Język C – zajęcia nr 9

I. Typy pochodne – c.d. - wskaźniki

Wskaźniki

Zmienna typu *"wskaźnik na obiekt danego typu"* może zawierać **adres** obszaru pamięci zajmowanego przez pewien obiekt tego typu.

Operator wyłuskania * . Jeżeli wartością zmiennej **p** jest wskaźnik na obiekt **q** typu **T**, to wyrażenie ***p** przedstawia ten właśnie obiekt **q**.

```
int *p;         // Wyrażenie *p jest typu int,
                // czyli p jest typu wskaźnik na int
```

Wyrażenie ***p** może pojawić się w programie we wszystkich miejscach, w których może wystąpić obiekt **q**.

```
x = *p + y;     // W wyrażeniu
*p = a + b;     // Po lewej stronie operatora =
fun(*p);        // Jako argument wywołania funkcji
```

Wskaźniki na wskaźniki. Ponieważ wskaźnik na obiekt **q** sam jest pewnym obiektem w pamięci komputera, więc można utworzyć wskaźnik na wskaźnik na obiekt **q** typu **T**.

```
int **p;  // Wyrażenie **p równoważne *(*p) ma typ int,
          // czyli (*p) ma typ wskaźnik na int, czyli
          // p jest typu wskaźnik na wskaźnik na int
```

Uwaga: Definicja wskaźnika zawsze przydziela pamięć **jedynie dla przechowywania wskaźnika** (adresu), ale nie przydziela pamięci dla wskazywanego obiektu. Błędem jest korzystanie ze wskaźnika przed przypisaniem mu wskazania na jakiś istniejący obiekt.

Zmiennej wskaźnikowej można nadać wartość zero (NULL), co oznacza, że zmienna ta nie wskazuje na żaden obiekt.

Uwaga: W zapisie deklaracji wskaźnika nie musi występować spacja. Umieszczanie jej służy jedynie podniesieniu czytelności tekstu programu.

Równoważne zapisy deklaracji:

```
int * p;
int *p;
int* p;
int*p;
```

Arytmetyka wskaźników

Dopuszczalne operacje na wskaźnikach:

- suma wskaźnika i liczby całkowitej
- różnica wskaźników
- porównywanie wskaźników

Operacje wykonywane na wskaźnikach podlegają ograniczeniom:

- Nie można odejmować i porównywać wskaźników różnych typów.
- Argumentami operatora przypisania nie mogą być wskaźniki różnych typów.

Ogólna zasada: Jeżeli **p** jest wskaźnikiem na typ **T**, oraz **n** jest wartością całkowitą , to wyrażenie **p+n** wskazuje obszar pamięci odległy od adresu odpowiadającego **p** o liczbę bajtów **n*sizeof(T)**, gdzie **sizeof** jest operatorem dającym rozmiar danego typu.

```
char s[30];    // Tablica typu char
char *p, c;    // Wskaźnik na typ char, zmienna typu char
p=s;           // p wskazuje na początek tablicy s
p=&c;          // p wskazuje na zmienną c
```

Operator **&** daje wskazanie (adres) swojego argumentu. Jeżeli **x** jest pewnym obiektem, to wyrażenie **&x** jest adresem tego obiektu.

Operacja indeksowania jest przekształcana do postaci operacji wskaźnikowych. Wyrażenie **x[k]** jest równoważne wyrażeniu ***(x+k).**

```
int x[10];
*x=x[1]+x[2];  // Równoważne x[0]=x[1]+x[2]
*(x+3)=5**x;   // Równoważne x[3]=5*x[0]
```

UWAGA: W języku C nie ma automatycznego sprawdzania przekroczenia zakresów tablic:

```
int x[10];
x[13]=7;       // Błąd, przekroczenie zakresu tablicy
*(x+2)=*(x-1); // Błąd, przekroczenie zakresu tablicy
```

