



Funkcje

(podprogramy)

39

ISO/ANSI C - funkcje

Funkcja – fragment kodu, który może być wykonywany wielokrotnie z różnych miejsc programu.

Ogólny zapis:

```

typ nazwa (argumenty)
{
    ciało funkcji
}

```

typ – określa typ danych zwracanych po wykonaniu funkcji
nazwa – identyfikator funkcji w kodzie programu
argumenty – jedna lub więcej wartości, które zostaną przekazane do użycia przez kod funkcji w postaci wewnętrznych zmiennych lokalnych
ciało funkcji – instrukcje, które zostaną wykonane po wywołaniu funkcji

© UKSW, WMP, SNS, Warszawa 40

40

ISO/ANSI C - funkcje

Argumenty wywołania i zmienne lokalne:

kiedy stosujemy funkcje, zmienne zadeklarowane w tych funkcjach mają ściśle określony obszar, w którym są dostępne, oraz czas, kiedy istnieją.

Przykład:

```


int silnia(int x)
{
    int i, wynik = 1;
}

```

- Argumenty wywołania: 'x'
- Zmienne lokalne: 'i' oraz 'wynik'

Zmienne lokalne – tworzone wtedy, kiedy wywoływana jest funkcja, istnieją tak długo, póki sterowanie znajduje się w obrębie kodu funkcji

Ileokroć wywoływana jest ta sama funkcja, jej zmienne są tworzone na nowo. Po wykonaniu ostatniej instrukcji w funkcji jej zmienne lokalne są usuwane.



© UKSW, WMP, SNS, Warszawa 41

41


ISO/ANSI C - funkcje

Dostęp do zmiennych lokalnych:

Zmienne lokalne – dostępne wyłącznie w kodzie funkcji, w której zostały zadeklarowane

Mamy dane dwie funkcje: 'main' i 'silnia'. Jeżeli 'main' wywołuje 'silnia', to:

- zmienne lokalne 'main' nie są dostępne w kodzie 'silnia', choć w momencie wywołania 'silnia' zmienne te nie są usuwane
- po zakończeniu działania 'silnia' nadal można pracować na zmiennych lokalnych 'main' tak długo, póki 'main' nie zakończy działania

 *Jak w kodzie 'silnia' odwołać się do wartości przechowywanych w zmiennych lokalnych 'main'?*

Przekazując ich wartości do wnętrza 'silnia' korzystając z argumentów wywołania 'silnia'.

© UKSW, WMP, SNS, Warszawa 42

42

ISO/ANSI C - funkcje

Przesłanie zmiennych:

Zmienne zdefiniowane w pliku w obszarze kodu pomiędzy funkcjami są nazywane **zmiennymi globalnymi**. Są one widziane wewnątrz wszystkich funkcji

Zmienne lokalne funkcji mogą mieć taką samą nazwę, jak zmienne globalne: następuje wtedy przesłanie zmiennych. Korzystając z takiej nazwy zawsze odwołujemy się do zmiennej zdefiniowanej „bliżej”, tj. zmiennej lokalnej.

Przesłanie dotyczy również argumentów wywołania funkcji.

© UKSW, WMP, SNS, Warszawa 43

43

ISO/ANSI C - funkcje

Argumenty wywołania:

```

int silnia(int x)
{
    int i, wynik = 1;
    ...
    return wynik;
}
int a,b,c,d,x;
...
a = silnia(b);
...
a = silnia(d);
...
c = silnia(x);

```

W momencie wywołania funkcji 'silnia' utworzona jest wewnątrz funkcji zmienna 'x', do której kopiowana jest zawartość zmiennej 'b'.

Zmiany wartości zmiennej 'x' zadeklarowanej wewnątrz funkcji, które byliby tam wykonane, nie powodują, że zmieniana jest również wartość zmiennej 'b' – 'x' jest tylko kopią ('b' i 'x' to dwie różne zmienne).

Wartość wyrażenia umieszczonego po 'return' jest wartością wywołania funkcji 'silnia' zwracaną po jej zakończeniu. Wartość ta jest przepisywana do lewej strony równania, w którym nastąpiło wywołanie 'silnia'.

© UKSW, WMP, SNS, Warszawa 44

44

ISO/ANSI C - funkcje

Użycie funkcji w tekście musi być (powinno być) leksykalnie poprzedzone jej definicją.

