

```
struct Punkt2D punkt = {3.0, 4.0};
```

Dostęp do elementów struktury: Aby uzyskać dostęp do elementów struktury, używa się operatora kropki (.):

```
punkt.x = 5.0;
punkt.y = 6.0;
float suma_wspolrzednych = punkt.x + punkt.y;
```

Przykład 3 - funkcja obliczająca odległość między dwoma punktami w przestrzeni 2D:

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

struct Punkt2D {
    float x;
    float y;
};

float odleglosc(struct Punkt2D a, struct Punkt2D b) {
    float dx = a.x - b.x;
    float dy = a.y - b.y;
    return sqrt(dx * dx + dy * dy);
}

int main() {
    struct Punkt2D p1 = {3.0, 4.0};
    struct Punkt2D p2 = {6.0, 8.0};
    float wynik = odleglosc(p1, p2);
    printf("Odleglosc miedzy punktami: %.2f\n", wynik);
    return 0;
}
```

Wskaźniki do struktur (na strukturę)

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

// Definicja struktury
struct Punkt2D {
    float x;
    float y;
```

```c
};

int main() {
    // Alokacja pamięci dla struktury Punkt2D
    struct Punkt2D *punkt = (struct Punkt2D *)malloc(sizeof(struct Punkt2D));

    // Przypisanie wartości do pól struktury poprzez wskaźnik
    punkt->x = 3.0;
    punkt->y = 4.0;

    printf("Punkt przed przesunieciem: (%.1f, %.1f)\n", punkt->x, punkt->y);

    // Przesunięcie punktu o wektor (2.0, 3.0)
    punkt->x += 2.0;
    punkt->y += 3.0;

    printf("Punkt po przesunieciu: (%.1f, %.1f)\n", punkt->x, punkt->y);

    // Zwolnienie zaalokowanej pamięci
    free(punkt);

    return 0;
}
```

typedef

`typedef` to słowo kluczowe w języku C, które pozwala na definiowanie aliasów dla typów danych. W kontekście struktur, `typedef` jest często używane w celu uproszczenia deklaracji zmiennych strukturalnych i funkcji.

Wariant 1 - Użycie `typedef` podczas definiowania struktury:

```c
#include <stdio.h>

typedef struct {
    float x;
    float y;
} Punkt2D;

int main() {
    Punkt2D punkt = {3.0, 4.0};
```

```
    printf("Punkt: (%.1f, %.1f)\n", punkt.x, punkt.y);
    return 0;
}
```

Wariant 2 - Użycie `typedef` po definiowaniu struktury:

```
#include <stdio.h>

struct Punkt {
    float x;
    float y;
};

typedef struct Punkt Punkt2D;

int main() {
    Punkt2D punkt = {3.0, 4.0};
    printf("Punkt: (%.1f, %.1f)\n", punkt.x, punkt.y);
    return 0;
}
```

Wariant 3 - Użycie `typedef` w kombinacji ze wskaźnikami na strukturę:

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

typedef struct {
    float x;
    float y;
} Punkt2D;

int main() {
    Punkt2D *punkt = (Punkt2D *)malloc(sizeof(Punkt2D));
    punkt->x = 3.0;
    punkt->y = 4.0;
    printf("Punkt: (%.1f, %.1f)\n", punkt->x, punkt->y);
    free(punkt);
    return 0;
}
```

Wariant 4 - Użycie `typedef` w funkcjach przyjmujących struktury jako argumenty:

```c
#include <stdio.h>

typedef struct {
    float x;
    float y;
} Punkt2D;

void wypisz_punkt(Punkt2D p) {
    printf("Punkt: (%.1f, %.1f)\n", p.x, p.y);
}

int main() {
    Punkt2D punkt = {3.0, 4.0};
    wypisz_punkt(punkt);
    return 0;
}
```

Przekazywanie struktur do funkcji

1. Przekazywanie struktury przez wartość:

```c
#include <stdio.h>

struct Punkt2D {
    float x;
    float y;
};

void wypisz_punkt(struct Punkt2D p) {
    printf("Punkt: (%.1f, %.1f)\n", p.x, p.y);
}

int main() {
    struct Punkt2D punkt = {3.0, 4.0};
    wypisz_punkt(punkt); // przekazanie struktury przez wartość
    return 0;
}
```

W powyższym przykładzie, funkcja `wypisz_punkt` przyjmuje strukturę `Punkt2D` jako argument. Struktura jest przekazywana przez wartość, co oznacza, że zostaje utworzona jej kopia w pamięci. Zmiany dokonane na kopii nie wpłyną na oryginalną strukturę.

2. Przekazywanie struktury przez wskaźnik:

```c
#include <stdio.h>

struct Punkt2D {
    float x;
    float y;
};

void przesun_punkt(struct Punkt2D *p, float dx, float dy) {
    p->x += dx;
    p->y += dy;
}

int main() {
    struct Punkt2D punkt = {3.0, 4.0};
    printf("Punkt przed przesunieciem: (%.1f, %.1f)\n", punkt.x, punkt.y);
    przesun_punkt(&punkt, 2.0, 3.0); // przekazanie struktury przez wskaźnik
    printf("Punkt po przesunieciu: (%.1f, %.1f)\n", punkt.x, punkt.y);
    return 0;
}
```

W powyższym przykładzie, funkcja `przesun_punkt` przyjmuje wskaźnik na strukturę `Punkt2D` jako argument. Struktura jest przekazywana przez wskaźnik, co oznacza, że funkcja otrzymuje bezpośredni dostęp do oryginalnej struktury. Zmiany dokonane przez funkcję wpłyną na oryginalną strukturę.

Zwracanie przez funkcję struktury

Przykład zwracania struktury jako zwykłej zmiennej:

```c
#include <stdio.h>

struct Point {
  int x;
  int y;
};

struct Point add_points(struct Point p1, struct Point p2) {
  struct Point result;
  result.x = p1.x + p2.x;
```

```
  result.y = p1.y + p2.y;
  return result;
}

int main() {
  struct Point p1 = {1, 2};
  struct Point p2 = {3, 4};
  struct Point sum = add_points(p1, p2);
  printf("sum: (%d, %d)\n", sum.x, sum.y);
  return 0;
}
```

Przykład zwracania struktury przez wskaźnik przekazany jako argument:

```
#include <stdio.h>

struct Point {
  int x;
  int y;
};

void add_points(struct Point p1, struct Point p2, struct Point *result) {
  result->x = p1.x + p2.x;
  result->y = p1.y + p2.y;
}

int main() {
  struct Point p1 = {1, 2};
  struct Point p2 = {3, 4};
  struct Point sum;
  add_points(p1, p2, &sum);
  printf("sum: (%d, %d)\n", sum.x, sum.y);
  return 0;
}
```

Przykład zwracania struktury przez wskaźnik przekazany (bezpośrednio):

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

struct Person {
    char *name;
```

```
    int age;
};

struct Person *create_person(char *name, int age) {
    struct Person *new_person = malloc(sizeof(struct Person));
    new_person->name = name;
    new_person->age = age;
    return new_person;
}

int main() {
    struct Person *person1 = create_person("John Doe", 30);
    printf("Name: %s, Age: %d\n", person1->name, person1->age);
    free(person1);
    return 0;
}
```

Unie

Unia (ang. union) to struktura w języku C, która umożliwia przechowywanie w jednym miejscu danych różnych typów, ale tylko jednego typu w danym momencie. Unia zajmuje tyle samo miejsca w pamięci, co największy element, który się w niej znajduje.

```
#include <stdio.h>

union Number {
    int i;
    float f;
    double d;
};

int main() {
    union Number num;
    num.i = 10;
    printf("int: %d\n", num.i);
    num.f = 3.14;
    printf("float: %f\n", num.f);
    num.d = 2.718;
    printf("double: %lf\n", num.d);
    printf("int: %d\n", num.i);
    return 0;
```

```
}
```

Typ wyliczeniowy `enum`

Typ wyliczeniowy (ang. enumeration) w języku C umożliwia zdefiniowanie nowego typu, który może przyjmować wartości zdefiniowane przez użytkownika. Każda wartość w typie wyliczeniowym jest stałą całkowitoliczbową i może być przypisana do zmiennej zdefiniowanej jako typ wyliczeniowy.

```c
#include <stdio.h>

enum Color {
    RED,
    GREEN,
    BLUE
};

int main() {
    enum Color c = GREEN;
    printf("Color value: %d\n", c);
    switch(c) {
        case RED:
            printf("Color: Red\n");
            break;
        case GREEN:
            printf("Color: Green\n");
            break;
        case BLUE:
            printf("Color: Blue\n");
            break;
        default:
            printf("Unknown color\n");
            break;
    }
    return 0;
}
```