Związek tablic ze wskaźnikami:

```
#include <stdio.h>
int main()
{
    int i,x[]={3,7,11};
    printf("\n%d %d %d %d",2**x,2**x+2,2**(x+2),2*(*x+2));
}
```

```
6 8 22 10
```

Słowo kluczowe *void* *(w odniesieniu do wskaźników)*

Słowo void może być m.in. użyte:

- Jako typ bazowy pewnego wskaźnika oznacza uniwersalne beztypowe wskazanie adresowe. Wskaźnikowi typu void **można** przypisać wskazanie dowolnego typu. Na wskaźnikach typu void **nie można** wykonywać żadnych operacji arytmetycznych, **można** je natomiast porównywać między sobą i z zerem.

```
void *p;            // p jest wskaźnikiem beztypowym
```

Stałe wskaźniki i wskaźniki na stałe

Wskaźnik na obiekt pewnego typu może zostać zdefiniowany z modyfikatorem const na trzy sposoby jako:

Stały wskaźnik - nie można zmienić jego wartości, wskazuje zawsze na ten sam obszar w pamięci. Można natomiast dowolnie zmieniać wskazywaną przez ten wskaźnik wartość.

Wskaźnik na stałą - nie można zmieniać wskazywanej przez niego wartości. Można zmienić wartość samego wskaźnika w ten sposób, aby wskazywał inny obszar pamięci, ale również wówczas nie można zmienić wskazywanej wartości.

Stały wskaźnik na stałą - nie można zmieniać zarówno wartości wskaźnika, jak i wskazywanej przez niego wartości.

Stały wskaźnik:

```
int x=4;
int *const p=&x;
*p=3;               // Wartość wskazywaną przez p można
                    // zmieniać
p=NULL;             // Błąd, wartości p nie można zmieniać
```

Wskaźnik na stałą:

```
int x=12;
const int *p;
p=&x;               // Wartość p można zmieniać
*p=8;               // Błąd, wartości wskazywanej przez p
                    // nie można zmieniać
```

Stały wskaźnik na stałą:

```
int x=6,y=8;
const int *const p=&x;
p=&y;               // Błąd, wartości p nie można zmieniać
*p=7;               // Błąd, wartości wskazywanej przez p
                    // również nie można zmieniać
```

Wskaźniki na funkcje

Szczególnym rodzajem wskaźnika jest wskaźnik na funkcję, który w uproszczeniu można traktować jak adres początku tej funkcji. Równocześnie sama nazwa funkcji ma wartość adresu początku kodu stanowiącego obraz tej funkcji w pamięci.

Wywołanie pewnej funkcji poprzez wskaźnik p na tę funkcję można dokonać używając wyrażenia:

(**p*) (*lista_argumentów*)

Wskaźnik na funkcję musi być wcześniej zadeklarowany jako wskaźnik odpowiedniego typu, z którego wynika zarówno typ zwracanej przez funkcję wartości, jak i liczba oraz typ poszczególnych argumentów tej funkcji.

Deklaracje wskaźników na funkcje:

```
void (*p1)();                  //Wskaznik na funkcje bez
                               //argumentow i nie zwracajaca
                               //wartosci
int (*p2)(int,char);        //Wskaznik na funkcje o argumentach
                               //int i char, zwracajaca int
int *(*p3)(int);            //Wskaznik na funkcje o argumencie
                               //int, zwracajaca wskaznik na int
```

Uwaga: Operacje na wskaźnikach na funkcje podlegają dodatkowemu ograniczeniu: **Na wskaźnikach na funkcje nie można wykonywać operacji arytmetycznych**.

