Programowanie obiektowe

Laboratorium 2

Ćwiczenie 1

Nazwisko i imię:Tobolski Tymon
Nr indeksu: 181037

Prowadzący: Michał Kucharzak
Grupa: WT/TP 13:15 - 15:00

Ad 1.
Typ `void *` jest nadrzędnym typem wszystkich typów wskaźnikowych. Przechowuje on informacje o adresie ale nie o rozmiarze (jak w przypadku innych typów wskaźnikowych). Wykorzystywany jest m.in. w funkcjach mogących przyjmować dowolny typ parametru. Dla przykładu funkcja `qsort`:

```
void qsort(void * base, size_t num, size_t size,
     int (* comparator)(const void *, const void *));

int comp(const void * a, const void * b){
    return (*(int*)a - *(int*)b);
}
```

Funkcja `qsort` przyjmuje jako parametr wskaźnik do funkcji porównującej, dzięki czemu jest uniwersalna (działa dla każdego typu). Funkcja porównująca przyjmuje argumenty typu `const void *` więc aby porównać dwa elementy trzeba je najpierw rzutować na (w tym przykładzie) `int*` a następnie wyłuskać ich wartość poprzez operator `*`.

Ad 2.

```
char * wskCh = "Tekst";
cout << (void *)wskCh << endl;
```

Wymagana jest konwersja na `void *`.

Ad 3.
Przesyłanie argumentu do funkcji przez referencję lub wskaźnik ma na celu możliwość operacji na zmiennej niedostępnej wewnątrz funkcji. Zmiennej referencyjnej można używać tak jak normalnej zmiennej, natomiast zmienna wskaźnikowa przechowuje jedynie adres innej zmiennej, tak więc konieczne jest użycie operatorów wskaźnikowych (`wsk->metoda()` zamiast `ref.metoda()` itp.) Ponadto, w przeciwieństwie do zmiennej wskaźnikowej, zmienna referencyjna nie może zmienić swojej wartości - raz ustawiona zawsze będzie wskazywać na ten sam obiekt.

Ad 4.

```
int T[10];
int size = sizeof(T)/sizeof(int);
```

`sizeof(T)` - zwraca rozmiar całej tablicy w bajtach
`sizeof(int)` - zwraca rozmiar jednego elementu

Ad 5.

```
double tab[5] = {1.1, 2.2, 3.3, 4.4, 5.5};
```

`tab+1` - adres drugiego elementu
`tab+1` - drugi element tab plus jeden
`&tab+1` - adres wskaźnika na tab[5] plus jeden, czyli 40 bajtów dalej
`*(tab+1)` - drugi element tab
`tab[0]+1` - pierwszy element plus jeden
`&tab[0]+1` - adres drugiego elementu
`tab++` - błąd kompilacji

Ad 6.

```
double tab[5] = {1.1, 2.2, 3.3, 4.4, 5.5};
double *w = tab;
```

`w+1` - *adres drugiego elementu tablicy tab*
`w+=1` - *przesuniecie wskaznika o jedna pozyjce (na nastepny element tablicy)*
`*w+1` - *pobranie elementu wskazanego przez w i dodanie do niego 1*
`&w+1` - *adres nastepnej komorki pamieci po adresie w*
`*(w+1)` - *drugi element tablicy*
`w[0]` - *pierwszy element tablicy plus jeden*
`&w[0] + 1` - *adres drugiego elementu tablicy*
`w++` - *przesuniecie wskaznika o jedna pozycje (na nastepny element tablicy)*
`*(&w[2]-1)` - *drugi element tablicy*

Ad 7.

```
int a[] = {1,6};
int * pa = &a[0];
int * pb = &a[1];
```

Porównanie wskaźników ma sens jedynie dla wskaźników tej samej tablicy. Jeżeli `pa < pb` to znaczy ze `pa` wskazuje na element o mniejszym indeksie niż `pb`. Jeżeli `pa == pb` to znaczy ze oba wskaźniki wskazują na ten sam adres w pamięci.

Dodawanie dwóch wskaźników do siebie nie ma większego sensu.
Odejmowanie pozwala określić jak "daleko od siebie" są umieszczone elementy wskazywane przez wskaźniki np.`pa-pb` = 1 (`pb` wskazuje na jeden element dalej niż `pa`).

Dodawanie (i odejmowanie) liczby całkowitej do wskaźnika przesuwa jego wartość o liczba*rozmiar_wskaźnika, np:

dla wskaźnika `double*p p+1` przesunie o 8 bajtów
dla wskaźnika `char*p p+1` przesunie o 2 bajty

Ad 8.

```
int *p = 0;
p = (int*)&p;
```

Można ustawić wskaźnik sam na siebie, wymagane jest jednak rzutowanie adresu wskaźnika na `int*` .

Ad 9.

```
double (*(*(*T[11])(char * const))[])[]
```

Ad 10.

```
double (*fun())();
```