Podstawy programowania 1

Struktury i unie

Arkadiusz Chrobot

Katedra Systemów Informatycznych

11 grudnia 2024

Plan

Struktury

Struktury w języku C są strukturami danych, które pozwalają zgromadzić w obrębie jednej zmiennej wartości wielu typów. Są one odpowiednikami rekordów używanych w innych językach programowania. Aby użyć w programie struktury najpierw musimy zdefiniować jej typ. Wzorzec takiej definicji jest następujący:

```
struct nazwa_typu_struktury
{
        typ_pola nazwa_pola_1;
        typ_pola nazwa_pola_2;
        ...
        typ_pola nazwa_pola_n;
};
```

Pola wewnątrz struktury są deklarowane tak jak zwykłe zmienne i mogą mieć dowolny typ, w tym mogą być tablicą, a nawet strukturą lub unią, która zostanie przedstawiona w dalszej części wykładu.

Struktury

Zmienne

Aby zadeklarować zmienną, która jest strukturą musimy w jej deklaracji nie tylko użyć nazwy typu struktury, ale także umieścić przed nią słowo kluczowe `struct`. Wzorzec deklaracji takiej zmiennej jest następujący:

```
struct nazwa_typu_struktury nazwa_zmiennej;
```

Można także zadeklarować zmienną strukturalną w miejscu definicji jej typu, według następującego wzorca:

```
struct nazwa_typu_struktury
{
        typ_pola nazwa_pola_1;
        ...
        typ_pola nazwa_pola_n;
} nazwa_zmiennej_1, nazwa_zmiennej_2;
```

W powyższym wzorcu zdefiniowano dwie zmienne typu strukturalnego. Jeśli potrzebowalibyśmy tylko jednej, to jej nazwę wystarczy umieścić między zamykającym nawiasem klamrowym, a średnikiem.

Struktury

Przykłady typów i zmiennych

Struktury mają wiele zastosowań. Używa się ich do gromadzenia danych różnych typów, ale logicznie ze sobą powiązanych. Mogą one np. posłużyć do przechowywania informacji o osobie:

```
struct personal_data
{
        char name[LENGTH], surname[LENGTH];
        unsigned char age, height, weight;
};
```

Zmienne typu `struct personal_data` mogą przechowywać takie informacje jak imię, nazwisko, wiek, waga i wzrost określonej osoby. Proszę zwrócić uwagę, że tak jak w przypadku zwykłych zmiennych możemy zadeklarować kilka pól tego samego typu podając najpierw identyfikator typu, a potem kolejne ich nazwy rozdzielając je przecinkami i kończąc całość średnikiem.

Struktury

Przykłady typów i zmiennych

Struktury mogą być także użyte do przechowywania pewnej liczby wartości tego samego typu, ale mających szczególne znaczenie w rozwiązywanym problemie. Przykładowo, struktura może posłużyć do przechowywania współrzędnych punktu w trójwymiarowej przestrzeni:

```
struct coordinates
{
        double x,y,z;
} point;
```

Struktury

Struktury zagnieżdżone

Jak wspomniano wcześniej, język C pozwala na zagnieżdżanie struktur. Przykładowo, można uzupełnić strukturę `struct personal_data` o pole przechowujące adres osoby (`personal_address`) określając typ tego pola jako strukturę:

```
struct personal_data {
    char name[LENGTH], surname[LENGTH];
    unsigned char age, height, weight;
    struct address {
        char street_name[LENGTH], postal_code[LENGTH];
        unsigned short int house_number;
    } personal_address;
};
```

Struktury

Tablice struktur

Język C pozwala tworzyć także tablice struktur. Przykładowo, można utworzyć tablicę zdefiniowanych wcześniej struktur typu `struct personal_data` w następujący sposób:

```
struct personal_data people[NUMBER_OF_ELEMENTS];
```

Stała `NUMBER_OF_ELEMENTS` określa liczbę elementów takiej tablicy i powinna zostać zdefiniowana przed deklaracją tej tablicy. Można również zadeklarować tablice struktur w miejscu definiowania typu struktury, podobnie jak zwykłą zmienną.

