Rys historyczny

Język C został zaprojektowany przez Dennisa Ritchie w 1972 roku. W 1972 roku system operacyjny UNIX został napisany w tym języku. W roku 1989 Amerykański Narodowy Instytut Standaryzacji (American National Standards Institute) przyjął standard języka C zwany odtąd ANSI C.

Język C jest imperatywnym, strukturalnym językiem programowania wysokiego poziomu. Język C zatem:

- 'wykonuje' instrukcje sekwencyjnie (jedna po drugiej) zmieniając stan programu,
- umożliwia tworzenie programu w postaci jednolitych bloków bez korzystania z instrukcji skoku oraz numerowania linii,
- pozwala na dzielenie kodu na procedury (funkcje).

Tworzenie programu

- algorytm (np. przepis na ciasto)
- programowanie w C (lub C++), w wyniku otrzymujemy kod źródłowy, np. **hello.c**
- kompilacja, np. **gcc hello.c**
- program w asemblerze
- kod maszynowy, np. **a.out**
- ładowanie i wykonanie programu

```c
#include <stdio.h>
int main()
{
    printf("Hello World!");
    return 0;
}
```

Etapy kompilacji do pliku wykonywalnego

Postać kodu

```
dyrektywy preprocesora
int main()
{
  deklaracje
  instrukcje
}
```

```c
#include <stdio.h>
int main()
{
    printf("Hello World!");
    return 0;
}
```

Preprocesor przetwarza tekst wejściowy w sposób określony za pomocą poleceń preprocesora przez programistę na tekst wyjściowy. Przetworzony tekst poddawany jest analizie składniowej i kompilacji. W wyniku otrzymujemy tekst wyjściowy podlegający następnie kompilacji.

ZADANIE

Przetestuj działanie powyższego kodu programu. Do czego służy funkcja **printf()**?

Dyrektywy preprocesora

- **#include** - dyrektywa włącza tekst innego pliku źródłowego,
- **#define** - definiuje stałą i makroinstrukcję (pseudofunkcje)
- **#undef** - usuwa definicje stałej lub makra
- **#ifdef**, ..., **#ifndef**, **#if**, **#elif**, **#else**, **#endif**

Uwaga: dyrektywy przetwarzane są w sposób sekwencyjny, a nie rekurencyjny.

Przykłady

```c
#include<stdio.h>
#define pi 3.14
int main()
{
    printf("Liczba pi wynosi ~ %f", pi);
    return 0;
}
```

```c
#include<stdio.h>
#define pi 3.14
int main()
{
    int i=10;
    printf ("%f + %i wynosi %f \n", pi ,i ,pi+i);
    return 0;
}
```

Jak już pewnie wiesz, funkcja **printf()** służy do wypisywania komunikatów – w tym również wyników działania programu – na konsolę. Aby móc z niej korzystać w naszym programie konieczne jest użycie dyrektywy **#include<stdio.h>**. To właśnie w pliku (bibliotece) **stdio.h** znajduje się funkcja **printf()**.

Z kolei użycie dyrektywy **#define pi 3.14** definiuje stałą **pi** o wartości **3.14**.

ZADANIE

Przetestuj działanie powyższych kodów programów. Zdefiniuj inną stałą np. e=2,72 i wypisz jej wartość.

Komentarze

Komentarze można wykorzystać aby opisać kod źródłowy: co robi i do czego służy.

Komentarze jednolinijkowe poprzedzane są dwoma slashami **//** informują one kompilator aby ignorował on wszystko od tych dwóch slashów aż do końca bieżącej linii.

```c
#include <stdio.h>
int main()
{
    printf("Hello World!"); //To jest komentarz jednolinijkowy
    return 0; // i to tez jest komentarz jednolinijkowy
}
```

Gdy zachodzi potrzeba za komentowania więcej niż jednej linii kodu należy na początku komentarza wstawić **/*** i odpowiednio zamknąć komentarz znakami ***/**

```c
#include <stdio.h>
int main()
{
    printf("Hello World!"); //To jest komentarz jednolinijkowy
    /* A to jest komentarz ...
    wielolinijkowy */
    return 0;
}
```

Zmienne i identyfikatory

Procesor komputera stworzony jest tak, aby przetwarzał dane, znajdujące się w pamięci komputera. Z punktu widzenia programu napisanego w języku C dane umieszczane są w postaci tzw. **zmiennych**. Zmienne ułatwiają programiście pisanie programu. Dzięki nim programista nie musi się przejmować gdzie w pamięci owe zmienne się znajdują, tzn. nie operuje fizycznymi adresami pamięci, jak np. 0x14613467, tylko prostą do zapamiętania nazwą zmiennej.

