Zmienne dynamiczne

- Przypomnijmy, że zmienne dynamiczne to zmienne, które są tworzone w trakcie wykonywania programu, ich ilość w pamięci nie musi być znana w momencie kompilacji programu.
- Zmienne te nie mają nazwy, i można się do nic odnosić tylko przy pomocy wskaźników.
- Najczęściej zmienne te są powiązane pomiędzy sobą w ten sposób, że poszczególne zmienne zawierają w sobie wskaźniki na inne. Logika tych powiązań powinna być dostosowana do sposobu pracy ze zmiennymi. Obecnie opiszemy dwa typowe sposoby „aranżacji" zmiennych dynamicznych - listę i drzewo binarne.

Zmienne dynamiczne

- Przypomnijmy jeszcze, że zmienne dynamiczne z reguły nie są prostymi zmiennymi, tylko obiektami lub strukturami. O obiektach jeszcze nie mówiliśmy, więc rozważmy dane, które są strukturami. Przypomnijmy, że strukturę (złożony typ zmiennej) deklarujemy następująco:

```
struct nazwa
{
 int jeden_element;
 double inny_element;
 char jakis_napis[200];
   ...
 nazwa *wsk;
};
```

- Zauważmy, że elementem struktury jest wskaźnik na taką samą strukturę. Obecność takiego wskaźnika umożliwia budowę powiązań pomiędzy zmiennymi. Taki wskaźnik może być jeden jeżeli, na przykład, nasze dane zorganizowane są w listę, i wskaźnik zawarty w jednej zmiennej pokazuje na następną zmienną na liście.

Zmienne dynamiczne

- Często wskaźniki są 2, jeden wskazuje na następny element na liście, a drugi na poprzedni element. Tak zorganizowane dane nazywają się listą dwukierunkową, i taka organizacja jest często stosowana. W standardowych bibliotekach C++ znajduje się szablon takiej listy, który pozwala posługiwać się nią bez wnikania w szczegóły techniczne. Jednak zanim przejdziemy do takich szablonów (na C++ 2), postaramy się opanować taka listę „ręcznie”.
- Przykład 1.
- Przykład powyższy jest szkieletem programu, który umożliwia wprowadzanie danych bibliograficznych książek. Umożliwia dodawanie książki do listy, usuwanie z listy, przeglądanie listy itp. jest to tylko szkielet, nie wszystkie opcje są zaimplementowane, i wiele rzeczy jest zrealizowane nieoptymalnie. Celem przykładu jest pierwsze praktyczne wykorzystanie wskaźników.

Zmienne dynamiczne

- W programie zadeklarowana jest struktura o nazwie `book`, która zawiera pewne dane dotyczące książki, a także dwa wskaźniki, `nast` oraz `poprz`, na takie same struktury.

```
struct book
{
    char tytul[1000];
    char autor_nazw[50];
    char autor_imie[20];
    int cena;
    int rok_wyd;
    book *nast;
    book *poprz;
};
```

- Wspominaliśmy wcześniej, że zmienne dynamiczne w praktyce realizowane są jako tak zwane obiekty, czyli coś bardziej ogólnego niż struktury. Obiektami zajmiemy się wkrótce.

Zmienne dynamiczne

- Lista, która powstanie na stercie wymaga różnych funkcji, które będą na niej działać. Te funkcje to tak zwany interfejs listy. Łatwo sobie wyobrazić, jakie funkcje będą potrzebne. Na przykład funkcja, która pobierze od użytkownika dane, i doda odpowiednią strukturę z tymi danymi do listy. W naszym przykładzie interfejs składa się z następujących funkcji:

```
book *dodaj_ksiazke(book *&s_ptr, book *l_ptr);
void wypisz_liste(book *s_ptr);
void wypisz_ksiazke(book *s_ptr);
void usun_ksiazke(book *&s_ptr);
void usun_liste(book *s_ptr);
```

- Wewnątrz funkcji `main()` są dwa dodatkowe elementy interfejsu, wskaźniki `start_ptr` i `end_ptr`. Pierwszy pokazuje na pierwszy element listy, drugi na ostatni element.