Struktury

Dostęp do pól struktury

Dostęp do pola struktury uzyskujemy za pomocą jego nazwy poprzedzonej kropką i nazwą zmiennej strukturalnej, w której jest ono osadzone, czyli według następującego wzorca:

`nazwa_zmiennej.nazwa_pola`

Przykład dla zadeklarowanej wcześniej zmiennej `point`:

```
point.x = 3;
```

Odwołanie do pola struktury zagnieżdżonej w innej strukturze wymaga dwukrotnego użycia kropki, np tak:

```
person.personal_address.house_number = 127;
```

Jeśli struktura jest elementem tablicy, to do pola tej struktury można uzyskać dostęp zastępując nazwę zmiennej we wzorcu odwołaniem do konkretnego elementu w tablicy, np.:

```
people[0].age = 37;
```

Struktury

Inicjacja zmiennych strukturalnych

Struktury mogą być zarówno zmiennymi globalnymi, jak i lokalnymi. W pierwszym przypadku są one domyślnie inicjowane wartościami zerowymi. W drugim przypadku komputer nie dokonuje ich inicjacji i powinniśmy ją przeprowadzić samodzielnie. Istnieją również sytuacje, w których chcemy nadać zmiennej globalnej typu strukturalnego inną wartość początkową niż domyślna. Inicjacji zmiennej strukturalnej możemy dokonać na trzy sposoby.

Struktury

Inicjacja zmiennych strukturalnych - pierwszy sposób

Inicjacji zmiennej typu strukturalnego możemy dokonać w miejscu jej deklaracji, w podobny sposób, jak inicjacji tablicy - umieszczając wartości dla pól w nawiasach klamrowych i rozdzielając je przecinkami:

```
#include <stdio.h>

struct coordinates
{
    double x, y, z;
} point = {1.0, 2.0, 3.0};

int main(void)
{
    struct coordinates another_point = {4.0, 5.0, 6.0};
    printf("x: %f ",another_point.x);
    printf("y: %f ", another_point.y);
    printf("z: %f\n", another_point.z);
    return 0;
}
```

Struktury

Inicjacja zmiennych strukturalnych - pierwszy sposób

Przykład zaprezentowany na poprzednim slajdzie pokazuje inicjację dwóch zmiennych typu `struct coordinates`: `point` i `another_point`. Pierwsza zmienna została zadeklarowana w miejscu definicji typu i tam też przypisano wartości jej polom, w takiej kolejności jak zostały one zadeklarowane, tj. polu `x` przypisano wartość `1.0`, polu `y` liczbę `2.0` itd. Druga zmienna została zadeklarowana jako zmienna lokalna i jej polom również przypisano wartości w miejscu deklaracji. Gdyby zamiast trzech liczb w nawiasach klamrowych umieszczono tylko dwie, to zgodnie z zapisami w standardzie języka C trzecie pole (w przykładzie to pole o identyfikatorze `z`) otrzymałoby wartość zero. Funkcje `printf()` i `scanf()` nie dysponują specjalnymi ciągami formatującymi, które pozwoliłby bezpośrednio wypisać wartości takich zmiennych na ekran lub pobrać je z klawiatury. Musimy to zrobić osobno dla każdego pola, tak jak pokazano to w przykładowym programie.

