Föreläsningsanteckning 7
DA105A - Programmering med C, grundkurs, 7.5 hp

..

Innehåll

1 Inledning

Denna föreläsning behandlar pekare, tecken och strängar, samt dataposter. Föreläsningen ger översiktlig information om pekare, tecken och strängar, samt exemplifierar hur dataposter kan definieras och användas. Föreläsningen behandlar material som ingår i kapitel 7, kapitel 8 och kapitel 10, i kursboken *C How to Program - Fifth Edition* av H.M. Deitel och P.J. Deitel.

Beteckningen [DEITEL] används nedan för att referera till denna bok.

Föreläsningen använder bostadskalkyl-programmet som beskrivits i tidigare föreläsningar, och exemplifierar hur dataposter kan användas för att modifiera detta program. Modifieringarna utgår från den version av programmet som behandlats i föreläsning 6 [1].

2 Pekare

En *pekare* i programspråket C är en variabel, vars värde utgör en *adress*. Denna adress refererar till ett ställe i minnet, där en (annan) variabel, för det mesta av en given typ, kan finnas lagrad. Man kan därför tala om en pekare till en variabel av typen *int*, en pekare till en variabel av typen *double*, etc. Det finns även pekare som inte adresserar någon speciell datatyp. Sådana pekare kan t.ex. användas när man vill använda en referens till en godtycklig adress, eller till en variabel vars datatyp inte är känd. Pekare som inte adresserar någon speciell datatyp behandlas inte ytterligare här.

Man kan även använda pekare som adresserar olika typer av *datastrukturer*. På detta sätt kan man bygga upp sammansatta datastrukturer, som t.ex. kallas *länkade listor* eller *träd*.

2.1 Definition och användning

Pekare kan deklareras som variabler. En pekare, t.ex. kallad *i_ptr*, till variabler av typen *int*, kan deklareras enligt

```
/* a pointer to an integer */
int *i_ptr;
```

Denna pekare kan t.ex. adressera en variabel deklarerad enligt

```
/* an integer */
int i;
```

Om pekaren *i_ptr* tilldelas ett värde enligt

```
/* i_ptr is set to the address of i */
i_ptr = &i;
```

så resulterar detta i att pekaren *i_ptr* nu adresserar variabeln *i*. Detta innebär att det värde som variabeln *i_ptr* har är *adressen* till variabeln *i*. Man kan då säga att variabeln *i_ptr adresserar* variabeln *i*, alternativt att variabeln *i_ptr pekar på* variabeln *i*, eller att variabeln *i_ptr refererar till* variabeln *i*.

Notationen **i_ptr* används för att referera till det som *i_ptr* pekar på. Om *i_ptr* pekar på *i*, enligt ovanstående exempel, så innebär detta att notationen **i_ptr* markerar den plats i minnet där variabeln *i* är lagrad. Detta innebär att om variabeln *i* ges ett värde, t.ex. enligt

```
/* i is assigned a value */
i = 5;
```

så kommmer **i_ptr* att vara 5.

Innehållet i variabeln *i* kan därmed läsas eller skrivas på två sätt, antingen direkt via *i* självt, eller *indirekt*, via pekaren *i_ptr*. Detta kan illustreras genom att göra en utskrift, där värdet på *i* skrivs ut på två sätt, enligt

```
/* print i in two ways */
printf("i: %d, via pointer: %d\n", i, *i_ptr);
```

På motsvarande sätt kan man skriva ut värdet på pekaren *i_ptr* på två sätt, enligt

```
/* print the address of i in two ways */
printf("address of i: %d, via pointer: %d\n", &i, i_ptr);
```

vilket alltså är en utskrift av en *adress*.

Man använder ofta *hexadecimala* tal när man arbetar med adresser. Det går bra att skriva ut *i_ptr* som ett hexadecimalt värde, genom att använda styrkoden `%X` till printf, enligt

```
/* print the address of i in hexadecimal in two ways */
printf("address of i: 0x%X, via pointer: 0x%X\n", &i, i_ptr);
```

där även teckensekvensen *0x* har lagts till, framför de hexadecimala värdena. Denna teckensekvens används ofta som ett prefix för hexadecimala tal, och används i programspråket C för att markera att ett givet heltal skall tolkas som ett hexadecimalt tal.

2.2 Referensanrop och pekare

Avsnitt 3.2 i föreläsning 5 [2] ger en kort beskrivning av *referensanrop*. Vid ett referensanrop överförs adressen till en variabel till en funktion. Detta kan också uttryckas som att en *pekare* överförs som en parameter till funktionen.

