Podstawy programowania.
Wykład 13

Projektowanie struktur danych

Struktury, wskaźniki i lista powiązana

- Lista powiązana (*linked list*) jest strukturą danych zawierającą elementy, tworzące węzły listy, zawierające dowolne dane
  - lista powiązana jest strukturą dynamiczną, liczba elementów listy może się zmieniać w trakcie wykonania programu
  - dostęp do elementów listy wykorzystuje zasadę, że każdy element listy wskazuje na element następny
  - najważniejszym elementem listy jest jej "głowa" – uchwyt (wskaźnik) do elementu początkowego
    - "głowa" jest reprezentacją listy w funkcjach korzystających z listy
    - chcąc uzyskać dostęp do dowolnego elementu listy należy uzyskać dostęp do jej "głowy", a następnie odwiedzać kolejne węzły (korzystając ze wskaźników w celu przejścia od jednego węzła do kolejnego), aż do znalezienia szukanego elementu
    - inicjowanie listy polega na nadaniu wartości NULL "głowie" listy
  - końcem listy jest węzeł, który nie wskazuje na żaden kolejny węzeł

Struktury, wskaźniki i lista powiązana

- Lista powiązana (*linked list*)
  - struktury są wygodnym typem danych do przechowywania zawartości elementów listy:

```
typedef struct element_listy{
    char* nazwa_wezla;
    // dowolne inne składniki pojedynczego elementu listy
    wsk_el_list nastepny_wezel; // wskaźnik do kolejnego elementu
} el_list;
```

  - wskaźnik do następnego elementu może posługiwać się typem:

```
typedef struct element_listy* wsk_el_list;
```

  - pojedyncze elementy listy są obiektami typu el_list:

```
el_list element_1 = { "Węzeł 1", NULL };
```

  - operacje na elementach mogą posługiwać się wskaźnikami:

```
el_list* element_1_wsk = &element_1;
```

Struktury, wskaźniki i lista powiązana

- Lista powiązana (*linked list*)
  - inicjowanie pustej listy:

```
el_list* Glowa = NULL;
```

    - zmiana liczby elementów listy polega na wstawianiu kolejnych elementów i ich usuwaniu
  - węzeł listy można wstawić w dowolne miejsce: na początek (zmieniając wartość wskaźnika będącego "głową" listy), w środek (pomiędzy dwa węzły) lub na koniec
  - przeglądanie kolejnych elementów listy:

```
el_list* Element_wsk=Glowa;
while(Element_wsk!=NULL){
   // dostęp do zawartości poprzez Element_wsk->....
   Element_wsk = Element_wsk->nastepny_wezel;
}
```

Struktury, wskaźniki i lista powiązana

- Lista powiązana (*linked list*)
  - miejsce w pamięci dla elementów listy można alokować dynamicznie

```
el_list* element_wsk = malloc( sizeof(el_list) );
```

  - wstawienie i usunięcie węzła polega zawsze na modyfikacji powiązań pomiędzy węzłami, np.:

```
int wstaw_na_poczatek( // funkcja zwraca kod sukcesu lub błędu
  el_list** Glowa_wsk,   // lista - identyfikowana przez wskaźnik
  el_list* Element_wsk  // wskaźnik do elementu wstawianego na listę
){
  // na początku: obsługa błędów danych wejściowych; następnie:
  Element_wsk->nastepny_wezel=*Glowa_wsk;
  *Glowa_wsk=Element_wsk; // modyfikacja Głowy używając *Głowa_wsk
  return(0); }
```

  - wywołanie funkcji:

```
wstaw_na_poczatek( &Glowa, element_wsk );
```

Struktury, wskaźniki i lista powiązana

- Lista powiązana (*linked list*)
  - niektóre z operacji na liście mogą wymagać przeglądania kolejnych elementów listy:

```
int usun_element_listy( // funkcja zwraca kod sukcesu lub błędu
    el_list** Glowa_wsk,   // lista - identyfikowana przez wskaźnik
    el_list*  Element_wsk  // wskaźnik do elementu usuwanego z listy
) {
 // na początku: obsługa błędów danych wejściowych
 if(*Glowa_wsk==Element_wsk)  *Glowa_wsk = Element->nastepny_wezel;
 else{
      el_list* aktualny_wezel_wsk = *Glowa_wsk;
      while(aktualny_wezel_wsk->nastepny_wezel != Element_wsk){
                aktualny_wezel_wsk = aktualny_wezel_wsk->nastepny_wezel;
      }
      aktualny_wezel_wsk->nastepny_wezel = Element->nastepny_wezel;
 }  // funkcja może także zwalniać pamięć jeśli była zaalokowana
 return 0; }
```

Przykład złożonej struktury danych

```
// definicje struktur w pakiecie mg – moja grafika – wersja 1
typedef struct mg_punkt_2D{ // punkt w przestrzeni 2D
  double   x;        // współrzędna x
  double   y;      // współrzędna y
} mg_punkt_2D;

typedef struct mg_krawedz_2D{ // krawędź w przestrzeni 2D
  mg_punkt_2D koniec[2]; // punkty będące wierzchołkami
} mg_krawedz_2D;

typedef struct mg_trojkat_2D{ // trójkąt w przestrzeni 2D
  mg_krawedz_2D krawedzie[3];    //  krawędzie trójkąta
} mg_trojkat_2D;
```

- Uwaga: nie ma punktów wspólnych dla krawędzi, krawędzi wspólnych dla trójkątów itp.
- Uwaga: zmiana współrzędnych punktów nie zmienia położenia krawędzi