Struktury

Inicjacja zmiennych strukturalnych - drugi sposób

Drugi sposób jest podobny do pierwszego, ale występuje w nim dodatkowy element w postaci nazw pól, którym przypisujemy wartości. Są one umieszczane w nawiasie klamrowym i poprzedzone znakiem kropki, tak jak w przykładowym programie.

```
#include<stdio.h>

struct coordinates
{
    double x, y, z;
} point = {.x=1.0, .z=2.0, .y=3.0};


int main(void)
{
    struct coordinates another_point = point;
    printf("x: %f ",another_point.x);
    printf("y: %f ", another_point.y);
    printf("z: %f\n", another_point.z);
    return 0;
}
```

Struktury

Inicjacja zmiennych strukturalnych - drugi sposób

Dzięki użyciu nazw pól można je inicjować w dowolnej kolejności. Możliwe jest również zainicjowanie tylko części z nich. Te, które zostaną pominięte uzyskają wartość zero. Przykładowy program pokazuje również, że zmiennej strukturalnej możemy przypisać inną zmienną strukturalną tego samego typu. W wyniku takiego przypisania wartości pól ze zmiennej strukturalnej znajdującej się po prawej stronie instrukcji przypisania są kopiowane do odpowiednich pól w zmiennej znajdującej się po lewej stronie tej instrukcji. Takie działanie nie jest możliwe dla zmiennych strukturalnych o różnych typach. Nie można także rzutować zmiennej strukturalnej na inny typ strukturalny.

Struktury

Inicjacja zmiennych strukturalnych - trzeci sposób

Ostatni sposób inicjacji zmiennej strukturalnej polega na przypisaniu wartości jej polom poza miejscem jej deklaracji:

```
#include <stdio.h>

struct coordinates
{
    double x, y, z;
};


int main(void)
{
    struct coordinates point;

    point.x = point.y = point.z = 1.0;

    printf("x: %f ", point.x);
    printf("y: %f ", point.y);
    printf("z: %f\n", point.z);
    return 0;
}
```

Struktury

Inicjacja zmiennych strukturalnych - trzeci sposób

W przykładowym programie każde pole zmiennej `point` uzyskało wartość `1.0`. Jeśli pominięte zostałoby choć jedno pole, to jego wartość zależałaby do miejsca definicji zmiennej. W przypadku zmiennych globalnych pominięte pola miałyby wartość zero, a w przypadku lokalnych nieokreśloną.

Unie

Unia jest podobną konstrukcją do struktury w języku C. Różnica między nimi, oprócz sposobu definicji typu i deklaracji zmiennej, polega na tym, że pola zdefiniowane wewnątrz unii są umieszczone w tym samym obszarze pamięci, czyli nakładają się na siebie. W związku z tym unia zajmuje mniej miejsca w pamięci operacyjnej niż struktura o takich samych polach, ale modyfikacja jednego z jej pól wpływa najczęściej również na wartości pozostałych pól.

Unie

Przykład unii

Typ unii jest definiowany analogicznie do typu struktury. Również zmienne takiego typu są deklarowane tak jak zmienne typów strukturalnych.

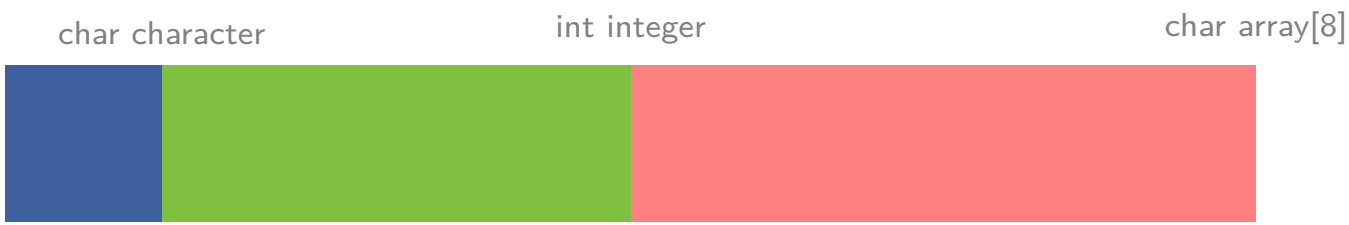
```
union union_type_example
{
    char character;
    int  integer;
    char array[8];
} union_example;
```

Listing zawiera definicję typu unii i deklaracje zmiennej, która nią jest. Jej rozmiar (liczbę zajmowanych bajtów) można określić za pomocą operatora `sizeof`. Uzyskane rezultaty mogą być różne w zależności od konfiguracji komputera i kompilatora, ale zawsze unia zajmuje mniej miejsca w pamięci niż odpowiadająca jej struktura.

