0.0.1 Arytmetyka na wskaźnikach

Działania arytmetyczne na wskaźnikach są ograniczone do dodawania, odejmowania i porównań.

Operacje arytmetyczne na wskaźnikach typu „wskaźnik do ***typ***u" automatycznie uwzględniają rozmiar ***typ***u tj. ilość bajtów potrzebną do przechowywania obiektu typu ***typ*** w pamięci.

Wewnętrzna arytmetyka wykonywana na wskaźnikach zależy od modelu pamięci, w którym działamy i ew. obecności modyfikatorów wskaźników (zmieniających „długość" wskaźników.

Przy wykonywaniu operacji arytmetycznych na wskaźnikach, zakłada się, że wskaźnik wskazuje na tablicę obiektów. Gdy mamy deklarację wskaźnika wskazującego na ***typ*** to dodanie wartości całkowitej n do wskaźnika „przesuwa" go tak by wskazywał o n obiektów typu ***typ*** dalej. Jeżeli ***typ*** ma rozmiar 10 bajtów i dodamy 5 do wskaźnika wskazującego na ***typ*** to wskaźnik będzie wskazywał miejsce w pamięci przesunięte o 5 obiektów dalej, czyli o 50 bajtów dalej.

Wynik różnicy jest ilością elementów tablicy dzielących dwa elementy wskazywane przez odejmowane wskaźniki. Na przykład gdy *ptr1* wskazuje na trzeci element tablicy a *ptr2* na dziesiąty to różnica *ptr2 – ptr1* wyniesie 7.

Różnica między wskaźnikami ma sens jedynie wtedy gdy oba wskaźniki wskazują na tą samą tablicę.

Gdy dodajemy lub odejmujemy od „wskaźnika do ***typ***u" liczbę całkowitą to wynik też jest typu „wskaźnik do ***typ***u".

Nieformalnie, możemy myśleć o dodawaniu `P + n` jak o „przesuwaniu" wskaźnika o `n * sizeof(typ)` bajtów, pod warunkiem, że wskaźnik znajduje się w dopuszczalnym zakresie (do pierwszego tj. zerowego, elementu do następnego za ostatnim).

Odejmowanie dwóch wskaźników, wskazujących elementy z tej samej tablicy, daje w wyniku liczbę całkowitą typu `ptrdiff_t` zdefiniowanego w stddef.h. Wartość ta przedstawia różnicę pomiędzy indeksami elementów, na które wskazują wskaźniki, pod warunkiem, że leży (mieści się) w zakresie `ptrdiff_t`. W wyrażeniu `P1 − P2`, gdzie `P1` i `P2` są typu wskaźnik do ***typ***u (lub wskaźnik do typu dającego się zakwalifikować jako ***typ***) `P1` i `P2` muszą wskazywać jakiś istniejący element lub następny za ostatnim. Gdy `P1` wskazuje `i`-ty element a `P2` `j`-ty to `P1 − P2` ma wartość `i − j`.

0.0.2 Konwersja wskaźników

Wskaźniki do jakiegoś typu danych mogą zostać zamienione (poddane konwersji) na wskaźniki do innego typu danych za pomocą mechanizmu rzutowania:

```
char *str;

int *ip;

str = (char *) ip;
```

Ogólnie, rzutowanie (***typ****) zamieni dowolny wskaźnik na „wskaźnik do ***typ***u".

0.0.3 Wskaźniki do funkcji

Wskaźnik do funkcji jest jej adresem (położeniem w pamięci) – zazwyczaj w segmencie, w którym przechowywany jest kod wykonywalny funkcji – tj. adres, do którego przekazywane

jest sterowanie programem gdy funkcja zostanie wywołana.

Wskaźnik do funkcji ma typ „wskaźnik do funkcji zwracającej `typ`", gdzie typ jest `typem` zwracanym przez funkcję. Na przykład

```
void (*func)();
```

W C++ jest to wskaźnik do funkcji nie mającej argumentów i zwracającej typ `void` (pusty – efektywnie to samo co procedura w Pascal'u). W C jest to wskaźnik do funkcji pobierającej nieznaną liczbę argumentów i zwracającą `void`. W poniższym przykładzie

```
void (*func)(int);
```

`*func` jest wskaźnikiem do funkcji pobierającej argument typu `int` i zwracającej `void`.

