Notatki z Programowania Proceduralnego w C – w prostym ujęciu

1. Wskaźniki i operacje na wskaźnikach w języku C

Czym jest wskaźnik?

- Wskaźnik (ang. *pointer*) to **zmienna**, która przechowuje **adres** innej zmiennej w pamięci.
- Dzięki wskaźnikom możemy w prosty sposób manipulować danymi, przekazywać duże struktury do funkcji bez kopiowania, czy zarządzać pamięcią dynamicznie.

Deklaracja i podstawowe użycie

- Deklaracja wskaźnika:

```
int *ptr; // ptr to wskaźnik na int
```

  Tutaj `ptr` będzie mógł przechowywać adres zmiennej typu `int`.
- Aby przypisać adres do wskaźnika, używamy operatora `&` (adresu):

```
int x = 10;
ptr = &x; // teraz ptr "wskazuje" na x
```

- Aby odczytać wartość spod adresu, na który wskaźnik wskazuje, używamy operatora dereferencji `*`:

```
printf("%d\n", *ptr); // wyświetla 10
```

Operacje na wskaźnikach

1. **Inkrementacja/dekrementacja**:
   Jeśli mamy `int *p;`, to `p++` spowoduje przejście wskaźnika o rozmiar typu, na który wskazuje (np. zwykle 4 bajty dla `int` w popularnych systemach).
   Analogicznie `p--`, `p += 2`, `p -= 3` itp.
2. **Porównywanie**:
   `if (p == q)`, sprawdzamy czy wskaźniki `p` i `q` wskazują na **ten sam** adres.
3. **Dodawanie/odejmowanie liczby całkowitej**:
   `p + 3` oznacza przesunięcie wskaźnika o 3 pozycje w pamięci typu `int` (czyli faktycznie 3 * rozmiar `int` bajtów).
4. **Różnica między dwoma wskaźnikami**:
   `p - q` zwraca liczbę elementów (w jednostkach typu, np. `int`) znajdujących się między adresami przechowywanymi w `p` i `q` (o ile obydwa leżą w tej samej tablicy).

Przykładowe pytania i odpowiedzi

1. **Pytanie:** Czym jest wskaźnik w C i jak go zadeklarować?
   Odpowiedź: Wskaźnik to zmienna przechowująca adres innej zmiennej. Deklarujemy np. `int *ptr;` – wskaźnik na `int`.

2. **Pytanie:** Do czego służy operator `*` przy wskaźniku?
   Odpowiedź: Operator `*` (dereferencja) pozwala na dostęp do wartości znajdującej się pod adresem przechowywanym w wskaźniku.

---

2. Wskaźnik podwójny, tablice wskaźników w języku C

Wskaźnik do wskaźnika

- **Wskaźnik podwójny** (`int **pp`) to zmienna, która przechowuje adres innego wskaźnika (`int *`).
- Pozwala to m.in. na modyfikację wskaźnika w funkcji, gdy przekazujemy `int **` jako argument (tzw. przekazywanie wskaźnika do wskaźnika).

Przykład:

```
int x = 10;
int *px = &x;
int **ppx = &px; // ppx przechowuje adres wskaźnika px
```

Tablice wskaźników

- Gdy deklarujemy np. `int *arr[5];`, tworzymy tablicę 5 wskaźników do `int`.
- Każdy element tej tablicy może przechowywać adres innej zmiennej typu `int`.

  ```
  int a=1, b=2, c=3;
  int *ptrArr[3] = { &a, &b, &c };
  ```

- Często używane, np. do przechowywania **tablic łańcuchów znakowych**.

Przykładowe pytania i odpowiedzi

1. **Pytanie:** Po co nam wskaźnik do wskaźnika?
   Odpowiedź: Umożliwia on zmianę wartości wskaźnika przekazanego do funkcji lub zarządzanie tablicami wskaźników. Mamy wtedy dostęp nie tylko do wartości, lecz także do samego adresu wskaźnika.