Unie

Nakładanie pól

Rysunek zamieszczony na tym slajdzie ilustruje nakładanie pól unii.



Ilustracja ma charakter poglądowy. Sposób nakładania pól zależny jest od typu i konfiguracji komputera, niemniej zawsze modyfikacja jednego z nich może oznaczać również modyfikację pozostałych lub przynajmniej części z nich.

Unie

Podobieństwa między strukturami i uniami

Nie tylko definicje typów i deklaracje zmiennych struktur i unii są podobne. Unie mogą być inicjowane podobnie jak struktury, ale jeśli zastosujemy pierwszy z opisanych w tych materiałach sposobów do inicjacji więcej niż jednego pola unii, to kompilator wystosuje ostrzeżenie, że ostateczna wartość pól może być inna od tej, której się spodziewamy.
Zarówno w przypadku unii, jak i struktur możemy skrócić zapis typu zmiennej, który składa się ze słowa `stuct` lub `union` i nazwy typu, nadając tym typom inną nazwę przy pomocy słowa kluczowego `typedef`. To rozwiązanie zmniejsza jednak czytelność kodu i nie zawsze zalecane jest jego stosowanie.

Unie

Zastosowania

Bardzo często fakt nakładania się pól unii jest wykorzystywany do konwersji typów różnych wartości, np. adresów IP z zapisu dziesiętnego na binarny, czy liczby z zapisu dziesiętnego na BCD. Taka konwersja polega na zapisaniu wartości do określonych pól unii i odczytaniu wartości innych jej pól. Nie jest to jednak zalecane przez standard języka C zastosowanie unii, ponieważ wynik takiej konwersji może być różny dla różnych komputerów i nie zawsze poprawny. Standard zaleca, aby odczytywać zawsze to pole unii, które ostatnio było w programie zapisane. Lepiej więc zastosować unie jako pola struktury, celem zaoszczędzenia miejsca w pamięci. Przy takim sposobie korzystania z unii konieczne jest zadeklarowanie w strukturze pola, które będzie pełniło rolę selektora dla pól unii. Zastosowanie to zostanie przedstawione w programie zaprezentowanym w dalszej części wykładu.

Pola bitowe

W języku C możliwe jest określanie wielkości pól struktury w bitach. Pola o tak zdefiniowanym rozmiarze nazywamy *polami bitowymi*. Nie można zastosować dla nich operatora `sizeof`, ani przypisać im wartości większej, niż wynika to z ich rozmiaru. Nie oznacza, to jednak, że rozmiar struktury, która je zawiera jest sumą ich rozmiarów. Np. struktura, która zawiera wyłącznie dwa pola o rozmiarze pięciu bitów ma rozmiar co najmniej dwóch bajtów, a nie dziesięciu bitów. Rozmiar każdej struktury jest całkowitą, dodatnią wielokrotnością jednego bajta i wynosi co najmniej jeden bajt. Pola bitowe są specjalnym zapisem, który wymusza na komputerze używanie do przechowywania wartości określonej przez programistę części bitów pola. Do określenia typu pola bitowego może być zastosowany jedynie któryś z typów danych pozwalających przechowywać liczby całkowite lub naturalne, np. `int` lub `unsigned char`. Choć możliwe jest zadeklarowanie w unii pól bitowych, to przydatność takiego rozwiązania jest znikoma.

