Język ANSI C

część 12

tablice 2D, alokacja pamięci

Jarosław Gramacki
Instytut Informatyki i Elektroniki

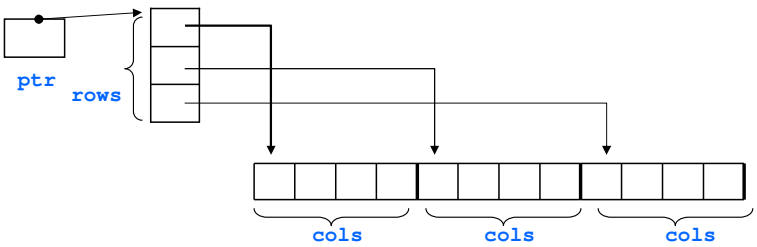
Funkcja malloc()

– gdy każdy wiersz alokowanej tablicy 2D ma taką samą długość, to można uprościć procedurę alokacji pamięci

```
int main (void) {
int i, rows=3, cols=4;    // lub podawane "dynamicznie"
int **ptr = (int**) malloc (rows * sizeof(int*));
ptr[0] = (int*) malloc (rows * cols * sizeof(int));

// "ustaw" wskaźniki
for (i=1; i<rows; i++)
  ptr[i] = ptr[0] + i * cols;

// zwolnienie pamięci (prostsze niż poprzednio)
free( ptr[0] );           // zwolnienie całej pamięci "roboczej"
free ( ptr );             // zwolnienie wskaźników do pamięci "roboczej"
return 1 }
```

jeden spójny obszar pamięci

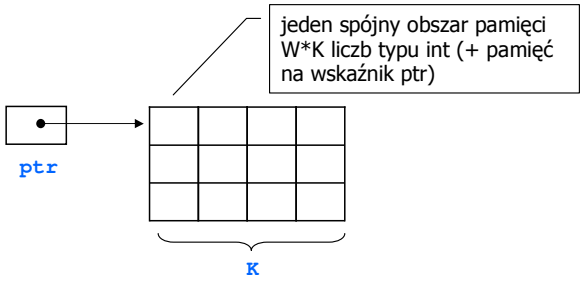
dostęp do tablicy dalej możliwy w postaci `ptr[i][j]`



Funkcja calloc()

– tabica 2D W x K może być traktowana jako
1-no wymiarowa tablica K-elementowych tablic

```
#include <assert.h>
#define K 4

int main (void) {
int i, j;
int (*ptr)[K]; // wskaźnik tablicy K zmiennych typu int

// tu wczytanie liczby wierszy W
ptr = (int(*)[K]) calloc (W, sizeof(ptr[0]));       // bardziej uniwersalne
// lub:
// ptr = (int(*)[K]) calloc (W, K * sizeof(int));   // mniej uniwersalne
assert(ptr != NULL);                                // akcja gdy false

for (i=1; i<W; i++)
 for (j=1; j<K; j++)
  ptr[i][j] = i * j;

for (i=1; i<W; i++)
 for (j=1; j<K; j++)
  printf("%d ", ptr[i][j]);

free(ptr);
getchar();
}
```



Alokacja pamięci, kilka uwag

– jaka jest różnica pomiędzy:

```
int (*ptr)[80]
int A[10][80]
```

– `ptr` to zmienna, `A` to stała

– `ptr + 1` wskazuje na kolejną tablicę, czyli o 80 obiektów typu `int` dalej niż `ptr`

– `A + 1` wylicza wskazanie przesunięte o 80 obiektów typu `int` dalej niż `A`

– Wniosek:
zmienna `ptr` i stała `A` są tego samego typu

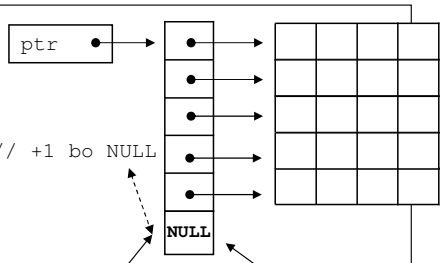
Uniwersalny "alokator" pamięci

```
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>

void* AlokujTablice (int wierszy, int kolumn, int RozElem);
void  ZwolnijTablice (void *ptr);

int main (void)
{
  int **A;                                  // testujemy dla int-ów
  int i, j, w=3, k=4;

  A = (int**) AlokujTablice(w, k, sizeof(**A));
  // sizeof(**A) - rozmiar 1 elem. tablicy
  // rzutowanie konieczne
  assert( A!= NULL);

  A[1][2] = 3;                              // tu można już używać [][]
  A[2][2] = 5;

  for (i=1; i<w; i++)                       // test
    for (j=1; j<k; j++)
    printf("%d ", A[i][j]);

  ZwolnijTablice(A);
}
```

porównaj prototypy standardowych funkcji malloc() i free()