Zmienne dynamiczne

- Przyjrzyjmy się funkcjom. `dodaj_element()` jako argumenty przyjmuje dwa wskaźniki, będą to wskaźniki na początek listy i na koniec.
- Funkcja ta tworzy nową strukturę i dopisuje ją na końcu listy. Do tego jest jej potrzebny wskaźnik na koniec listy. Ponieważ funkcja powinna zmodyfikować wskaźnik na koniec listy (został dodany nowy element), to wskaźnik na koniec listy jest jej przekazywany przez referencję (operator &).
- Jeżeli funkcja wywoływana jest po raz pierwszy, to żadnych zmiennych na liście nie ma. W tej sytuacji wskaźnik `start_ptr` ma wartość `NULL`. Funkcja tworząc pierszy element listy musi więc zainicjalizować wskaźnik `start_ptr`. Dlatego argument ten też przekazywany jest do funkcji przez referencję.

Zmienne dynamiczne

- Funkcja najpierw sprawdza, czy lista jest pusta (`s_ptr == NULL`). Nastepnie tworzy nową zmienną dynamiczną typu `book`. Następnie „podłącza” ją do listy na końcu. Wskaźnik `nast` dotychczasowego ostatniego elementu zostaje zapisany adresem nowego elementu. Wskaźnik `poprz` nowego elementu zostaje zapisany adresem poprzednio ostatniego elementu. W końcu wskaźnik `nast` nowo utworzonego elementu zostaje zapisany wartością `NULL`.
- Jeżeli lista była pusta, to oba wskaźniki w nowej zmiennej są ustawiane na `NULL`, oraz oba wskaźniki, na początek i na koniec listy są ustawiane na adres nowej zmiennej.
- Następnie funkcja pobiera od użytkownika dane, i zapisuje je do zmiennej. Przypomnijmy, że wprowadzanych stringów nie możemy bezpośrednio wpisywać do zmiennej, jeżeli zawierają spacje.

Zmienne dynamiczne

- Proste podstawienie

```
cin >> (*temp_str).tytul;
```

  zapisze jedynie pierwszy wyraz tytułu. Pozostałe wyrazy będą w buforze wejściowym, i do ich odczytania byłyby potrzebne kolejne instrukcje `cin`.
- Zamiast tego korzystamy z funkcji `cin.getline()`, która odczytuje całą linijkę aż do znaku końca linii `'\n'` (klawisz `enter`), i zapisuje całość do podanego bufora. Dodatkowym argumentem funkcji `cin.getline()` jest maksymalna ilość znaków do zapisania do bufora.

Zmienne dynamiczne

- Zwróćmy uwagę na dodatkowe wywołanie funkcji `get()` przed samym odczytem tytułu książki. Kiedy użytkownik wybrał opcję 1., wcisnął też klawisz `enter`. Do bufora wejściowego wczytana została liczba 1, a także znak końca linii. Program pobrał z bufora samą liczbę 1, ale znak końca linii pozostał. Funkcja `cin.getline()` odczytuje wszystko do znaku końca linii (odczytuje też ten znak, ale zapisuje zamiast niego znak `'\0'`, kończący string). W sytuacji, gdy w buforze wejściowym `cin` czeka sam znak końca linii, funkcja odczyta pusty string. Żeby tego uniknąć iżywamy funkcji `cin.get()` która, w tej postaci, pobiera z bufora wejściowego jeden bajt i nic z nim nie robi. Bufor jest więc pusty i czeka na nasz string.
- Przy kolejnych wywołaniach funkcji `cin.getline()` nie ma tego problemu, bo ta funkcja pobiera znak końca linii z bufora wejściowego.

Zmienne dynamiczne

- Po wywołaniu funkcji `dodaj_ksiazke()` z parametrami `start_ptr` oraz `end_ptr` do listy dołączony zostaje na końcu nowy element, zapisany danymi podanym przez użytkownika. Wskaźniki `start_ptr` oraz `end_ptr` po wywołaniu funkcji zawierają adresy początku i końca listy.
- Kolejna funkcja operująca na liście to `wypisz_liste()`. Jako argument otrzymuje wskaźnik na początek listy. Wypisuje całość. Logika funkcji jest jasna. Wypisywanie pozycji następuje w pętli. Koniec pętli wykrywany jest kiedy wskaźnik `nast` danej zmiennej ma wartość `NULL` (nie ma następnego elementu).

