Zajęcia VI

Cele:

- omówić alokację pamięci

Problem

Załóżmy, że nie mamy `std::string`. Jak wczytać string od użytkownika?

Można tak:

```
#include <iostream>

int main() {
    char array[10];

    std::cin >> array;
    std::cout << array << std::endl;

    return 0;
}
```

Ale co jeśli użytkownik wpisze `11` znaków?

To zróbmy: `char array[20];`

Ale co jeśli użytkownik wpisze `25` znaków?

Rozwiązanie 💡🎯✅🧩

Dynamiczna alokacja pamięci

```
char * array = new char[10];
int * number_array = new int[10];

struct dna {
    int a,t,c,g;
};

int count = 10;
dna * dnas = new dna[count];
```

Katastrofa ❌

```
#include <iostream>

int main() {

    while(1) {
        char * memory = new char[1024];
    }

    return 0;
}
```

Katastrofa zażegnana ✅

```
#include <iostream>

int main() {

    while(1) {
        char * memory = new char[1024];

        delete[] memory;
    }

    return 0;
}
```

Zadanie Lab I

Napisz własną implementację `std::string`

```cpp
int main()
{
  int size = 10;
  char *array = new char[size];
  int length = 0;

  char ch;
  while (std::cin.get(ch) && ch != '\n')
  {
    if (length >= size - 1)
    {
      size = size * 2;

      char *new_array = new char[size];
      for (int i = 0; i < length; ++i)
        new_array[i] = array[i];

      delete[] array;

      array = new_array;
    }
    array[length++] = static_cast<char>(ch);
  }

  array[length] = '\0';

  std::cout << "Input: " << array << std::endl;
  std::cout << "Length: " << length << std::endl;
  std::cout << "Size: " << size << std::endl;

  delete[] array;
  return 0;
}
```

Referencja - rozszerzenie wskaźników

```
void process_string( std::string dna ) {
  char third_element = dna[2];
}
```

```
void process_string( std::string * dna ) {
  char third_element = (*dna)[2];
}
```

```
void process_string( std::string & dna ) {
  char third_element = dna[2];
}
```

Zadanie Lab II

Napisać program, który sprawdzi działanie referencji vs kopiowania vs wskaźnika

```cpp
#include <iostream>

// przekazać argument przez wartość
// potem przez wskaźnik
// potem przez referencję
void process(std::string dna)
{
  std::cout << dna.length() << std::endl;

  while (1)
  {
    // simulate processing
  }
}

int main()
{

  std::string dna;

  for (int i = 0; i < 100000000; i++)
    dna += "A";

  process(dna);

  return 0;
}
```

Zadanie I

Napisz program, który:

- wczyta z pliku ciągi dna, gdzie pierwszy wiersz będzie liczbą oznaczającą ilość ciągów w pliku, a przed każdym ciągiem będzie również linia z jego długością.
- wypisze na wyjście informację dla każdego ciągu:
  - NO - jeśli ciąg nie jest palindromem
  - YES - jeśli ciag jest palindromem
  - INVALID - jeśli ciąg zawiera niepoprawne znaki

- do badania 'palindomowości' ciągu stwórz osobną funkcję, która skorzysta ze wskaźników aby określić, czy string wejściowy jest palindomem. Skorzystaj z `std::string` i referencji.
- musi zaalokować dynamicznie dokładnie tyle miejsca, ile wskazują na to wartości w pliku. Może się zdarzyć, że ciąg testowy będzie miał długość 10, a może 10 tysięcy, a może `n` milionów.
- pamiętaj obsłużyć duże i małe literki
- pamiętaj 'posprzątać' po sobie zaalokowaną pamięć

Przykład:

Wejście w pliku:

```
6
5
ATTCG
4
ttCt
4
TTAT
9
ilovecpp!
7
ttCGACT
5
ATTTA
```

Wyjście:

```
NO
NO
NO
INVALID
NO
YES
```