II. Operatory języka C – ciąg dalszy

Operator wywołania funkcji

Dwuargumentowym operatorem **wywołania funkcji** jest para nawiasów **( )**. Pierwszy argument ***arg1*** występuje z lewej strony pary nawiasów i ma wartość nazwy funkcji, a drugi argument ***arg2*** zawarty jest wewnątrz pary nawiasów i reprezentuje argumenty przekazywane do wywoływanej funkcji.

arg1 **(** ***arg2*** **)**

Drugi argument ***arg2*** operatora **( )** to lista argumentów wywołania funkcji. Pierwszym argumentem ***arg1*** musi być wyrażenie, którego wartością jest wywoływana funkcja. Najczęściej jest to wprost **nazwa funkcji** lub **wyłuskanie wskaźnika** na daną funkcję.

Wyrażenie:

(****p*** **) (** ***lista*** **)**

wywołuje funkcję, na którą wskazuje zmienna ***p***.

Uwaga: Występująca w wyrażeniu nazwa funkcji bez operatora jej wywołania (czyli bez pary nawiasów po nazwie funkcji) ma wartość stałego wskaźnika na początek tej funkcji.

Wprowadź i uruchom program, zinterpretuj kod źródłowy i wyniki:

```
#include <stdio.h>
int main()
{
int (*p)(int)=putchar;   //Definicja wskaznika p na funkcje
                         //o argumencie int i zwracajaca int
                         //i przypisanie mu adresu putchar
    (*p)('k');
    (*p)('o');
    (*p)('t');
     getch();
}
```

```
kot
```

Operator indeksowania

Dwuargumentowy operator indeksowania [ ] jest operatorem **sumowania**: wylicza sumę swoich argumentów i dostarcza wartości obszaru pomięci, na który wskazuje uzyskana suma. Jednym z argumentów musi być wyrażenie mające wartość wskaźnika, a drugim wyrażenie o wartości całkowitej. Wyrażenie postaci:

a [b]

ma wartość:

***(a+b)**

Operator indeksowania umożliwia między innymi dostęp do elementów tablicy. Ponieważ nazwa tablicy jest równocześnie stałym wskaźnikiem na jej początek, wartością wyrażenia:

tab [k]

jest wartość elementu o numerze **k** tablicy **tab**.

Uwaga: Z racji przemienności operacji indeksowania wyrażenie **`tab[k]`** jest w pełni równoważne wyrażeniu **`k[tab]`**, czyli **`x[5]`** ma tę samą wartość, co **`5[x]`**.

Operator adresu

Jednoargumentowy operator & zastosowany do l-wartości lub nazwy funkcji dostarcza adresu swojego argumentu. Wyrażenie postaci:

& *obiekt*

ma wartość adresu argumentu *obiekt*.

Operator wyłuskania

Jednoargumentowy operator wyłuskania ***** dostarcza wartości (zawartości) obszaru pamięci na który wskazuje argument operatora.

Jeżeli wartością zmiennej **p** jest wskaźnik na obiekt **q**, to wyrażenie ***p** ma wartość obiektu **q**.

Priorytety i wiązania operatorów:

(porównaj z tabelą priorytetów i wiązań zamieszczoną w materiałach C03)

Priorytet	Operator	Wiązanie	Nazwa operatora
1	**( )**	Lewostronne	Wywołanie funkcji
	[ ]	Lewostronne	Indeksowanie
2	*****	Prawostronne	Wyłuskanie
	&	Prawostronne	Pobranie adresu

Zadanie C09-1.

Modyfikacja programu C06-1. Wczytaj z klawiatury ciąg *n* liczb rzeczywistych (zakładamy że *n* jest wcześniej podane przez użytkownika i *n* <=100) , po czym wypisz te liczby w kolumnie w kolejności odwrotnej do wczytywania.

Wymóg: Zadeklaruj tablicę, ale w instrukcjach **nie korzystaj z odwołania do elementów tablicy za pomocą operatora indeksowania [ ], wykorzystaj wskaźniki i operator wyłuskania *.**

Kod źródłowy programu C09-1 wraz z oknem zawierającym efekty przykładowego uruchomienia tego programu, należy zapisać w **jednym** dokumencie (**format Word 2003**) o nazwie **C09.doc** i przesłać za pośrednictwem platformy Moodle w wyznaczonym terminie.