Pola bitowe

Przykład struktury z polami bitowymi

```
struct bit_field_example
{
    int flag:1;
    char small_number:2;
};
```

Pojedyncze bity często wykorzystywane są jako tzw. flagi, czyli zmienne np. pamiętające czy wystąpił błąd, czy też nie. Stąd nazwa pola o rozmiarze jednego bita w zaprezentowanej definicji typu strukturalnego.

Struktury i unie a funkcje

Zarówno unie jak i struktury mogą być zwracane przez funkcje. Na listingu zamieszczony jest kod źródłowy przykładowej funkcji zwracającej strukturę. Dla unii byłaby ona zdefiniowana analogicznie.

```
struct coordinates get_point(double x, double y, double z)
{
    struct coordinates point;

    point.x = x;
    point.y = y;
    point.z = z;

    return point;
}
```

Struktury i unie a funkcje

Zwracanie struktury przez funkcję - komentarz do przykładu

Zaprezentowana na poprzednim slajdzie funkcja umieszcza przekazane jej przez parametry wartości w strukturze zadeklarowanej lokalnie, a następnie zwraca ją jako wynik swojego działania. Ta funkcja może być wywołana np. następująco:

```
struct coordinates start = get_point(0.0, 0.0, 0.0);
```

W wyniku wykonania funkcji wartości z jej lokalnej zmiennej strukturalnej zostaną skopiowane do zmiennej `start`.

Struktury i unie a funkcje

Przekazywanie przez parametry

Struktury i unie mogą pełnić rolę argumentów i parametrów w funkcjach. Domyślnie, tak jak w przypadku każdej zmiennej innej niż tablica, są one przekazywane przez wartość. Jeśli chcielibyśmy, aby modyfikacje dokonane w ich polach nie uległy zniszczeniu po zakończeniu działania funkcji, to możemy je przekazać przez wskaźnik (adres). Dostęp do pól w tak przekazanej strukturze można uzyskać na dwa sposoby. Pierwszy jest mniej czytelny i polega na zastosowaniu następującego wzorca odwołania:

```
(*nazwa_wskaźnika_do_struktury).nazwa_pola
```

Drugi jest bardziej czytelny, dzięki zastosowaniu specjalnego operatora zapisywanego jako -> i przez to częściej stosowany:

```
nazwa_wskaźnika_do_struktury->nazwa_pola
```

Struktury i unie a funkcje

Przykład

```
void move(struct coordinates *point,
                                    struct coordinates vector)
{
    point->x += vector.x;
    point->y += vector.y;
    point->z += vector.z;
}
```

Struktury i unie a funkcje

Komentarz do przykładu

Funkcja z poprzedniego slajdu wylicza nowe współrzędne punktu po przesunięciu go o zadany wektor w trójwymiarowej przestrzeni. Pierwszy parametr jest wskaźnikiem na strukturę, która przed rozpoczęciem wykonania funkcji będzie zawierała początkowe współrzędne punktu, a po jej zakończeniu współrzędne końcowe. Przez drugi parametr jest przekazywana struktura opisująca wektor, o jaki ma być przesunięty punkt. Zawartość każdego pola tej struktury należy zatem traktować nie jako wartość współrzędnej punktu, a jako wartość współrzędnej wektora. Funkcja dodaje odpowiednie wartości odpowiednich pól do siebie i zapisuje je w zmiennej `point`. Jako pierwszy argument jej wywołania należy przekazać wskaźnik na strukturę typu `struct coordinates` lub adres takiej struktury, a jako drugi wprost taką strukturę. Może ona zatem być wywołania np. następująco:

```
move(&start,distance);
```

Przykłady

Tablica struktur

Jako pierwszy przykład zostanie zaprezentowany program, w którym zastosowano tablicę struktur do przechowywania danych osobowych ludzi, takich jak imię, nazwisko i wiek. Dane te będą tworzone w sposób pseudolosowy, tzn. wiek będzie losowany z zakresu od 1 roku do 120 lat, a imię i nazwisko będą losowane ze wcześniej zdefiniowanych tablic. Każda z nich zawiera zarówno męskie, jak i żeńskie imiona i nazwiska.