Identyfikator to nazwa zmiennej, funkcji lub dowolnego innego elementu zdefiniowanego przez użytkownika. Identyfikator rozpoczyna się od litery (A-Z lub a-z) lub znaku podkreślenia (_), po którym występują dodatkowe litery, podkreślenia i cyfry (od 0 do 9). W języku C rozróżnia się wielkość liter.

Przykład:

```c
#include <stdio.h>
int main()
{
    int _aaa; // Utworzenie zmiennej typu calkowitego o nazwie _aaa
    _aaa = 123; // Przypisanie watosci do zmiennej
    int bb = 2; // Zmienna calkowita bb o wartosci 2
    printf("Liczba _aaa wynosi %d, zas bb wynosi %d", _aaa, bb);
    return 0;
}
```

Typy proste

Każda ze zmiennych musi posiadać określony typ przechowywanych danych. W języku C mamy dostępne poniższe typy podstawowe.

typ danych	opis	przykład użycia
char	znak	char znak='c';
int	liczba całkowita	int i=1;
float	liczba zmiennoprzecinkowa	float f=2.56;
double	liczba zmiennoprzecinkowa podwójnej precyzji	double d=2.65;
void	brak wartości	
bool	wartość logiczna - wymaga dołączenia stdbool.h	bool b=true;

Wypisywanie zmiennych, znaki przekształceń

Rozważmy przykład:

```c
#include <stdio.h>
int main()
{
    int age = 18;
    printf("Wiek przed: %d\n", age);
    age = 22;
    printf("Wiek po: %d\n", age);
    return 0;
}
```

W powyższym programie została utworzona zmienna **age** wraz z przypisaniem jej wartości. Pierwsza instrukcja **printf()** wypisuje komunikat wzbogacony o wartość przechowywaną w zmiennej **age** w tym celu w komunikacie wyświetlanym znajduje się znak przekształcenia **%d** wyznaczający miejsce wstawienia wartości odczytanej ze zmiennej której nazwa znajduje się po przecinku(**age**).

Lista znaków przekształceń	
%d, %i	argument będzie przekształcony do postaci dziesiętnej ze znakiem
%u	argument będzie przekształcony do postaci dziesiętnej bez znaku
%o	argument będzie przekształcony do postaci ósemkowej (ze znakiem)
%x	argument będzie przekształcony do postaci szesnastkowej (bez znaku)
%c	argument będzie traktowany jako jeden znak
%s	argument jest tekstem znakowym (wskaźnik do łańcucha znaków)
%e	argument będzie traktowany jako liczba typu float ([-]m.nnnnnE[+-]xx)
%f	argument będzie traktowany jako liczba typu float ([-]mmm.nnn)
%g	liczba będzie wyświetlana jak w przypadku e lub f zależnie od zadanej precyzji
%l	argument zostanie zinterpretowany jako typu long int dla znaków typu d, i, o, u, x; dla znaków e, f, g jako typ double

Wypisywanie większej ilości zmiennych w pojedynczym komunikacie odbywa się w następujący sposób:

```
printf("zmienna age= %d, zmienna x= %d\n", age, x );
```

