Pytania z języka C/C++

1. Programy języka C i C++ to kompilatory czy edytory?
2. Czym się różni język C od C++ i jakie są rozszerzenia plików źródłowych w tych językach.
3. Program źródłowy a program skompilowany – rozszerzenia plików.
4. Środowiska programistyczne C i C++.
5. Jakie mogą być używane edytory do pisania programów w C – czy może to być np. Word?
6. Jaka funkcja musi wystąpić zawsze w języku C?
7. Wersje funkcji main().
8. Co zwraca zwykle funkcja **main**().
9. Czym kończymy instrukcje w C?
10. Komentarze w C i C++.
11. Z czego składa się program?
12. Czy nazwy identyfikatorów (stałych, zmiennych) mogą zawierać spacje?
13. Do czego służą **dyrektywy preprocesora**, jakie są najczęściej stosowane w **#include.**
14. Postacie #include dla wersji C w środowisku Borland Ci Turbo C oraz Device C++
15. Do czego służy dyrektywa **#define**
16. Zdefiniuj **PI** w dyrektywie **#define**
17. Co to są **słowa zastrzeżone** w programie?
18. Co to są znaki specjalne np. **\n, \t, \f** ?
19. Do czego używamy znaku specjalnego **\n** ?
20. Jak zatrzymać pracę programu?
21. Czyszczenie ekranu w C – funkcja i jaka potrzebna dyrektywa na początku programu?
22. Stałe i zmienne – deklaracja, inicjalizacja, definicja. Podać przykłady.
23. Przypisanie wartości zmiennej łańcuchowej.
24. Numeracja znaków w łańcuchu zaczyna się od 0 czy 1?
25. Jeśli chcemy wprowadzić 10 znaków do napisu to ile trzeba zadeklarować?
26. Wprowadzanie danych do programu i wyprowadzanie wyników – najważniejsze funkcje **getchar(), gets(), scanf(), puchar(), puts(), printf()**
27. Jaką funkcją można wczytać i wydrukować liczby
28. Do czego służą podstawowe znaki konwersji, np. **%d, %u, %x, %o, %f, %e, %c, %s**
29. Wprowadzanie i wyświetlanie (wyjście) w języku C++: instrukcje **cin** i **cout**
30. Grupowanie instrukcji (instrukcja złożona) – jakie nawiasy?
31. **Operatory** w języku C/C++: arytmetyczne, porównania, logiczne, przypisania, unarne, rozmiaru, konwersji, operator warunkowy
32. Priorytet operatorów
33. Instrukcja złożona, instrukcja przypisania
34. Podejmowanie decyzji w programie: instrukje: if**…else; switch**
35. Organizacja obliczeń cyklicznych – pętle, instrukcje: **while, do…while, for, break, continue;**
36. Funkcja: deklaracja, definicja, wywołanie.
37. Zmienne lokalne i globalne
38. Argumenty funkcji głównej
39. Przydatne funkcje matematyczne, w tym **atan2(y,x)** – wykorzystanie w geodezji do obliczenia azymutu

Zadanie: Napisać program obliczania pola figury płaskiej na podstawie programów:
1-pole trójkąta – dane a, h, 2 – pole trapezu, 3- pole rombu w 2 wersjach danych, 4 – pole równoległoboku,
5 – pole koła, 6 – pole trójkąta o bokach a, b, c; 7 – pole wycinka koła o kącie środkowym i promieniu,
8 – pole sześciokąta foremnego, 9 – pole prostopadłościanu o danych 3 krawędziach, 10 – pole czworościanu, 11 – pole sześcianu, 12 – pole walca o promienu r i wysokości H, 13 – pole kuli o promieniu r, 14 – pole stożka o danych r i H
polepr1a.c polepr1b.c polepr1.cpp
oraz pola i obwody rożnych figur na bazie programu: pola_1.cpp pola_1a.cpp
Wykorzystać oprogramowanie **Dev C++** lub inne np. Turbo C, CodeBlocks

Podsumowanie instrukcji języka C/C++ – pomoc

1. Programy w C mają rozszerzenia plików **C** a w C++ rozszerzenie CPP.
2. Do pisania kodu źródłowego używamy edytora tekstowego by plik wynikowy zawierał tylko znaki ASCII (bez znaków formatowania jak np. styl tekstu w Wordzie).
3. Polecenia kończymy średnikiem.
4. Kod funkcji umieszcza się w nawiasach klamrowych.
5. W jednej linii mogą 2 instrukcje ale nie jest to zalecane.
6. Są w **C** **2 rodzaje komentarzy** /* **wielowierszowy** w **C i C++** */
   i jednowierszowy w **C++:** od znaku // do końca linii
   - można w nich pisać uwagi, pomijane przez kompilator.
7. Komentarze typu /* … */ w C i C++ mogą być wielo-liniowe
   oraz typu **//** w jednej linii tylko w C++ (choć są zwykle tolerowane obecnie też w C).
8. Program zawiera zawsze: funkcję **main**() zwykle w wersji: int main() { … return 0}
   - zwraca 0 gdy wykonana poprawnie.