Dlatego funkcja 'silnia' została w pliku umieszczona przed 'main'

© UKSW, WMP, SNS, Warszawa

45

45

ISO/ANSI C - funkcje

Użycie funkcji w tekście musi być leksykalnie poprzedzone jej definicją.

Problem:



```
int fun1(int k) {
    ..
    x = fun2(k);
    ..
}

int fun2(int m) {
    ..
    y = fun1(m);
    ..
}
```

© UKSW, WMP, SNS, Warszawa

46

46

ISO/ANSI C - funkcje

Definicje i deklaracje funkcji

Jeżeli w kodzie pojawi się odwołanie do funkcji, której wcześniej nie zdefiniowano, kompilator nie będzie potrafił określić, czy:

1. ta funkcja istnieje gdzieś w kodzie, czy też
2. jest to pomyłka programisty.

Kompilator w takim momencie nie potrzebuje pełnej definicji. Wystarczy mu zapewnienie, że taka funkcja istnieje. Tzn. wystarczy, że zna jej *prototyp*.

© UKSW, WMP, SNS, Warszawa

47

47

ISO/ANSI C - funkcje

Definicje i deklaracje funkcji

Deklaracja – podanie prototypu funkcji.

Deklaracja ma formę nagłówka funkcji, zawiera nazwę funkcji, informacje o tym, jakiego typu ma parametry oraz jakiego typu wartość zwraca. Po tym nagłówku następuje średnik.

Przykład:

```
int silnia(int);
```

Ta sama funkcja może być deklarowana w kodzie wielokrotnie.

Definicja – podanie pełnego kodu funkcji (nagłówek i ciało funkcji).

Definicja funkcji może wystąpić w kodzie programu tylko raz.

Deklaracje i definicje funkcji muszą być ze sobą zgodne.

© UKSW, WMP, SNS, Warszawa

48

48

ISO/ANSI C - funkcje

Rozwiązanie problemu:

W deklaracji nie musimy podawać nazw dla argumentów wywołania. Wystarczy, że określimy prawidłowo typ dla każdego z nich.

```
int fun2(int);
/* deklaracja 'fun2' */

int fun1(int k) {
    ..
    x = fun2(k);
    ..
}

int fun2(int m) {
    ..
    y = fun1(m);
    ..
}
```

© UKSW, WMP, SNS, Warszawa

49

49

ISO/ANSI C - funkcje

Deklaracje funkcji:

```
int silnia(int x);
```

```
void fun1(int a, int b);
```

```
void fun2(int a, int b=0,
          int c=3.14);
```

```
int fun3(void);
```

Wywołanie:

```
a = silnia(b);
silnia(b);
```

```
fun1(x,y);
```

```
fun2(x,y,z);
fun2(x,y);
fun2(x);
```

```
a = fun3();
fun3();
```

© UKSW, WMP, SNS, Warszawa

50

50

ISO/ANSI C - funkcje

Przekazywanie argumentu przez wartość i wskaźnik

```
void fun(int x, int *y)
{
    x = 2*x;
    *y = 4* (*y);
}
```

x – przekazywany przez wartość – pracuje na kopii lokalnej
y – przekazywany przez wskaźnik – pracuje na adresie oryginału

Wywołanie:

```
int a=1,b=2;
fun(a, &b);
```

© UKSW, WMP, SNS, Warszawa

51

51

ISO/ANSI C - funkcje

Przekazywanie tablic w argumencie wywołania funkcji

```
void funtab1(double t[])
```

Wywołanie:

```
double tab[] = { 0 }; /* tablica zainicjowana zerami */
funtab1(tab);
```

funtab1 – 't' jest typu **double*** przy czym rozmiar tablicy nie jest znany

© UKSW, WMP, SNS, Warszawa

52

52

ISO/ANSI C - funkcje

Argumenty wywołania programu

```
int main(int argc, char *argv[])
```

argc – liczba łańcuchów tekstowych, jakie zostały podane przy wywołaniu programu (z nazwą programu włącznie)

argv – tablica z wartościami tekstowymi, **argv[0]** – nazwa wywołanego programu

```
np.: dir /O:E /P /Q
argv[0] : dir
argv[1] : /O:E
argv[2] : /P
argv[3] : /Q
argc : 4
```

© UKSW, WMP, SNS, Warszawa

53

53

ISO/ANSI C
łańcuchy tekstowe

54

ISO/ANSI C – łańcuchy tekstowe

Łańcuchy tekstowe (tzw. C-napisy):

Ciąg znaków zakończony znakiem zero.