Przykład

```
#include <stdio.h>
int main()
{
    int val = 64;
    printf("val = %d\n", val);
    printf("val = %i\n", val);
    printf("val = %u\n", val);
    printf("val = %o\n", val);
    printf("val = %x\n", val);
    printf("val = %c\n", val);
    printf("val = %e\n", val);
    printf("val = %f\n", val);
    printf("val = %g\n", val);
    printf("val = %ld\n", val);
    printf("val = %lf\n", val);
    float val2 = 64.64;
    printf("val2 = %d\n", val2);
    printf("val2 = %i\n", val2);
    printf("val2 = %u\n", val2);
    printf("val2 = %o\n", val2);
    printf("val2 = %x\n", val2);
    printf("val2 = %c\n", val2);
    printf("val2 = %e\n", val2);
    printf("val2 = %f\n", val2);
    printf("val2 = %g\n", val2);
    printf("val2 = %ld\n", val2);
    printf("val2 = %lf\n", val2);
    double val3 = 64.64;
    printf("val3 = %d\n", val3);
    printf("val3 = %i\n", val3);
    printf("val3 = %u\n", val3);
    printf("val3 = %o\n", val3);
    printf("val3 = %x\n", val3);
    printf("val3 = %c\n", val3);
    printf("val3 = %e\n", val3);
    printf("val3 = %f\n", val3);
    printf("val3 = %g\n", val3);
    printf("val3 = %ld\n", val3);
    printf("val3 = %lf\n", val3);
    char ch = '@'; // zmienna znakowa
    printf("ch = %d\n", ch);
    printf("ch = %i\n", ch);
    printf("ch = %u\n", ch);
    printf("ch = %o\n", ch);
    printf("ch = %x\n", ch);
    printf("ch = %c\n", ch);
    printf("ch = %e\n", ch);
    printf("ch = %f\n", ch);
    printf("ch = %g\n", ch);
    printf("ch = %ld\n", ch);
    printf("ch = %lf\n", ch);
    char tch[] = "Ala ma kota."; // tablica znakow (lancuch znakow)
    printf("tch = %d\n", tch);
    printf("tch = %i\n", tch);
    printf("tch = %u\n", tch);
    printf("tch = %o\n", tch);
    printf("tch = %x\n", tch);
    printf("tch = %c\n", tch);
    printf("tch = %s\n", tch);
    printf("tch = %e\n", tch);
    printf("tch = %f\n", tch);
    printf("tch = %g\n", tch);
    printf("tch = %ld\n", tch);
    printf("tch = %lf\n", tch);
    return 0;
}
```

ZADANIE

Przepisz kod z powyższego programu i przetestuj jego działanie. Dlaczego w przypadku niektórych wywołań funkcji printf wyświetlany jest błędny rezultat? Wskaż które znaki przekształceń mogą być używane dla poszczególnych typów danych aby wartości zmiennych były poprawnie wyświetlane.

Znaki specjalne

Jak zapewne zauważyłeś we wcześniejszych przykładach, w funkcji printf używane było wyrażenie **\n**. Jest to jeden ze znaków specjalnych, który powoduje przejście do nowej linii (łamanie wiersza). Poza tym znakiem, w języku C funkcjonują następujące znaki białe:

- \a - alarm (sygnał akustyczny terminala)
- \r - powrót kursora (karetki) do początku wiersza
- \n - znak nowego wiersza
- \" - cudzysłów
- \' - apostrof
- \\ - ukośnik wsteczny (backslash)
- \t - tabulacja
- \? - znak zapytania (pytajnik)

Przykład

```c
#include <stdio.h>
int main()
{
    printf("Ala ma kota.\nKot Ali nazywa sie\t\'Mruczek\'.\nMruczek lubi zabawe.\n");
    printf("Kiedy jest zadowolony wydaje specyficzny dzwiek.\a");
    printf("\nMruczek\rKot Ali to pieszczoch.\nA jaki jest Twoj kot\?");
    return 0;
}
```

ZADANIE

Przepisz powyższy przykład i sprawdź jego działanie.

Operatory

Operator numeryczne	Opis	Operator porównania	Opis	Operatory przypisania	Zapis równoważny
x + y	suma x oraz y	x != y	różne	x+=2	x=x+2
x - y	różnica x oraz y	x == y	równe	x-=2	x=x-2
x * y	iloczyn x oraz y	x > y	większe	x*=2	x=x*2
x / y	iloraz x oraz y	x < y	mniejsze	x/=2	x=x/2
x % y	reszta z dzielenia x / y	x <= y (x >= y)	mniejsze (większe) lub równe	x%=2	x=x%2

Język C posiada też specjalne operatory, takie jak „++” będący inkrementacją tzn. że wartość zmiennej zostaje zwiększona o 1 oraz „--” nazywany dekrementacją czyli zmniejszenie wartości zmiennej o 1.