Przykład

Tablica struktur

```c
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<time.h>

#define LENGTH 50
#define NUMBER_OF_PEOPLE 5

enum gender {MALE, FEMALE};
```

Przykład

Tablica struktur - komentarz

Fragment kodu zaprezentowany na poprzednim slajdzie oprócz instrukcji włączenia odpowiednich plików nagłówkowych zawiera także dwie definicje stałych, z których pierwsza będzie używana do określenia liczby elementów tablic przechowujących imię i nazwisko osoby, a druga określa dla ilu osób zostaną wygenerowane dane. Ponadto został zdefiniowany także typ wyliczeniowy, którego elementy będą stosowane jako stałe określające płeć osoby.

Przykład

Tablica struktur

```
struct name_forms
{
    char male_name[LENGTH], female_name[LENGTH];
} names[]={{.male_name = "Andrzej", .female_name = "Anita"},
           {.male_name="Edward", .female_name="Katarzyna"},
           {.male_name="Henryk", .female_name="Magdalena"},
           {.male_name="Ireneusz", .female_name="Beata"},
           {.male_name="Jakub", .female_name="Joanna"}},

 surnames[]={{.male_name="Kowalski", .female_name="Kowalska"},
             {.male_name="Zapolski", .female_name="Zapolska"},
       {.male_name="Sienkiewicz", .female_name="Orzeszkowa"},
         {.male_name="Żeromski", .female_name="Konopnicka"},
         {.male_name="Dąbrowski", .female_name="Potocka"}};
```

Przykład

Tablica struktur - komentarz

Zaprezentowany na poprzednim slajdzie fragment kodu źródłowego wygląda dosyć skomplikowanie, ale jest on po prostu deklaracją dwóch zainicjowanych tablic o nazwach `names` i `surnames`. Elementy tych tablic są strukturami typ `struct name_forms`. Każdy z nich przechowuje męskie lub żeńskie imię lub nazwisko. Elementy tych tablic będą losowane celem utworzenia rekordów informacji o osobach.

Przykład

Tablica struktur

```
struct personal_data
{
    char name[LENGTH], surname[LENGTH];
    unsigned char age;
} people_data[NUMBER_OF_PEOPLE];
```

Przykład

Tablica struktur - komentarz

Poprzedni slajd zawiera definicję tablicy, w której będą przechowywane informacje o osobach. Tablica ta będzie miała liczbę elementów określoną stałą `NUMBER_OF_PEOPLE`. Każdy z nich będzie strukturą, która przechowuje imię, nazwisko i wiek pojedynczej osoby.

Przykład

Tablica struktur

```
struct personal_data get_randomized_data(struct name_forms names[], struct name_forms surnames[])
{
   struct personal_data person;

   person.age = 1+rand()%120;
   unsigned char gender = rand()%2;
   if(gender==FEMALE) {
        strncpy(person.name,names[rand()%NUMBER_OF_PEOPLE].female_name,LENGTH-1);
        strncpy(person.surname,surnames[rand()%NUMBER_OF_PEOPLE].female_name,LENGTH-1);
   } else {
       strncpy(person.name,names[rand()%NUMBER_OF_PEOPLE].male_name,LENGTH-1);
       strncpy(person.surname,surnames[rand()%NUMBER_OF_PEOPLE].male_name,LENGTH-1);
   }
   return person;
}
```

Przykład

Tablica struktur - komentarz

Funkcja `get_randomized_data()` generuje dane pojedynczej osoby i zwraca je w strukturze typu `struct personal_data`. Przekazywane są do niej przez parametry dwie wcześniej opisane tablice zainicjowane zawierające imiona i nazwiska męskie i żeńskie. Funkcja najpierw losuje wiek z ustalonego zakresu i zapisuje go w odpowiednim polu lokalnej struktury. Następnie losowana jest płeć osoby. Jeśli będzie to kobieta, to losowane są elementy z zainicjowanych tablic imion i nazwisk, a następnie kopiowane jest z nich imię i nazwisko żeńskie do wynikowej struktury. Analogicznie funkcja postępuje gdy wylosowany zostanie mężczyzna.