Zmienne dynamiczne

- Kolejną funkcją jest `wypisz_ksiazke()`. Funkcja prosi użytkownika o podanie roku wydania książki, po czym przegląda listę. Jeżeli znajdzie zmienną o pasującym roku wydania, wypisuje ją, i kończy. Jeżeli nie znajdzie pasującej książki, informuje o tym. Zauważmy, że tak napisana funkcja wypisuje tylko pierwszą napotkaną książkę o podanym roku wydania. Zastanówmy się, jak zrobić, żeby funkcja po znalezieniu jednej pozycji kontynuowała szukanie następnej. Oczywiście możemy łatwo wybrać inny parametr wyszukiwania, na przykład nazwisko autora.
- Kolejną istotną funkcją jest `usun_ksiazke()`. Użytkownik identyfikuje książkę do usunięcia po roku wydania. Lista jest przeglądana, i kiedy odnaleziona zostaje książka o podanym roku wydania, zostaje usunięta.

Zmienne dynamiczne

- Zauważmy, że sytuacja jest różna w zależności od tego, czy znaleziona pozycja jest na początku listy, czy gdzieś dalej. Jeżeli książka do usunięcia jest na początku listy, to trzeba zaktualizować wskaźnik początku listy `start_ptr`. Z drugiej strony, jeżeli pozycja do usunięcia jest gdzieś dalej na liście, to listę trzeba „skleić" po usunięciu elementu. Zastanówmy się, jak zrobić, żeby po znalezieniu pozycji do usunięcia wypisać ją, i poprosić użytkownika o potwierdzenie usunięcia.
- Ostatnią zaimplementowaną funkcją jest `usun_liste()`, która usuwa wszystkie elementy listy z pamięci. Otrzymuje wskaźnik na koniec listy i rekurencyjnie, od końca, usuwa zmienne.
- Wszystkie zmienne utworzone przy użyciu instrukcji `new` powinniśmy usunąć, przy pomocy instrukcji `delete`.
- Zauważmy, że w programie wielokrotnie używamy konstrukcji `while(wskaznik)`. Wskaźnik może mieć wartość `NULL` lub jakąś konkretną. C++ traktuje ten pierwszy przypadek jako logiczne 0, a ten drugi jako logiczne 1.

Zmienne dynamiczne

- Interfejs naszej listy dwukierunkowej jest dosyć ubogi, zawiera tylko to, czego bezpośrednio potrzebowaliśmy. W wielu wypadkach taką listę obudowuje się dodatkowymi funkcjami. Typowym przykładem jest sortowanie.
- Zauważmy, że zupełnie bez znaczenia dla programu ma faktyczna, fizyczna, lokalizacja utworzonych zmiennych dynamicznych. Ważna jest tylko struktura logiczna, wyznaczona przez relację, która zmienna jest za/przed którą.
- Idąc dalej, zauważmy, że jedna lista może mieć wiele różnych struktur logicznych. Wymaga to tylko większej ilości wskaźników (na następny i poprzedni element) wbudowanych w listę. W szczególności lista może być posortowana jednocześnie według różnych parametrów. Tego typu zadanie było sporym problemem dla niektórych studentów w ubiegłym roku jako ostatni projekt.

Zmienne dynamiczne

- Kończąc omawianie listy dwukierunkowej zwróćmy uwagę na jeszcze jedną rzecz. W naszej liście nowe elementy zawsze były dopisywane na końcu. Równie dobrze moglibyśmy dopisywać nowe elementy na początku. Ale też moglibyśmy utrzymywać naszą listę cały czas posortowaną (względem jakiegoś parametru) i nowe elementy dopisywać zawsze we „właściwym miejscu".

Zadanie domowe 6 (do oddania 7.06.)

Uzupełnij w przykładzie 1 elementy o których była mowa w tekście. To znaczy dodaj potwierdzenie przed usunięciem pozycji (po jej znalezieniu), oraz wypisywanie wielokrotnych pozycji z tym samym rokiem wydania.