2. **Pytanie:** Jak zadeklarować tablicę 5 wskaźników do typu `float`?
   Odpowiedź: `float *arr[5];`

---

3. Dynamiczna alokacja pamięci w języku C. Wycieki pamięci

Podstawy dynamicznej alokacji

- Funkcje standardowe do rezerwowania i zwalniania pamięci:
    1. `malloc(size_t size)` – rezerwuje **blok** pamięci o wielkości `size` bajtów, zwraca wskaźnik typu `void *`.
    2. `calloc(size_t n, size_t size)` – rezerwuje pamięć na `n` elementów po `size` bajtów każdy i dodatkowo pamięć ta jest **wyzerowana**.
    3. `realloc(void *ptr, size_t new_size)` – zmienia rozmiar już zaalokowanej pamięci (np. powiększa tablicę).
    4. `free(void *ptr)` – zwalnia pamięć wskazywaną przez `ptr`.

Przykład użycia `malloc`:

```
int *tab = malloc(10 * sizeof(int));
if (tab == NULL) {
    // Obsługa błędu - np. brak pamięci
}

// ... korzystamy z tablicy 'tab' ...

free(tab); // zwalniamy pamięć
```

Wycieki pamięci (memory leaks)

- **Wyciek pamięci** następuje, gdy przydzielimy pamięć dynamicznie (np. przez `malloc`), ale **nie** zwolnimy jej (`free`) przed utratą możliwości dostępu do tego wskaźnika.
- Powoduje to ciągły wzrost użycia pamięci przez program, ponieważ „zgubiona” pamięć nie może być ponownie użyta.

Przykładowe pytania i odpowiedzi

1. **Pytanie:** Czym różni się `malloc` od `calloc`?
   Odpowiedź: `malloc` przydziela blok pamięci bez inicjalizacji, `calloc` natomiast przydziela pamięć i **zeruje** wszystkie bajty w tym bloku.

2. **Pytanie:** Co to jest wyciek pamięci i jak go uniknąć?
   Odpowiedź: To sytuacja, w której pamięć została zaalokowana dynamicznie, ale nie zwolniona. Należy pamiętać, aby każdą zaalokowaną pamięć (np. przez `malloc`, `calloc`) zwalniać za pomocą `free`.

---

4. Łańcuchy znakowe w języku C

Podstawowe informacje

- Łańcuch znakowy w C (tzw. **string**) to tablica typu `char` zakończona **znakiem** `'\0'` (wartość zero).
- Można go tworzyć na dwa sposoby:
    1. **Statycznie** (np. `char napis[6] = "Hello";` – tu trzeba zarezerwować miejsce na `'\0'`).

2. **Jako wskaźnik** (np. `char *napis = "Hello";`) – w tym przypadku `napis` wskazuje na miejsce w pamięci, w którym przechowywany jest ten dosłowny napis.

Pamiętaj, że napis zadeklarowany jako `char *napis = "Hello";` może być tylko do odczytu (w wielu kompilatorach), więc nie można w nim zmieniać znaków.

Podstawowe funkcje do obsługi łańcuchów (z biblioteki `<string.h>`)

1. `strlen(const char *s)` – długość łańcucha (nie liczy `'\0'`).
2. `strcpy(char *dest, const char *src)` – kopiuje napis `src` do `dest`.
3. `strcat(char *dest, const char *src)` – dokleja na koniec `dest` napis `src`.
4. `strcmp(const char *s1, const char *s2)` – porównuje leksykograficznie dwa łańcuchy, zwraca wartość ujemną, 0 lub dodatnią.

Przykładowe pytania i odpowiedzi

1. **Pytanie:** Do czego służy znak `'\0'` w łańcuchach znakowych?
   Odpowiedź: To znak końca łańcucha, wskazuje gdzie napis się kończy (o wartości 0 w kodzie ASCII).

