DYNAMICZNE PRZYDZIELANIE PAMIECI

Pamięć komputera, dostępna dla programu, dzieli się na cztery obszary:

- **kod programu**,
- dane **statyczne** (np. stałe i zmienne globalne programu),
- dane **automatyczne**
  → zmienne tworzone i usuwane automatycznie przez kompilator na tzw. **stosie** (*ang. stack*) np. zmienne lokalne wewnątrz funkcji

```
void przykladowa_funkcja(void)
{
  float zmienna_lokalna;
  zmienna_lokalna=10;
}
```

- dane **dynamiczne**
  → organizowane przez menadżera-zarządcę pamięci dynamicznej, można je tworzyć i usuwać w dowolnym momencie pracy programu, w pamięci wolnej komputera → na tzw. **stercie** (*ang. heap*)

Zmienne dynamiczne:

→ odpowiedzialnym za ich utworzenie (rezerwację pamięci) oraz za ich usunięcie (zwolnienie pamięci) **jest programista !!!**

→ dostęp do takiej zmiennej możliwy jest jedynie poprzez jej adres w pamięci (przechowywany w zmiennej wskaźnikowej)

→ korzystanie z nieprzydzielonego obszaru najprawdopodobniej spowoduje błąd!

→ próba zwolnienia już zwolnionego obszaru spowoduje błąd!

Przykład → ilustracja czasu „życia" zmiennych

zmienna statyczna	↔	komputer na własność (cały czas)
zmienna lokalna	↔	komputer w laboratorium (tylko na czas zajęć)
zmienna dynamiczna	↔	komputer z wypożyczalni (na dowolny czas)

Dostęp do **obiektu** za pomocą **wskaźnika-adresu-odsyłacza**

WSKAŹNIK	→	**OBIEKT**
numer telefonu	→	telefon
adres internetowy	→	strona HTML na serwerze
adres pocztowy	→	mieszkanie
numer PESEL	→	Kowalski Jan
numer pokoju	→	wynajęty apartament w hotelu
numerek z szatni	→	**dynamicznie przydzielone miejsce-wieszak w szatni**

W języku „**C**” do dynamicznego przydzielania pamięci (tworzenia zmiennych dynamicznych) służyły specjalne funkcje z bibliotek: **<alloc.h>** lub **<stdlib.h>**

```
void *malloc( size_t  rozmiar );                  // przydział bloku o zadanej wielkosci
void *calloc( size_t  il_elementow,  size_t rozmiar);   // przydział tablicy
void *realloc( void* stary_wskaznik,  size_t nowy_rozmiar);  // zmiana wielkości
void  free( void* wskaznik);                      // zwolnienie wskazywanego obszaru
```

np.
```
int main( void )
{
   int *wsk = NULL;  // zmienna wskaźnikowa do zapamiętania adresu liczby int
   • • •
   wsk = (int*) malloc( sizeof(int) );        // przydzielenie pamięci na liczbę int
   if( wsk == NULL )
      {  printf( "Błąd przydziału pamięci" );  return;
   • • •
   *wsk = 10;                    // przykładowe operacje na dynamicznej liczbie int
   *wsk *= 2;
   printf( "%d", *wsk );
   scanf( "%d", wsk );
   • • •
   free( wsk );                  // zwolnienie pamięci przed zakończeniem programu
   return 0;
}
```

Przykład operacji na dynamicznej tablicy o dowolnej ilości elementów:

np.
```
void main( void )
{
   int rozmiar_tablicy;
   double *tablica_liczb;
   printf( "Ile liczb chcesz wprowadzić: " );
   scanf( "%d", &rozmiar_tablicy );

   if( tablica_liczb = (double*) calloc( rozmiar_tablicy, sizeof(double) ) )
      {
         for( int i = 0; i < rozmiar_tablicy, i++ );
            *( tablica_liczb+i ) = 100;          // tablica_liczb[ i ] = 100;
         • • •
         free( tablica_liczb );
      }
   return 0;
}
```

W języku „**C++**” do dynamicznego przydzielania pamięci
można nadal wykorzystywać funkcje z biblioteki <alloc.h>
ale dużo lepiej jest korzystać z nowych operatorów: **new** oraz **delete**

<wskaźnik_na_obiekt> = **new** <typ_obiektu> [*parametry_inicjalizacyjne*] ;
delete <wskaźnik_na_obiekt> ;

np.

```
int*  wsk ;                    // wskaźnik na zmienną typu całkowitego
wsk = new(nothrow) int ;       // utworzenie nowego obiektu (nowej zmiennej int)
if( wsk != NULL )
   {
      *wsk = 10 ;              // przypisanie wartości  (poprzez wskaźnik)
      printf( "%d" , *wsk );   // wydrukowanie zawartości zmiennej dynam.
      • • •
      delete wsk ;             // usunięcie zmiennej dynam. (zwolnienie pamięci)
   }
```

