Wstęp do programowania w języku C

Grupa TDr

Lista 6.

1. *(10 punktów w trakcie pracowni, później 5 punktów)*

Napisz funkcje o podanych deklaracjach i wymaganiach (można korzystać z `string.h`):

1. Funkcja `const char* print_line(const char* str);` ma wypisać napis `str` oraz znak końca linii i zwrócić napis `str`.

2. Funkcja `char* end_of_strcpy(char* dst, const char* src);` ma skopiować napis `src` do pamięci `dst` i zwrócić wskaźnik na bajt, w który został wpisany znak `'\0'`.

3. Funkcja `void print_lines(int argc, const char* argv[]);` ma wypisać w osobnych liniach napisy z tablicy `argv`, której rozmiar to `argc`.

W funkcji `int main(int argc, const char* argv[]);` umieść wywołanie `print_lines` z argumentami funkcji `main`. Zakładając, że Twój skompilowany program nazywa się `a.out`, wypróbuj wywołanie:

```
./a.out Hello world "Hello world" </dev/null
```

4. Funkcja `int sum_strlen(int argc, const char* argv[]);` ma zwrócić sumę długości napisów z tablicy `argv`, której rozmiar to `argc`.

5. Funkcja `char* copy_join(char* dst, const char* sep, int argc, const char* argv[]);` ma wpisać do pamięci `dst` sklejenie napisów z tablicy `argv` (której rozmiar to `argc`) poprzedzielanych napisem `sep` i zwrócić wskaźnik na bajt, w który został wpisany znak `'\0'`.

Przykładowo *efektem* poniższych instrukcji powinno być wypisanie linijki `"1 + 1 + 2"`.

```
const char memory[10];
const char* str_numbers[] = { "1", "1", "2", "5" };
copy_join(memory, " + ", 3, str_numbers);
print_line(memory);
```

2. *(10 punktów, zadanie sprawdza zdalnie Mateusz Wasylkiewicz)*

Napisz funkcję o deklaracji (specyfikacja na następnej stronie):

```
int realloc_strings(char dst[], const char* strings[], int count);
```

Zakładamy, że w tablicy `dst` jest wystarczająco wiele miejsca. Funkcja ma wypełnić minimalnie wiele bajtów w tablicy `dst` i nadpisać tablicę `strings` (wielkości `count`) tak, aby wskaźniki tablicy `strings` wskazywały na pamięć tablicy `dst`, ale reprezentowały nadal te same napisy.

Optymalizacja pamięciowa ma polegać więc na tym, że sufiksy i powtórki słów nie są pamiętane osobno.

Wskazówka. Przed wypełnianiem tablicy `dst`, można ustalić wygodną kolejność jej wypełniania.

Kolejność słów w tablicy `strings` po wywołaniu `realloc_strings` powinna być taka sama, jak była oryginalnie (2 punkty).

Pod względem wydajnościowym wystarczy, żeby funkcja działała sprawnie dla co najwyżej kilkunastu łańcuchów po co najwyżej kilkanaście znaków.

Przykład. Wywołanie poniższej funkcji `main`

```
int main() {
    const char* strings[] = {"owa", "krowa", "a", "banan", "ban", "ban"};
    const int count = 6;

    char buffer[200];
    const int bytes_reserved = realloc_strings(buffer, strings, count);

    for(int i = 0; i < bytes_reserved; i += 1)
        printf("%c", buffer[i] ? buffer[i] : '_');
    for(int i = 0; i < count; i += 1)
        printf(" %ld", strings[i] - buffer);
    printf("\n");
    print_lines(count, strings);
}
```

gdzie `print_lines` pochodzi z poprzedniego zadania, może np. wypisać:

```
krowa_banan_ban_ 2 0 4 6 12 12
owa
krowa
a
banan
ban
ban
```

3. *(10 punktów)*

w serwisie SKOS