Przykład

Tablica struktur

```
void fill_array(struct personal_data array[],
                struct name_forms names[],
                struct name_forms surnames[])
{
    srand(time(0));
    int i;
    for(i=0;i<NUMBER_OF_PEOPLE;i++)
        array[i]=get_randomized_data(names, surnames);
}
```

Przykład

Tablica struktur - komentarz

Funkcja `fill_array()` inicjuje generator liczb pseudolosowych i nadaje każdemu z elementów tablicy przekazanej jej przez pierwszy parametr wartość zwróconą przez wywołanie funkcji `get_randomized_data()`.

Przykład

Tablica struktur

```
void print_array(struct personal_data array[])
{
    int i;
    for(i=0;i<NUMBER_OF_PEOPLE;i++) {
        printf("Imię: %s\n",array[i].name);
        printf("Nazwisko: %s\n",array[i].surname);
        printf("Wiek: %u\n",array[i].age);
        puts("");
    }
}
```

Przykład

Tablica struktur - komentarz

Funkcja `print_array()` wypisuje zawartość przekazanej jej przez parametr tablicy na ekran. Wartość każdego pola każdego elementu jest wypisywana w osobnym wierszu. Dodatkowo po wypisaniu każdego elementu kursor jest przenoszony o jeden wiersz niżej, aby odseparować od siebie wizualnie kolejne elementy.

Przykład

Tablica struktur

```
int main(void)
{
    fill_array(people_data, names, surnames);
    print_array(people_data);
    return 0;
}
```

Przykład

Tablica struktur - komentarz

W funkcji `main()` wywoływane są funkcje `fill_array()` i `print_array()` z odpowiednimi argumentami. Proszę zwrócić uwagę, że w obu funkcjach tablica `people_data` jest przekazywana przez parametr o nazwie `array`. Przypominam, że nazwy parametrów nie muszą być takie same jak nazwy argumentów, które są pod nie podstawiane w miejscu wywołania funkcji. Muszą się jedynie zgadzać ich typy.

Przykład

Struktura i unia

Kolejny przykład pokazuje zastosowanie unii jako składowej (pola) w strukturze. Innymi słowy unia będzie zagnieżdżona w strukturze. Obie zmienne zostaną zastosowane w programie, który tworzy i wyświetla na ekranie informacje o postaci z gry komputerowej.

Przykład

Struktura i unia

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

#define LENGTH 10

enum character_type {WARRIOR, SORCERER};
```

Przykład

Struktura i unia - komentarz

Fragment kodu z poprzedniego slajdu zawiera instrukcje włączające pliki nagłówkowe, definicję stałej, która posłuży do określenia liczby elementów tablicy przechowującej nazwę postaci oraz definicję typu wyliczeniowego, który określa typ postaci: czarodziej lub wojownik.

Przykład

Struktura i unia

```
struct playable_character {
    char name[LENGTH];
    enum character_type type;
    union {
        float strength;
        double magic_power;
    } abilities;
};
```

Przykład

Struktura i unia - komentarz

Poprzedni slajd zawiera definicję typu struktury przechowującej informacje o postaci. Pierwsze pole jest tablicą, która będzie przechowywała jej nazwę, drugie będzie zawierało liczbę identyfikującą jej typ. Trzecie jest polem, którego typ jest określony unią. Jeśli postać będzie wojownikiem, to zapisywane będzie pole `strength`, przechowujące informację o mierze siły wojownika, a jeśli czarownikiem, to w polu `magic_power` będzie przechowywana informacja o mierze jego mocy magicznej.

Przykład

Struktura i unia