Porównanie utworzenia zwykłej tablicy i tablicy dynamicznej:

```
// operacja utworzenia zwykłej tablicy
const  int ROZMIAR_TABLICY = 100;
double zwykła_tablica[ ROZMIAR_TABLICY ];
```

```
// operacja utworzenia i zwolnienia tablicy dynamicznej
int rozmiar_tablicy;
cout << "Ile liczb chcesz wprowadzić: " ;
cin >> rozmiar_tablicy ;

double *tablica_dynamiczna = NULL;
tablica_dynamiczna = new(nothrow) double[ rozmiar_tablicy ];

• • •

for(int i=0; i<rozmiar_tablicy; i++)
   tablica_dynamiczna[ i ] = 10.5;

• • •

for(int i=0; i<rozmiar_tablicy; i++)
   cout<<endl<<"tablica[" << i+1 << "] = " << tablica_dynamiczna[ i ];

• • •

delete [ ] tablica_dynamiczna;
```

```
double* wsk;
try {
   wsk = new double [100 ];
} catch( bad_alloc& err ) {
   cout << "Blad: "<< err.what() << endl;
}
```

Przykład 1 - pojedyncza realokacja (zmiana rozmiaru) tablicy jednowymiarowej

```
int main( )
{
   // utworzenie 10-cio elementowej tablicy zawierającej liczby z przedziału -50÷50
   int rozmiar=10;
   long* tablica = new(nothrow) long[ rozmiar ];
   for(int i=0; i< rozmiar; i++)
      tablica[ i ] = random(101)-50;

   cout<<endl<<"Zawartosc tablicy po wylosowaniu elementów: "<<endl;
   for (int i=0; i<rozmiar ; i++)
      cout << endl <<"tab[" << i << "] = " << tablica[ i ];
   cout<<endl<<"Rozmiar tablicy: "<<rozmiar<<endl;

   // policzenie ile z wylosowanych liczb ma dodatnią wartość
   int ilosc_dodatnich=0;
   for(int i=0; i<rozmiar; i++)
      if( tablica[i]>0 )
         ilosc_dodatnich++;

   // usunięcie wszystkich liczb ujemnych →z jednoczesnym zmniejszeniem tablicy
   long* nowa_tablica = new(nothrow) long [ilosc_dodatnich];
   if( nowa_tablica==NULL )
      cout<<"UWAGA - blad tworzenia nowej tablicy";
   else
      {
         int j=0;
         for(int i=0;i<rozmiar;i++)
            if( tablica[i]>0 )
               {
                  nowa_tablica[ j ]=tablica[ i ];
                  j++;
               }

         delete [ ] tablica;
         tablica=nowa_tablica;
         rozmiar=ilosc_dodatnich;
      }

   cout<<endl<<"Zawartosc tablicy po usunieciu liczb ujemnych:"<<endl;
   for (int i=0; i<rozmiar ; i++)
      cout << endl <<"tab[" << i << "] = " << tablica[ i ];
   cout<<endl<<"Rozmiar tablicy: "<<rozmiar<<endl;

   cin.get();
   delete [ ] tablica;
   return 0;
}
```

Przykład (2) – inna wersja programu przykład (1) → z wykorzystaniem funkcji

```
bool  USUN_UJEMNE(long* &wsk_tablicy, int &rozmiar_tablicy)
{
   int ilosc_dodatnich=0;
   for(int i=0; i<rozmiar_tablicy; i++)
      if( wsk_tablicy[i]>0 )
         ilosc_dodatnich++;

   long* nowa_tablica = new(nothrow) long [ilosc_dodatnich];
   if( nowa_tablica==NULL )
      return false;

   int j=0;
   for(int i=0; i<rozmiar_tablicy; i++)
      if( wsk_tablicy[i]>0 )
         {
            nowa_tablica[ j ]=wsk_tablicy[ i ];
            j++;
         }

   delete [ ] wsk_tablicy;
   wsk_tablicy=nowa_tablica;
   rozmiar_tablicy=ilosc_dodatnich;
   return true;
}


long* LOSUJ_UJEMNE_i_DODATNIE(int ilosc_liczb);
void   WYSWIETL(long* tablica, int rozmiar_tablicy);
bool   USUN_UJEMNE(long* &wsk_tablicy, int &rozmiar_tablicy);



int main( )
   {
      int n=10;
      long *tablica = LOSUJ_UJEMNE_i_DODATNIE (n);
      WYSWIETL( tablica,n);
      cin.get();
      if( USUN_UJEMNE(tablica,n)==false )
         cout<<"UWAGA - blad operacji usuwania ujemnych";
      cout<<endl<<endl<<"Po wywolaniu funkcji USUN_UJEMNE:"<<endl;
      WYSWIETL( tablica,n);
      cin.get();
      delete [ ] tablica;
      return 0;
   }
```

dalszy ciąg Przykładu (2)

```
void WYSWIETL(long* tablica, int rozmiar_tablicy)
{
   for (int i=0; i<rozmiar_tablicy ; i++)
      cout << endl <<"tab[" << i << "] = " << tablica[ i ];
   cout<<endl<<"Rozmiar tablicy: "<<rozmiar_tablicy<<endl;
}


long* LOSUJ_UJEMNE_i_DODATNIE(int ilosc_liczb)
{
   long* nowa_tablica = new(nothrow) long[ ilosc_liczb ];
   for(int i=0; i<ilosc_liczb ; i++)
      nowa_tablica[ i ] = random(100)-50;
   return nowa_tablica;
}
```