Deklaracje:

•Wprost: "to jest napis"

•Jako tablica: `char napis[20] = "i to jest napis";`

Przykłady użycia:

```
printf("to jest napis");
printf("%s", "to też jest napis");
printf("%s", napis);
```



Tablica 'napis' jest dłuższa, niż jej treść. Skąd program wie, ile znaków wypisać w oknie konsoli?

© UKSW, WMP, SNS, Warszawa

55

55

ISO/ANSI C – łańcuchy tekstowe

Łańcuchy tekstowe (tzw. C-napisy):

Zawartość tablicy 'napis':

t	o		j	e	s	t		n	a	p	i	s	\0	?	?	?	?	?	?
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19

%s – wypisuje wszystkie znaki aż do znaku zera: `printf("%s", napis);`

%c – wypisuje jeden znak z tej tablicy: `printf("%c", napis[1]);`

Zmiana zawartości tablicy:

```
Napis[7] = '\0';
```

Skutek:

t	o		j	e	s	t	\0	n	a	p	i	s	\0	?	?	?	?	?	?
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19

© UKSW, WMP, SNS, Warszawa

56

56

ISO/ANSI C – łańcuchy tekstowe

łańcuchy tekstowe (tzw. C-napisy):

Zmiana zawartości tablicy:

```
Napis[7] = '\n';
```

Skutek:

t	o	j	e	s	t	\n	W	n	a	p	i	s	\n	?	?	?	?	?	?
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19

```
printf("%s", napis);
```

W oknie konsoli pojawi się tekst w dwóch liniach:

```
To jest
napis
```

© UKSW, WMP, SNS, Warszawa

57

57

ISO/ANSI C – łańcuchy tekstowe

łańcuchy tekstowe (tzw. C-napisy):

Różne metody deklaracji i inicjalizacji tablic z C-napisami:

```
char napis1[] = "to jest napis";
char napis2[] = {'t','o',' ','j','e','s','t','\0'};
char napis3[20] = {'\0'};
```

ale nie można tak:

```
char* napis4 = "bar"; // błąd!
```

Ewentualnie można jeszcze tak:

```
const char* napis4 = "bar"; // OK
```

© UKSW, WMP, SNS, Warszawa

58

58

ISO/ANSI C – łańcuchy tekstowe

łańcuchy tekstowe (tzw. C-napisy):

Biblioteka `#include <string.h>`

Kopiowanie:

```
char napis[20];
strcpy(napis, "to jest napis");
```

Łączenie:

```
strcat(napis1, napis2);
```

© UKSW, WMP, SNS, Warszawa

59

59

ISO/ANSI C – łańcuchy tekstowe

łańcuchy tekstowe (tzw. C-napisy):

Biblioteka `#include <string.h>`

Porównanie (pod względem wartości kodu ASCII znaków):

```
wynik = strcmp(napis1, napis2);
if (wynik<0) {
    ... /* pierwszy jest mniejszy */
} else if (wynik>0) {
    ... /* drugi jest mniejszy */
} else {
    ... /* są równe */
}
```

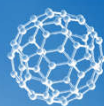
© UKSW, WMP, SNS, Warszawa

60

60

ISO/ANSI C

Struktury i unie



61

ISO/ANSI C – struktury

- C-struktury to struktury podlegające regułom języka C (różniącym się od C++)
- Struktura jest to typ danych definiowany przez użytkownika.
- Jest kolekcją nazwanych zmiennych składowych (pól), które mogą być różnych typów:

```
struct Nazwa {
    Typ1 sklad1;
    Typ2 sklad2;
    .. /* tutaj ewentualne inne pola.. */
};
```

© UKSW, WMP, SNS, Warszawa

62

62

ISO/ANSI C – struktury

Przykład #1:

```
struct film_t {
    char tytul[80];
    int rok;
};
int main() {
    struct film_t S;
    strcpy(S.tytul, "Deadpool");
    S.rok = 2016;
    ...
}
```

© UKSW, WMP, SNS, Warszawa

63

63

ISO/ANSI C – struktury

Przykład #2:

```
struct film_t {
    char tytul[80];
    int rok;
};
int main() {
    struct film_t S[10];
    strcpy(S[0].tytul, "Deadpool");
    S[0].rok = 2016;
    strcpy(S[1].tytul, "Ben-Hur");
    S[1].rok = 2016;
    strcpy(S[2].tytul, "Geniusz");
    S[2].rok = 2016;
    ...
}
```