Det är brukligt att använda tecknet & framför en variabel när denna skall överföras till en funktion via referensanrop, t.ex. när man anropar *scanf* för att läsa in ett värde, eller när man anropar andra funktioner som ger utdata via sina parametrar. Eftersom tecknet & här markerar adressen till en variabel, så kan man säga att man skickar en *pekare* till variabeln som argument, vid funktionsanropet. Man kan notera att denna pekare inte har något specifikt *namn*, t.ex. som pekaren *i_ptr* i avsnitt 2.1. Det går bra att även använda namngivna pekare i samband med referensanrop, vilket då innebär att man inte behöver använda tecknet &. Denna situation uppkommer t.ex. om man skriver en funktion för att läsa in två tal, enligt

```
/* read_two_numbers: reads two integer numbers, and returns the two
   numbers in parameters *number_1 and *number_2 */
void read_two_numbers(int *number_1, int *number_2)
{
    printf("enter two numbers, separated by a blank: ");
    scanf("%d %d", number_1, number_2);
}
```

där funktionen *read_two_numbers* använder sina två parametrar som två namngivna pekare, *number_1* och *number_2*, vid anropet av *scanf*.

2.3 Samband mellan pekare och fält

Ett fält som används som parameter till en funktion överförs via referensanrop. Detta beskrivs kort i avsnitt 5 i föreläsning 6 [1]. Ett annat sätt att

formulera detta är att säga att en *pekare till fältets data* överförs i samband med att den aktuella funktionen anropas. I själva verket är det så att ett fält i programspråket C även kan betraktas som en *pekare till fältets första element*.

Tolkningen av ett fält som en pekare kan anges explicit i parameterlistan till en funktion. Detta innebär att en funktion, t.ex.

```
/* fill_double_array: fills the array arr, having the
   length length, with the value value */
void fill_double_array(double arr[], int length, double value)
{
    /* loop counter */
    int i;

    for (i = 0; i < length; i++)
    {
        arr[i] = value;
    }
}
```

även kan skrivas som

```
/* fill_double_array: fills the array arr, having the
   length length, with the value value */
void fill_double_array(double *arr, int length, double value)
{
    /* loop counter */
    int i;

    for (i = 0; i < length; i++)
    {
        arr[i] = value;
    }
}
```

där funktionens första parameter istället anges som en pekare till variabler av typen *double*.

Denna alternativa tolkning av fält är speciellt användbar om man vill skapa fält vars storlek specificeras dynamiskt, dvs. medan programmet exekverar. Man använder då en pekare, samt allokerar minne till denna pekare, där storleken på det allokerade minnet motsvarar fältets önskade storlek. Detta ger en mer flexibel hantering än när man definierar fältets storlek med *#define*-direktiv, vilket är den metod som använts (och kommer att användas, i de allra flesta fall) i denna kurs.

3 Tecken och strängar

Ett *tecken* representeras i programspråket C genom att använda datatypen *char*. En *sträng*, i betydelsen en sekvens av tecken, representeras i programspråket C med ett fält, vars element är av typen *char*. Det finns en konvention, i programspråket C, som innebär att en sträng skall lagras i ett fält på ett sådant sätt att strängens slut markeras av tecknet *null*.

Denna konvention innebär att man, givet en sträng, lagrad i ett fält, kan ta reda på strängens storlek, genom att räkna antalet tecken tills man träffar på null-tecknet. Detta utnyttjas av ett flertal funktioner som ingår i standardbiblioteket till C.

Som ett exempel kan man t.ex. studera funktionen *evaluate_buy*, som beskrivs i avsnitt 3.1 i föreläsning 5 [2]. Denna funktion utför ett anrop av *printf*, där en konstant sträng skickas med som argument. Eftersom denna sträng representeras som en följd av tecken som avslutas med ett null-tecken så kan funktionen *printf* på ett enkelt sätt ta reda på strängens storlek, och därmed skriva ut rätt antal tecken på skärmen.

För ytterligare information om strängar, samt de funktioner som finns tillgängliga i standardbiblioteket för att arbeta med strängar, hänvisas till kapitel 8 i [DEITEL].

4 Dataposter

En *datapost* är en datastruktur med ett antal *element*, där elementen tillåts vara av *olika* typ. Detta kan jämföras med ett fält, där alla element är av *samma* typ. En datapost i programspråket C definieras med hjälp av det reserverade ordet *struct*. Man kan kombinera det reserverade ordet *struct* med det reserverade ordet *typedef*, och definiera en datapost som också blir en *datatyp*. Detta kan exemplifieras, med referens till det tidigare beskrivna bostadskalkyl-programmet, genom att definiera en datatyp för de olika kvantiteter som påverkar boendekostnaden. Dessa kvantiteter, som också skulle kunna kallas för *parametrar*, kan då hållas samman i en datapost, som definieras enligt

```
/* data structure holding parameters that affect
   the living cost */
typedef struct
{
    /* price of flat */
    double price;
```

```
    /* interest rate (per year) */
    double interest_p;

    /* montly income (SEK) */
    double monthly_income;

    /* monthly instalment (amortering) */
    double monthly_instalment;

    /* monthly rent */
    double monthly_rent;

    /* number of years for prediction */
    int n_years_future;
} living_parameter_type;
```

Denna deklaration definierar en *datatyp*, som har namnet *living_parameter_type*.