```
void generate_character(struct playable_character *character)
{
    puts("Nazwij swoją postać:");
    scanf("%9s",character->name);
    srand(time(0));
    if(rand()%2==WARRIOR) {
        character->type = WARRIOR;
        character->abilities.strength = rand()%1000+(float)rand()/(RAND_MAX+1.0);
    } else {
        character->type = SORCERER;
        character->abilities.magic_power = 1000+rand()%RAND_MAX
                                           + (double)rand()/(RAND_MAX+1.0);
    }
}
```

Przykład

Struktura i unia - komentarz

Funkcja `generate_character()` wypełnia strukturę, która jest jej przekazana przez parametr, danymi. Najpierw prosi użytkownika o wprowadzenie nazwy postaci z klawiatury. Ta nazwa jest wprowadzana do pola `name` struktury przy pomocy funkcji `scanf()`. Proszę zwrócić uwagę na sposób odwołania do tego pola. Ponieważ struktura przekazywana jest do funkcji przez wskaźnik, to stosujemy notację „ze strzałką", aby uzyskać dostęp do jej pól. Liczba znaków odczytywanych przez funkcję `scanf()` jest ograniczona do 9 ze względu na liczbę elementów pola. Następnie funkcja `generate_character()` uruchamia generator liczb pseudolosowych i losuje typ postaci. Jeśli będzie to wojownik, to informacja o tym jest zapisywana w polu `type` struktury i losowana jest wartość dla pola `strength` unii `abilities`. Proszę zwrócić uwagę, na sposób odwołania do pola unii. Wprawdzie struktura jest przekazana przez wskaźnik, ale unia jest zwykłą zmienną w niej umieszczoną, więc do jej pól odwołujemy się za pomocą notacji „kropkowej". W przypadku wylosowania czarownika funkcja postępuje analogicznie.

Przykład

Struktura i unia

```
void print_character(struct playable_character character)
{
    printf("Nazwa: %s\n",character.name);
    if(character.type==WARRIOR) {
        printf("Typ: wojownik\n");
        printf("Siła: %f\n",character.abilities.strength);
    } else {
        printf("Typ: czarownik\n");
        printf("Moc magiczna: %f\n",
                        character.abilities.magic_power);
    }
}
```

Przykład

Struktura i unia - komentarz

Funkcja `print_character()` wypisuje informacje o postaci, które są zgromadzone w strukturze. Ponieważ tym razem struktura przekazywana jest przez wartość, to do każdego jej pola, jak również pól unii odwołujemy się za pomocą notacji „kropkowej”. Wypisanie wartości pola `type` spowodowałoby pojawienie się na ekranie liczby `0` lub `1`, w zależności od typu postaci. Dlatego to pole służy jedynie do rozróżnienia dwóch rodzajów bohaterów i wypisaniu odpowiedniego komunikatu wraz z wartością odpowiedniego pola unii `abilities`.

Przykład

Struktura i unia

```
int main(void)
{
    struct playable_character character;
    generate_character(&character);
    print_character(character);
    return 0;
}
```

Przykład

Struktura i unia - komentarz

W funkcji `main()` tworzona jest lokalna zmienna strukturalna o nazwie `character`, a następnie jest ona wypełniana przy pomocy wywołania funkcji `generate_character()`, do której jest przekazywany jej adres. Potem zawarate w tej zmiennej informacje są wypisywane na ekranie za pomocą wywołnia funkcji `print_character()`.

Podziękowania

Składam podziękowania dla dra inż. Grzegorza Łukawskiego i dra inż. Leszka Ciopińskiego za udostępnienie materiałów, których fragmenty zostały wykorzystane w tym wykładzie.

Pytania

?

KONIEC

Dziękuję Państwu za uwagę.