Istnieje dwie możliwości zapisu operatorów inkrementacji i dekrementacji:

• Prefiks zwiększa wartość, a następnie kontynuuje wyrażenie. `++x;  //prefix`

• Postfiks ocenia wyrażenie, a następnie wykonuje inkrementację. `x++;  //postfix`

Przykład

```c
int x = 5;
y = ++x;
// x=6, y=6
```

```c
int x = 5;
y = x++;
// x=6, y=5
```

Pierwszeństwo i łączność operatorów języka C

Symbol [1]	**Typ operacji**	**Łączność**
[ ] ( ) . -> ++``--(przyrostek)	Wyrażenie	Od lewej do prawej
sizeof & * + - ~ ! ++``--(prefiks)	Jednoargumentowy	Od prawej do lewej
typecasts	Jednoargumentowy	Od prawej do lewej
* / %	Mnożenie	Od lewej do prawej
+ -	Dodawanie	Od lewej do prawej
<< >>	Przesunięcie bitowe	Od lewej do prawej
< > <= >=	Relacyjne	Od lewej do prawej
== !=	Równość	Od lewej do prawej
&	Bitowe AND	Od lewej do prawej
^	Bitowe wykluczające OR	Od lewej do prawej
|	Bitowe włącznie -OR	Od lewej do prawej
&&	Operator logiczny AND	Od lewej do prawej
||	Logiczne -OR	Od lewej do prawej
? :	Wyrażenie warunkowe	Od prawej do lewej
= *= /= %= += -= <<= >>= &= ^= |=	Przypisanie proste i złożone [2]	Od prawej do lewej
,	Ocena sekwencyjna	Od lewej do prawej

[1] Operatory są wymienione w kolejności malejącej pierwszeństwa. Jeśli kilka operatorów pojawia się w tym samym wierszu lub w grupie, mają one takie same pierwszeństwo.

[2] Wszystkie proste i złożone operatory przypisania mają takie same pierwszeństwo.

Przykład

```c
#include<stdio.h>
int main ( )
{
    int m=9, n=4, d;
    m++;
    printf("m=%d\n", m); /* m=10 */
    d = m / n;
    printf("d=%d\n", d); /* 10/4=2 */
    d = --m / n;
    printf("d=%d\n", d) ; /* ? */
    printf("%d\n", d++) ; /* ? */
    printf("%d\n", ++d) ; /* ? */
    return 0;
}
```

ZADANIE

Przeanalizuj powyższy przykład. Jakie wartości zostaną wypisane w 3 ostatnich wywołaniach funkcji printf? Spróbuj odpowiedzieć bez uruchamiana programu. Następnie sprawdź działanie powyższego przykładu wykorzystując narzędzie programistyczne języka C.

Wprowadzanie danych

Do pobierania danych od użytkownika wykorzystuje się funkcję **scanf()**. Poniższe przykłady pokazują, jak zaakceptować dane wprowadzane przez użytkownika i zapisać je w zmiennej *num*.

Przykłady

```c
#include <stdio.h>
int main()
{
    int num;
    printf("Podaj liczbe: ");
    scanf("%d",&num);
    printf("\nWprowadziles liczbe: %d\n", num);
    return 0;
}
```

```c
#include <stdio.h>
int main()
{
    float num;
    printf("Podaj liczbe: ");
    scanf("%f",&num);
    printf("\nWprowadziles liczbe: %f\n", num);
    return 0;
}
```

```c
#include <stdio.h>
int main()
{
    char ch;
    printf("Podaj znak: ");
    scanf("%c",&ch);
    printf("\nWprowadziles znak: %c\n", ch);
    return 0;
}
```

```c
#include<stdio.h>
int main()
{
    printf("Wprowadz swoje imie \n");
    char name[20];
    scanf("%s", &name);
    printf("Witaj %s!", name);
    return 0;
}
```

W powyższych przykładach przez **&num** (odpowiednio &ch i &name) odwołujemy się do adresu w pamięci komputera jaki został zaalokowany do przechowywania wartości zmiennej o nazwie *num* (*ch, name*).

Jak łatwo zauważyć, funkcja scanf() przyjmuje 2 argumenty: znak przekształcenia oraz adres zmiennej docelowej.

Zadania

1. Napisz program, który będzie obliczał iloczyn dwóch różnych liczb podanych przez użytkownika.
2. Napisz program, który będzie podawał liczbę o 1 większą i o 1 mniejszą niż podana. Do wykonania zadania użyj dokładnie jednej zmiennej oraz operatorów inkrementacji i dekrementacji (forma: prefix i postfix).
3. Napisz program obliczający powierzchnie kuli o zadanym promieniu. Przyjmij, że PI=3,14159.
4. Napisz program obliczający objętość kuli o zadanym promieniu. Przyjmij, że PI=3,14159.