Man kan skriva en funktion som fyller i värden i en variabel av denna typ, t.ex. default-värden som skall användas om inte användaren anger egna värden när programmet körs. En sådan funktion kan skrivas enligt

```
/* init_living_parameters: initialises the data structure *lp */
void init_living_parameters(living_parameter_type *lp)
{
    /* price of flat */
    lp->price = 1000000;

    /* interest rate (per year) */
    lp->interest_p = 3.5;

    /* montly income (SEK) */
    lp->monthly_income = 15000;

    /* monthly instalment (amortering) */
    lp->monthly_instalment = 2000;

    /* monthly rent */
    lp->monthly_rent = 1500;

    /* number of years for prediction */
    lp->n_years_future = 10;
}
```

Funktionen *init_living_parameters* har en parameter. Denna parameter är en *pekare till en variabel av typen living_parameter_type*, alltså en pekare till en

datapost, med utseende enligt ovanstående definition av *living_parameter_type*.

Funktionen *init_living_parameters* använder notationen `->` för att referera till de olika element som finns i dataposten. Notationen används för att tilldela default-värden till de olika elementen.

På motsvarande sätt kan man tillverka en funktion som läser in värden från användaren, samt lagrar dessa värden i en datapost av typen *living_parameter_type*. En sådan funktion kan skrivas enligt

```
/* read_input_values: reads input values and stores the values read
   in the struct referred to (pointed to) by lp */
void read_input_values(living_parameter_type *lp)
{
    /* get values from user, using %lg for input of double values */
    printf("Enter price: ");
    scanf("%lg", &lp->price);

    printf("Enter interest rate for loan: ");
    scanf("%lg", &lp->interest_p);

    printf("Enter monthly income: ");
    scanf("%lg", &lp->monthly_income);

    printf("Enter monthly instalment: ");
    scanf("%lg", &lp->monthly_instalment);

    printf("Enter monthly rent: ");
    scanf("%lg", &lp->monthly_rent);

    printf("Enter number of years for prediction: ");
    scanf("%d", &lp->n_years_future);
}
```

Man kan deklarera en variabel av typen *living_parameter_type*, enligt

```
    /* parameters determining the living cost */
    living_parameter_type lp;
```

Denna variabel kan sedan användas, vid anrop av t.ex. funktionen *init_living_parameters*, enligt

```
    /* initialise living parameters */
    init_living_parameters(&lp);
```

där *adressen* till variabeln *lp* används som argument vid anropet. Variabeln *lp* kan också användas vid ett anrop till *read_input_values*, t.ex. enligt

```
    /* ask if default values shall be used, and
       get values from user if answer is not 'y' */
    if (get_y_n_input_ch() != 'y')
    {
        read_input_values(&lp);
    }
```

Man kan också skapa en datatyp som är en datapost med ett eller flera *fält*, t.ex. enligt

```
/* data structure with information about
   the living cost */
typedef struct
{
    /* interest cost per month */
    double monthly_interest_cost[MAX_N_YEARS];

    /* the remains, for food, bills and pleasure */
    double monthly_balance[MAX_N_YEARS];

    /* the remaining debt */
    double net_debt[MAX_N_YEARS];
} living_cost_type;
```

Denna datatyp kan användas för att skriva om funktionen *calculate_future_data*, som beskrivs i avsnitt 5 i föreläsning 6 [1], så att funktionens parameterlista förenklas. Den resulterande parameterlistan innehåller nu endast två parametrar, och ges av

```
/* calculate_future_data: calculates future costs and balance
   for this year and the following lp->n_years_future years,
   given living parameters stored in *lp, and stores the
   calculated data in *lc */
void  calculate_future_data(living_parameter_type* lp, living_cost_type *lc)
```

Ovanstående exempel använder notationen `->` för att referera till elementen i en datapost. Denna notation används när man har en *pekare* till en datapost. Om man arbetar med själva dataposten, t.ex. om man direkt vill referera till de olika elementen i variabeln *lp* (vars deklaration beskrivs ovan), så använder man istället en *punkt*. Detta kan illustreras genom ett anrop till en funktion

```
/* n_years_to_debt_free: calculates number of years (expressed using a
```

```
   floating point value) until the debt is paid */
double n_years_to_debt_free(double price, double monthly_instalment)
{
    return (price * LOAN_P / 100) / (12*monthly_instalment);
}
```

som beräknar det antal år som återstår tills man blir skuldfri, givet ett lägenhetspris och en månatlig amortering.

Eftersom såväl lägenhetspriset som den månatliga amorteringen finns lagrade i variabeln *lp* så kan denna användas vid anropet, vilket då utförs enligt

```
printf("you will be debt free in %g years\n",
    n_years_to_debt_free(lp.price, lp.monthly_instalment));
```

där alltså en punkt används för att referera till datapostens olika element.

Dataposter kommer att användas i laboration 4 i kursen, och beskrivs ytterligare i kapitel 10 i [DEITEL].

Referenser

[1] Föreläsningsanteckning 6

-

[2] Föreläsningsanteckning 5

-