© UKSW, WMP, SNS, Warszawa

64

64

ISO/ANSI C – struktury

Struktury można zagnieżdżać:

```
struct film_t {
    char tytul[80];
    int rok;
};
struct przyjaciel_t {
    char imie[80];
    char email[80];
    struct film_t ulubiony_film[10];
};
```

© UKSW, WMP, SNS, Warszawa

65

65

ISO/ANSI C – struktury

```
struct film_t {
    char tytul[80];
    int rok;
};
struct przyjaciel_t {
    char imie[80];
    char email[80];
    struct film_t
    ulubiony_film[10];
};
struct film_t f1;
struct przyjaciel_t p1;
strcpy(f1.tytul, "Nerve");
f1.rok = 2016;
strcpy(p1.imie, "Eugeniusz");
strcpy(p1.email, "gienio@o2.pl");
strcpy(p1.ulubiony_film[0].tytul,
        f1.tytul);
p1.ulubiony_film[0].rok=f1.rok;
strcpy(p1.ulubiony_film[1].tytul,
        "Marsjanin");
p1.ulubiony_film[1].rok=2015;
```

© UKSW, WMP, SNS, Warszawa

66

66

ISO/ANSI C – unie

Unia to lista składowych, które reprezentują różne typy danych dla tego samego obszaru pamięci.

```
union Worek {
    int x;
    char t[32];
} w;
```

Zmienna 'w' zajmuje obszar takiego rozmiaru, jak największy z obszarów zadeklarowanych jako składniki unii (czyli 32 bajty).

```
strcpy(w.t, "I am Groot!");
w.x = 128;
/* ta instrukcja zamaże część poprzedniego zapisu */
```

© UKSW, WMP, SNS, Warszawa

67

67

ISO/ANSI C – unie

Odczytywanie pojedynczych bitów

```
struct bit_field {
    unsigned f1:1;
    unsigned f2:1;
    unsigned f3:1;
    unsigned f4:1;
    unsigned f5:1;
    unsigned f6:1;
    unsigned f7:1;
    unsigned f8:1;
};
union para {
    struct bit_field bits;
    unsigned char c;
};
union para pc;
pc.c = 'a';
printf("%u",pc.bits.f1);
```

© UKSW, WMP, SNS, Warszawa

68

68




ISO/ANSI C

Dostęp do plików

69

ISO/ANSI C – dostęp do plików

- Plik to uporządkowany ciąg danych
- Dostęp do danych w pliku jest sekwencyjny, tj. istnieje pojęcie elementu aktualnego (tzw. wskaźnika pliku). Możliwy dostęp do danych w pliku jest tylko tam, gdzie wskazuje wskaźnik. Można dokonać przesunięcia wskaźnika elementu aktualnego lub też przenieść go od razu na początek lub koniec całego pliku.
- *Do elementów pliku nie można się odwoływać tak swobodnie, jak do elementów tablicy – tj. po indeksie*



© UKSW, WMP, SNS, Warszawa 70

70

ISO/ANSI C – dostęp do plików

FILE* - wskaźnik na strukturę zawierającą informacje o pliku, do których samodzielnie programista nigdy się nie odwołuje; FILE to *alias*, który zastępuje nazwę tej struktury.

Nigdy nie deklaruje się w programie zmiennej typu FILE. Taka zmienna tworzona jest na żądanie jako dynamiczna, a program korzysta wyłącznie ze wskaźnika na tę zmienną.

Otwieranie pliku:
FILE *fopen(const char *nazwapliku, const char *tryb);
 Jeżeli zwrócona wartość nie jest NULL, to znaczy, że plik udało się pomyślnie otworzyć

© UKSW, WMP, SNS, Warszawa 71

71

ISO/ANSI C – dostęp do plików

FILE *fopen(const char *nazwapliku, const char *tryb);
 <stdio.h>

tryby otwarcia:

- r** – do odczytu
- w** – do zapisu (jeżeli plik o podanej nazwie już istniał, to jest kasowany)
- a** – do pisania na końcu pliku (dołączania), jeżeli coś w nim już było zapisane (jeżeli taki plik nie istnieje – tworzy nowy)
- r+** - do odczytu i zapisu (plik musi już istnieć)
- w+** - do odczytu i zapisu (plik nie może jeszcze istnieć, jeżeli istnieje, to istniejąca wersja jest kasowana)
- a+** - do odczytu i dołączania (jeżeli plik nie istnieje, jest tworzony)

Jaka jest różnica między 'a' i 'a+'?