2. **Pytanie:** Jak skopiować napis `src` do bufora `dest`?
   Odpowiedź: Można użyć funkcji `strcpy(dest, src)` z biblioteki `<string.h>`.

---

5. Rekurencja w języku C

Na czym polega rekurencja?

- **Rekurencja** to sytuacja, w której funkcja **wywołuje samą siebie** w swoim ciele.
- Aby uniknąć nieskończonego wywołania, musi istnieć **warunek bazowy** (punkt, w którym rekurencja przestaje się pogłębiać).

Przykład rekurencyjnego obliczania silni (n!):

```
int factorial(int n) {
    if (n == 0) {
        return 1; // warunek bazowy
    }
    return n * factorial(n - 1);
}
```

Gdzie spotyka się rekurencję?

- Obliczanie ciągu Fibonacciego.
- Przeszukiwanie drzew i struktur hierarchicznych.
- Metoda „dziel i zwyciężaj” (divide and conquer), np. sortowanie szybkie (quick sort).

Przykładowe pytania i odpowiedzi

1. **Pytanie:** Dlaczego musimy mieć warunek bazowy w funkcji rekurencyjnej?
   Odpowiedź: Bez warunku bazowego funkcja wywoływałaby się w nieskończoność, powodując przepełnienie stosu (tzw. stack overflow).

2. **Pytanie:** Jak w C wygląda przykładowa implementacja funkcji do liczenia silni?
   Odpowiedź:

```c
int factorial(int n) {
    if (n == 0) return 1;
    return n * factorial(n - 1);
}
```

---

6. Pliki w języku C. Pliki tekstowe i binarne. Dostęp sekwencyjny i swobodny

Otwieranie i zamykanie plików

- Do pracy z plikami używamy wskaźnika do struktury `FILE`, np. `FILE *fp;`.
- Otwieramy plik za pomocą `fopen("nazwa_pliku", "tryb");`.
- Zamykamy plik wywołaniem `fclose(fp);`.

Tryby otwarcia pliku (najpopularniejsze)

- `"r"` – odczyt tekstowy (plik musi istnieć).
- `"w"` – zapis tekstowy (tworzy plik lub nadpisuje istniejący).
- `"a"` – dopisanie na końcu w trybie tekstowym.
- `"rb"`, `"wb"`, `"ab"` – to samo co wyżej, ale w **trybie binarnym**.
- `"r+"`, `"w+"`, `"a+"` – tryby do odczytu i zapisu w trybie tekstowym.
- `"rb+"`, `"wb+"`, `"ab+"` – tryby do odczytu i zapisu w trybie binarnym.

Odczyt i zapis

- **Tekstowy**: `fprintf`, `fscanf`, `fgets`, `fputs`, `fgetc`, `fputc`.
- **Binarny**: `fwrite`, `fread`.

Dostęp sekwencyjny i swobodny

- **Dostęp sekwencyjny** – odczyt/zapis „po kolei” od początku do końca pliku.
- **Dostęp swobodny (losowy)** – możemy przeskakiwać w pliku dzięki funkcjom `fseek`, `ftell`, `rewind`.

Przykładowe pytania i odpowiedzi

1. **Pytanie:** Jak otworzyć plik do zapisu w trybie tekstowym tak, by nie nadpisywać istniejących danych, lecz dopisywać nowe na końcu?
   Odpowiedź: Należy użyć `fopen("nazwa.txt", "a");`.

2. **Pytanie:** Czym różni się dostęp sekwencyjny od dostępu swobodnego?
   Odpowiedź: Sekwencyjny czyta dane po kolei. Swobodny umożliwia ustawianie wskaźnika w dowolnym

miejscu w pliku (`fseek`).

---

7. Wskaźniki do funkcji

Co to jest wskaźnik do funkcji?

- To **zmienna**, która przechowuje **adres** jakiejś funkcji w pamięci.
- Dzięki temu możemy wywoływać funkcję przez jej wskaźnik, np. przekazywać funkcję jako argument do innej funkcji (tzw. **callback**).