W C++ taki wskaźnik może być użyty by uzyskać dostęp do składowych funkcji statycznych (static). Wskaźnik do innych składowych klas muszą używać specjalnych operatorów „wskaźnika-do-składowej".

0.1 Tablice, struktury i unie

0.1.1 Tablice

Deklaracja
type deklarator [<wyrażenie-stałe>]
deklaruje tablicę (ang. array) składającą się z elementów ***typ***u. Tablica jest ciągłym kawałkiem pamięci, o pojemności dokładnie takiej jaka jest potrzebna na przechowywanie wszystkich elementów

```
int days[7] = {1, 1, 1, 1, 1, 1, 1};
```

czyli

0	1	2	3	4	5	6
1	1	1	1	1	1	1

Indeksy elementów tablicy w C zaczynają się **zawsze od 0**. Inny przykład

```
char name[] = {"Unknown"};
```

tu kompilator utworzy ośmio-elementową tablicę znaków, której elementy przyjmą wartości `'U'` dla `name`[0], `'n'` dla `name`[1] itd. Tablice znaków rozumiane jako tekst (łańcuch ang. *string*), a nie jako jednobajtowe liczby całkowite, kończą się specjalnym znakiem tzw. znacznikiem końca łańcucha, oznaczanego jako NULL – binarne 0 lub `'\0'`. W konsekwencji każdy łańcuch tekstowy, np. taki jak w powyższym przykładzie, oprócz widocznych liter zawiera dodatkowo jeden bajt – znacznik końca łańcucha. Tekst składa się z 7-miu znaków, ale tablica zajmuje 8.

0	1	2	3	4	5	6	7
U	n	k	n	o	w	n	'\0'

W przypadku tablic innych typów podobnych znaczników nie stosuje się.

W języku C identyfikatory tablic są jednocześnie wskaźnikami do pierwszego (dokładniej zerowego) elementu tablicy. Dla deklaracji

```
int tabl[5];
char *adr = (char*) tabl;
```

mamy

tabl[0]	tabl[1]	tabl[2]	tabl[3]	tabl[4]
tabl	tabl + 1	tabl + 2	tabl + 3	tabl + 4
adr	adr + 4	adr + 8	adr + 12	adr + 16

przy założeniu, że typ `int` zajmuje 4 a `char` 1 bajt. W przypadku tablicy dwuwymiarowej

```
int tabl[3][5];
char *adr = (char*) tabl;
```

będziemy mieli elementy

	tabl [][0]	tabl [][1]	tabl [][2]	tabl [][3]	tabl [][4]
tabl [0][]	tabl [0][0]	tabl [0][1]	tabl [0][2]	tabl [0][3]	tabl [0][4]
tabl [1][]	tabl [1][0]	tabl [1][1]	tabl [1][2]	tabl [1][3]	tabl [1][4]
tabl [2][]	tabl [2][0]	tabl [2][1]	tabl [2][2]	tabl [2][3]	tabl [2][4]

lub

	tabl [][0]	tabl [][1]	tabl [][2]	tabl [][3]	tabl [][4]
tabl [0][]	tabl	tabl + 1	tabl + 2	tabl + 3	tabl + 4
tabl [1][]	tabl + 5	tabl + 6	tabl + 7	tabl + 8	tabl + 9
tabl [2][]	tabl + 10	tabl + 11	tabl + 12	tabl + 13	tabl + 14

a z punktu widzenia „czystego” bloku pamięci

	tabl [][0]	tabl [][1]	tabl [][2]	tabl [][3]	tabl [][4]
tabl [0][]	adr	adr + 4	adr + 8	adr + 12	adr + 16
tabl [1][]	adr + 20	adr + 24	adr + 28	adr + 32	adr + 36
tabl [2][]	adr + 40	adr + 44	adr + 48	adr + 52	adr + 56

W przypadku tablic o większej ilości wymiarów zachowanie indeksów i „gospodarka” pamięcią jest podobna. W rzeczywistych programach tablice dwu i więcej wymiarowe tworzy się inaczej, co pokażemy dalej na przykładzie mnożenia macierzy.