Przykład złożonej struktury danych

```
// definicje struktur w pakiecie mg – moja grafika – wersja 2
typedef struct mg_punkt_2D{ // punkt w przestrzeni 2D
  double   x;        // współrzędna x
  double   y;      // współrzędna y
} mg_punkt_2D;

typedef struct mg_krawedz_2D{ // krawędź w przestrzeni 2D
  mg_punkt_2D* koniec[2]; // punkty będące wierzchołkami
} mg_krawedz_2D;

typedef struct mg_trojkat_2D{ // trójkąt w przestrzeni 2D
  mg_krawedz_2D* krawedzie[3];    //  krawędzie trójkąta
} mg_trojkat_2D;
```

- Uwaga: mogą istnieć punkty wspólne dla krawędzi, krawędzie wspólne dla trójkątów itp.
- Uwaga: zmiana współrzędnych punktów zmienia położenie krawędzi

Przykład złożonej struktury danych

```
// definicje struktur w pakiecie mg – moja grafika – wersja 2.1
typedef struct mg_punkt_2D{        // punkt w przestrzeni 2D
  double   x;      // współrzędna x
  double   y;      // współrzędna y
  struct mg_trojkat_2D* *trojkaty;   // trójkąty zawierające punkt
                     // tablica o długości nieznanej w trakcie kompilacji
} mg_punkt_2D;

typedef struct mg_krawedz_2D{      // krawędź w przestrzeni 2D
  struct mg_punkt_2D* koniec[2];     // punkty będące wierzchołkami
  struct mg_trojkat_2D* trojkaty[2];   // trójkąty zawierające krawędź
} mg_krawedz_2D;

typedef struct mg_trojkat_2D{        // trójkąt w przestrzeni 2D
  struct mg_krawedz_2D* boki[3];     // krawędzie (boki) trójkąta
  struct mg_trojkat_2D* sasiedzi[3];  // sąsiednie trójkąty
} mg_trojkat_2D;
```

Przykład złożonej struktury danych

- Definicja struktury obejmującej całą siatkę trójkątów

```
typedef struct mg_siatka_2D{
  int liczba_punktow;
  mg_punkt_2D* tablica_punktow;
  int liczba_krawedzi;
  mg_krawedz_2D* tablica_krawedzi;
  int liczba_trojkatow;
  mg_trojkat_2D* tablica_trojkatow;
} mg_siatka_2D;
```

- Parametry siatki (liczba punktów, krawędzi, trójkątów) zadawane są w trakcie wykonania programu
- Tablice alokowane są w trakcie wykonania
- Składowe struktur zawierają wskaźniki do odpowiednich elementów tablic

Przykład złożonej struktury danych

- Zastosowanie tablic struktur pozwala na posługiwanie się indeksami w tablicach zamiast wskaźnikami do struktur:

```
// definicje struktur w pakiecie mg – moja grafika – wersja 2.2
typedef struct mg_punkt_2D{ // punkt w przestrzeni 2D
  double   x;        // współrzędna x
  double   y;       // współrzędna y
  int *trojkaty;   // indeksy trójkątów zawierających punkt
} mg_punkt_2D;

typedef struct mg_krawedz_2D{ // krawędź w przestrzeni 2D
  int koniec[2]; // indeksy punktów będących wierzchołkami
  int trojkaty[2];   // indeksy trójkątów zawierających krawędź
} mg_krawedz_2D;

typedef struct mg_trojkat_2D{ // trójkąt w przestrzeni 2D
  int boki[3];    // indeksy krawędzi (boków) trójkąta
  int sasiedzi[3]; // indeksy sąsiednich trójkątów
} mg_trojkat_2D;
```

Przykład złożonej struktury danych

- Zdefiniowane wcześniej struktury danych pozwalają na szybkie wykonywanie dużej liczby operacji na całej siatce (dodawanie nowych punktów, krawędzi, trójkątów)
- Istnieją inne możliwe struktury danych wystarczające do realizacji operacji, np. minimalistyczna struktura danych:

```
// definicje struktur w pakiecie mg – moja grafika – wersja 3
typedef struct mg_punkt_2D{ // punkt w przestrzeni 2D
      double   x;  double   y;      // współrzędne x i y
   } mg_punkt_2D;
typedef struct mg_trojkat_2D{ // trójkąt w przestrzeni 2D
      int wierzcholki[3];      // indeksy wierzchołków trójkąta w tablicy punktów
   } mg_trojkat_2D;
typedef struct mg_siatka_2D{
      int liczba_punktow;        mg_punkt_2D* tablica_punktow;
      int liczba_trojkatow;      mg_trojkat_2D* tablica_trojkatow;
   } mg_siatka_2D;
```

Przykład złożonej struktury danych

- W definicji siatki zastosowane zostały tablice struktur (AOS – *array of structures*)
- W celu zwiększenia wydajności obliczeń, stosuje się często alternatywne warianty struktur danych zawierające pojedynczą strukturę z tablicami zmiennych elementarnych typów liczbowych (SOA – *structure of arrays*)
- W przypadku programu grafiki 2D struktura danych SOA mogłaby mieć postać:

```
typedef struct mg_siatka_2D{
  int liczba_punktow;        double* wspolrzedne_punktow;
  int liczba_trojkatow;       int* punkty_trojkatow;
} mg_siatka_2D;
```

- Tak zdefiniowana struktura eliminuje potrzebę definiowania jakichkolwiek innych struktur
  - ceną jest zmniejszenie czytelności kodu operacji na siatce trójkątów