Uniwersalny "alokator" pamięci

```
void* AlokujTablice(int w, int k, int size) {
  int i;
  // to nasz "klucz" do pamięci
  void **ptr;

  ptr = (void**) calloc(w + 1, sizeof(*ptr)); // +1 bo NULL
  if ( !ptr ) return NULL;
  for (i=0; i<w; i++)
    if ( !(ptr[i] = calloc(k, size)))
      {
      // gdy błąd, zwolnij co już zdążono przydzielić
      ZwolnijTablice( ptr );
      return NULL;
      }
  ptr[w] = NULL;
  return ptr;             // zwróć zaalokowaną pamięć (wskaźnik) na zewnątrz
}

void ZwolnijTablice(void *ptr)     // porównaj: void free(void *ptr);
{
  int i;
  void **localptr = (void**) ptr;  // rzutowanie konieczne
  if (!ptr) return;                // gdy przekazano NULL nie rób nic
  for (i=0; localptr[i]; i++)      // korzystamy z tego NULL-a
    free(localptr[i]);
  free(ptr); }
```



Alokacja pamięci, wczytaj plik z dysku do pamięci

```
#include <stdio.h> #include ...
char* CzytajWiersz(FILE *fp);  int IleLinii(FILE *fp);

int main (void) {  int n, i;
  char **ptr;
  FILE *fp = fopen("plik.txt", "r"); assert (fp != NULL);

  n = IleLinii(fp);
  ptr = (char**) malloc ( n * sizeof(*ptr) ); assert (ptr != NULL);
  rewind(fp);                       // bo używamy fp w IleLinii()

  for (i=0; i < n; i++)
  {
   ptr[i] = CzytajWiersz(fp);       // alokacja pamięci na i-ty wiersz w
                                    // w funkcji CzytajWiersz()
   if ( ! ptr[i] )// to co wczytano do czasu ew. błędu też jest użyteczne
     {n = i;  break;}
  }

   for (i=0; i < n; i++)
    printf("--> %s", ptr[i]);      // plik już w pamięci

  fclose(fp);
  for(i=n-1; i>=0; i--)
    free( ptr[i] );
  free( ptr );
  return 1; }                       c.d.n.
```

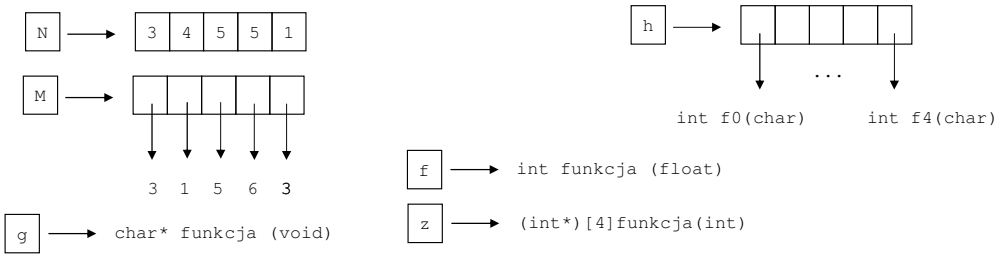

Alokacja pamięci, wczytaj plik z dysku do pamięci

```
                                  c.d.

char* CzytajWiersz(FILE *fp)
{
  char buf[256], *ptr;
  if ( ! fgets(buf, 255, fp) )
    return NULL;

  ptr = (char*)malloc(strlen(buf) + 1);

  if (ptr)
    strcpy(ptr, buf);
  return (ptr);
}

int IleLinii(FILE *fp)
{
  int c, ile = 0;
  while ((c = fgetc ( fp )) != EOF )
    if (c == '\n') ile++;
  return (ile);
}
```

Kilka bardziej złożonych deklaracji

– raczej nie stosować tak skomplikowanych (deklaracje 5 i 6) deklaracji !!!

```
char *L[5];                 // tablica L 5-u wskaźników do char
int (*N)[5];                // wskaźnik N do tablicy 5-u zmiennych typu int
int *(*M)[5];               // wskaźnik M do tablicy 5-u wskaźników do int

int (*f)(float);            // wskaźnik do funkcji pobierającej float a
                            // zwracającej int
char *(*g)(void);           // wskaźnik do funkcji bezparametrowej
                            // zwracającej char*

int (*(*h)[5])(char);       // wskaźnik tablicy 5-u wskaźników do funkcji o
                            // argumencie char zwracających int

int (*(*z)(int))[4];        // wskaźnik do funkcji o argumencie int zwracających
                            // wskaźnik do tablicy 4-ch elementów typu int
```



Kilka bardziej złożonych deklaracji

– następny wykład: wskaźniki do funkcji