Deklaracja wskaźnika do funkcji

```
// Wskaźnik do funkcji zwracającej int i przyjmującej (int, int):
int (*fun_ptr)(int, int);
```

- Potem możemy przypisać konkretną funkcję o odpowiednim prototypie:

```
int dodaj(int a, int b) {
    return a + b;
}

fun_ptr = dodaj;
```

- Wywołanie przez wskaźnik:

```
int wynik = fun_ptr(2, 3); // lub (*fun_ptr)(2, 3);
```

Przykładowe pytania i odpowiedzi

1. **Pytanie:** Jak wywołujemy funkcję przez jej wskaźnik `fun_ptr`?
   Odpowiedź: Możemy użyć `fun_ptr(args)` lub `(*fun_ptr)(args)`.

2. **Pytanie:** Jak zadeklarować wskaźnik do funkcji, która nie przyjmuje argumentów i nic nie zwraca (void)?
   Odpowiedź: `void (*fun_ptr)(void);`

---

8. Struktura programu w języku C. Klasy zmiennych. Czas trwania zmiennych, zasięg i łączność

Struktura programu

- Program w C składa się zwykle z **pliku źródłowego** (`.c`) i ewentualnych **plików nagłówkowych** (`.h`).
- Uruchamianie programu zaczyna się od funkcji `main()`.

- W plikach nagłówkowych często przechowuje się **deklaracje** funkcji czy struktur, a w plikach `.c` – **definicje** (implementacje).

Klasy zmiennych (klasy pamięci)

1. `auto` – domyślna dla zmiennych lokalnych w funkcji; żyją do momentu opuszczenia bloku.
2. `static` – zmienna statyczna zachowuje swoją wartość między wywołaniami funkcji i ma stałe miejsce w pamięci przez cały czas działania programu.
3. `extern` – informuje, że zmienna jest zadeklarowana w innym pliku (odwołanie zewnętrzne).
4. `register` – sugeruje kompilatorowi, żeby trzymał zmienną w rejestrze procesora (optymalizacja; w praktyce współczesne kompilatory i tak same optymalizują).

Czas trwania, zasięg, łączność

- **Czas trwania** (lifetime) – okres, przez który zmienna istnieje w pamięci (np. zmienna lokalna `auto` istnieje tylko w czasie wykonywania danej funkcji).
- **Zasięg** (scope) – obszar kodu, w którym zmienna jest dostępna (np. blok `{ ... }`, plik, przestrzeń globalna).
- **Łączność** (linkage) – określa, czy identyfikator (np. zmienna globalna) jest widoczny tylko w danym pliku (wewnętrzna) czy też w całym projekcie (zewnętrzna).

Przykładowe pytania i odpowiedzi

1. **Pytanie:** Jakie są klasy pamięci zmiennych w C?
   Odpowiedź: `auto`, `static`, `extern`, `register`.

2. **Pytanie:** Jaka jest różnica między zmienną globalną o zasięgu plikowym a zmienną lokalną?
   Odpowiedź: Zmienna globalna zasięgu plikowego (np. zadeklarowana jako `static int g;` na poziomie pliku) jest widoczna w całym pliku, ale **nie** w innych plikach. Z kolei zmienna lokalna istnieje i jest widoczna tylko wewnątrz funkcji (lub bloku) i przestaje istnieć po jej zakończeniu.

Podsumowanie

Powyższe notatki w przystępny sposób opisują najważniejsze zagadnienia z programowania proceduralnego w C:

- **Wskaźniki** (podstawowe, podwójne, tablice wskaźników),
- **Alokację pamięci** i zarządzanie nią,
- **Pracę z łańcuchami znakowymi**,
- **Rekurencję**,
- **Pracę z plikami** (tekstowe i binarne, dostęp sekwencyjny i swobodny),
- **Wskaźniki do funkcji**,
- **Strukturę programu** w C, klasy zmiennych i zasady widoczności (zasięg, łączność).