

**Zawartość:**

- teoria stringów odziedziczonych przez C++ z języka C

## C-Style string

String to jeden z najpopularniejszych typów danych. Każdy go używa. Nie inaczej jest w C++.

C++ oferuje stringi:

- `std::string`
- odziedziczone po C stringi C-style

Osoby mniej programujące w C/C++ pewnie uważają, że można żyć bez starych łańcuchów tekstowych. No właśnie niezbyt. Po pierwsze **std::string** to tylko nakładka abstrakcji na stare stringi. Po drugie nawet tworząc zmienną typu **std::string** używamy c-style stringa.

```
int main()
{
    std::string variable = "ASD";
}
```

☐ (const char [4])"ASD"  
[Search Online](#)

Visual Studio ładnie tutaj podpowiada, że tak naprawdę do zmiennej **variable** przypisujemy stary string c-style w postaci tablicy czterech charów/czterech bajtów (tak czterech).

## Co to jest c-style string?

Jest to tablica charów (char na każdy znak plus terminator w postaci bajtu/chara 0).

```
char variable[] = "ASD";

//line for break point
int a = 10;
```

Prosty kodzik. Poniżej podgląd na miejsce w pamięci gdzie tablica została umieszczona (adres można dostać wskaźnikiem/visual studio w trybie debugu wyświetla adresy zmiennych). Widać terminator w postaci, zera. Prosta sprawa. Po prawej interpretacja bajtów w postaci tekstu.

[illegible]

Meme apropos screena z kodem



Tablice C-style można zapisać na kilka sposobów. Nie każdy gwarantuje, że 0 na końcu się znajdzie.

```
//tu zero doda się automatycznie
//rozmiar tablicy też będzie auto 4
char variable[] = "ASD";

//tutaj zero też będzie
//wynika to z tego, że reszta buffora
//będzie dopełniona domyślną wartością dla char
//czyli 0
char variable2[10] = "ASD";

//tutaj zero nie będzie
//visual krzyczy bo jest inteligentny
//nie jest w stanie upchać czterech charów do trzech
char variable3[3] = "ASD";

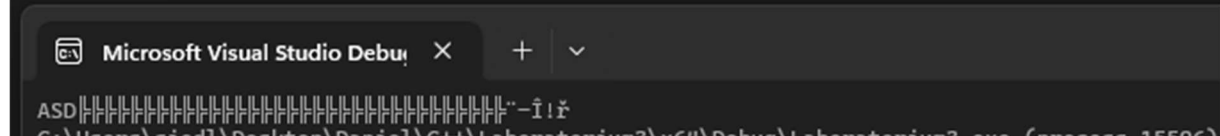
//tutaj zera nie będzie
//visual nie krzyczy
//ciekawo będzie print tej zmiennej
char variable4[3] = { 'A', 'S', 'D' };
```

Jak widać nie zawsze 0 dodaje się automatycznie.

Tak się kończy wypisanie stringa bez terminatora:

```
//tutaj zera nie będzie
//visual nie krzyczy
//ciekawy będzie print tej zmiennej
char variable4[3] = { 'A', 'S', 'D' };

std::cout << variable4;
```



Program leci po pamięci tak długo, aż trafi na zero. Nie każdy bajt da się przerobić na znak.

Zależnie gdzie tworzymy zmienną może ona być mniej lub bardziej losowo rzucona w pamięci. Nie zawsze wiemy co jest za nią, dlatego też taki print może mieć różną długość i wygląd przy każdym uruchomieniu.

**Chciałbym tutaj coś wyjaśnić, ale nie za bardzo wiem jak to nazwać.**

```
int main()
{
    int a = 10;
}
```

Każdy wie co to jest. Jest to zmienna. Zmienna to komórka pamięci. Powyższy kod alokuje gdzieś komórkę pamięci typu int i wstawia tam liczbę 10. Proste.

A to:

```
int main()
{
    std::cout << 10;
}
```

Nie ma tutaj zmiennej, ale jest liczba... No właśnie to jest stała 10. Prawda jest taka, że program niejawnie zrobił sobie gdzieś w pamięci jakąś stałą. I ją tutaj podstawiał.

```
const int var1 = 10;

int main()
{
    std::cout << var1;
}
```

Tak to wygląda w rzeczywistości z dużym prawdopodobieństwem.

Ze stringami jest podobnie:

```
int main()
{
    std::cout << "ASD";
}
```

>>>

```
const char* var1 = "ASD";

int main()
{
    std::cout << var1;
}
```

Z powyższego wynika, że:

```
int main()
{
    char napis[] = "ASD";
}
```

tutaj mamy dwa stringi „ASD”

Program w momencie kompilacji dostaje globalną stałą „ASD”. Następnie ta stała jest kopiowana do bufora czyli tablicy napis.

Z powyższych wniosków przejść można do kolejnych. Jak to jest z tą możliwością edycji tych stringów:

```
char napis[] = "ASD";
napis[0] = 'Q';
std::cout << napis;
```

Wyświetla „QSD”. Czyli można taki string edytować.

**Można edytować tę kopię wrzuconą do bufora napis.** Stałej (oryginalnego „ASD”) po prawej stronie operatora równa się nie można edytować.

Dlaczego stałe na poniższych obrazkach to stałe, a nie zmienne?

```
int main()
{
    std::cout << 10;
}
```

```
int main()
{
    std::cout << "ASD";
}
```

Kwestia optymalizacji. Jeżeli w naszym programie 10 razy wyświetlany jest łańcuch „Wynik działania algorytmu to:” to w pamięci występuje on tylko raz, bo po co 10 razy. Ten **const** ma za zadanie blokować edycję, ponieważ zmiana w jednym miejscu powodowałaby zmianę w każdym innym gdzie ten string jest używany.

Aby edytować stringi, należy stworzyć ich kopię do zmiennej tablicowej tak jak wyżej lub za pomocą funkcji `memcpy()`.

```
int main()
{
    std::cout << "ASD";
}
```

☐ `(const char [4])"ASD"`  
[Search Online](#)

Warto przeanalizować sobie w głowie ten obrazek.