© UKSW, WMP, SNS, Warszawa 72

72

ISO/ANSI C – dostęp do plików

FILE *fopen(const char *nazwapliku, const char *tryb);
 <stdio.h>

tryby otwarcia:

- r** – do odczytu
- w** – do zapisu (jeżeli plik o podanej nazwie już istniał, to jest kasowany)
- a** – do pisania na końcu pliku (dołączania), jeżeli coś w nim już było zapisane (jeżeli taki plik nie istnieje – tworzy nowy)
- r+** - do odczytu i zapisu (plik musi już istnieć)
- w+** - do odczytu i zapisu (plik nie może jeszcze istnieć, jeżeli istnieje, to istniejąca wersja jest kasowana)
- a+** - do odczytu i dołączania (jeżeli plik nie istnieje, jest tworzony)

Jaka jest różnica między 'a' i 'a+'?
Aby odpowiedzieć, najpierw należy coś wyjaśnić.

© UKSW, WMP, SNS, Warszawa 73

73

ISO/ANSI C – dostęp do plików

Tryb dostępu do plików: tekstowy vs. binarny

W trybie tekstowym istnieje specjalny znak oznaczający koniec pliku (EOF):

1. Unix – Ctrl-D (kod ASCII 04)
2. Win – Ctrl-Z (kod ASCII 26)

(ale getchar() zwróci -1, kiedy napotka EOF)

oraz dwa znaki CR-LF (kody ASCII: 13-10) oznaczające koniec linii.

Otwarcie pliku w trybie tekstowym sprawia, że:

1. najpierw w pliku znajdowane jest pierwsze wystąpienie EOF i usuwane. Po zakończeniu pracy w chwili zamknięcia pliku EOF jest dodawany na końcu pliku
2. każdorazowe wczytanie z pliku pary znaków CR-LF powoduje zwrócenie programowi tylko LF. Odwrotnie, wysyłanie do pliku tylko LF (znak '\n') powoduje w rzeczywistości zapisanie w pliku CR-LF

© UKSW, WMP, SNS, Warszawa 74

74

ISO/ANSI C – dostęp do plików

Tryb dostępu do plików: tekstowy vs. binarny

Otwarcie pliku w trybie binarnym sprawia, że *żadne* manipulacje ze znakami EOF ani CR-LF opisane na poprzednim slajdzie nie są wykonywane.

Uwagi:

Otwarcie w trybie binarnym może zostać wymuszone poprzez dopisanie znaku 'b' w definicji trybu otwarcia, np. 'rb', natomiast w trybie tekstowym – znaku 't', np. 'rt'

Jeżeli tryb otwarcia nie jest podany jawnie, domyślnie tryb jest odczytywany ze zmiennej globalnej '_fmode'. Domyślnym ustawieniem tej zmiennej w systemie jest wartość `_O_TEXT`, co oznacza tryb tekstowy.

© UKSW, WMP, SNS, Warszawa

75

75

ISO/ANSI C – dostęp do plików

Wróćmy do pytania: jaka jest różnica między 'a' i 'a+'?

'a' – otwiera plik w trybie pisania na końcu pliku, ale nie usuwa znaku EOF, tylko dopisuje nowe dane od razu (*jak w trybie binarnym*)

'a+' – otwiera plik w trybie czytania i dołączania, usuwa znak EOF na końcu pliku, a następnie dopisuje (jak w trybie tekstowym). Znacznik EOF nie jest przywracany po zakończeniu zapisu.

Opis wg MSDN

'a+' pozwala również na odczyt, ale każda próba zapisu powoduje, że wskaźnik pliku jest natychmiast przesuwany na koniec pliku, tak aby żadne dane nie zostały utracone (przejście z odczytu na zapis wymaga przemieszczenia wskaźnika pliku – będzie o tym na następnych slajdach).

© UKSW, WMP, SNS, Warszawa

76

76

UWAGA

W dalszej części będziemy zajmować się wyłącznie trybem tekstowym, który nie wymaga żadnych szczególnych ustawień w wywołaniach funkcji bibliotecznych bo jest trybem domyślnym.

© UKSW, WMP, SNS, Warszawa

77

77