9. Zwykle program zawiera nagłówek, dyrektywy preprocesora , zwłaszcza #include, czasem #define – stałe i makroinstrukcje.
10. Postać #include dla wersji C/C++: **#include <stdio.h>,** czasem też **#include <conio.h>**
11. Postać #include dla wersji C++ w wersji Dev C++ i CBuilder: **#include <iostream>**
#include <cstdlib> using namespace std;
12. Postać dla obliczeń matematycznych; #include <math.h>
13. Definicja PI w dyrektywie #define – np. #define PI 3.1415926
14. Przed funkcją główną mogą być zadeklarowane lub zdefiniowane zmienne globalne oraz funkcje.
15. By obejrzeć wyniki w Turbo C *(jeśli nie wprowadzono zatrzymania ekranu przy pomocy np. getch()),* to należy przełączyć się na ekran użytkownika **Alt F5.**
16. **Kompilacja** programów konsolowych (wykonywanych w oknie DOS) – przy pomocy BCC32: BCC32 program_zrodlowy lub **Alt F9** w Turbo C.
17. Program wynikowy ma rozszerzenie **EXE**.
18. Aby zatrzymać pracę programu, używamy funkcji ***getch();***.
19. Do czyszczenia ekranu służy funkcja ***clrscr();*.**
20. Na końcu funkcji **main** jest zwykle ***return 0;***.
21. W tekście możemy używać tzw. znaków specjalnych, np. przejście do następnej linii ***\n*.**
22. Aby wczytać liczbę należy użyć funkcji ***scanf*** w postaci: ***scanf("wzorzec",&zmienna);***
23. Aby wypisać wczytaną w ten sposób liczbę należy użyć funkcji ***printf***, która służy do wypisywania komunikatów. Postać funkcji: ***printf("Komunikat wzorzec",zmienna);***
24. W funkcji ***scanf*** zawsze przed nazwą zmiennej używamy znaku **&**, a nie robimy tego przy używaniu funkcji ***printf***.
25. Zmienna służy do przechowania danych, których wartość ustala się w trakcie działania programu i może być zmieniana.
26. Każda zmienna musi być zadeklarowana przed jej użyciem jako zmienna odpowiedniego typu: ***int, float, char*** itp.
27. Do wypisywania komunikatów służy funkcja ***printf***, a do wczytywania zmiennych funkcja ***scanf***.
28. Do poprawnego użycia obu funkcji należy znać podstawowe wzorce konwersji: ***%d, %f, %s***.
29. W tekście możemy używać tzw. znaków specjalnych, np. przejście do następnej linii ***\n*.**
30. **Program** składa się z ciągu rozdzielonych średnikami instrukcji położonych pomiędzy słowami kluczowymi **{** i **}**
31. **Instrukcje** mogą zawierać wyrażenia oraz wywołania funkcji.
32. **Wyrażenia** składają się ze stałych, operatorów i identyfikatorów.
33. **Identyfikatory** są nazwami obiektów składających się na program.
Mogą one zawierać litery, cyfry i znaki podkreślenia, nie mogą jednak zaczynać się od cyfr.
34. Instrukcje podstawowe w C: #include, main(), return, wyprowadzanie wyników: printf(), putchar(, puts(), cout <<; wprowadzanie danych : scanf(), getchar(), gets(), cin >>
35. Zmienne liczbowe mogą zawierać się w pewnych zakresach, których nie można przekraczać.
36. Deklaracja zmiennej znakowej: ***char znak;***
a zmiennej łańcuchowej: ***char *slowo;*** lub ***char slowo[ilość_znakow+1];*** np. char nazw[21];
37. Wartość zmiennej znakowej można przypisać w programie poprzez umieszczenie znaku w apostrofach lub przez napisanie jego kodu, np. char litera = ‘A’; char nl =’\n’;
38. Wartość zmiennej łańcuchowej można przypisać w programie poprzez umieszczenie napisu w cudzysłowie, np. char imie[] = ”JAN”;
39. Zmienne znakowe i łańcuchowe można wczytywać z klawiatury używając funkcji ***scanf()*** i odpowiednich wzorców konwersji**:** ***%s*** dla ciągu znaków i ***%c*** dla pojedynczego znaku.
40. Każdy znak posiada swój kod ASCII.
41. Kod ASCII mają również znaki nie przedstawione na klawiaturze komputera. np. ß, ö, à.
42. Łańcuch, który wygląda jak liczba nie liczbą.
Istnieją funkcje, które potrafią przekonwertować łańcuch liczbowy do postaci liczby.
43. Mając dany łańcuch, możemy odczytać dowolny jego znak używając nawiasów kwadratowych. Pierwszy wpisany znak ma numer 0, a nie 1.
44. Każdy ciąg znaków (łańcuch) kończy znak **'\0'.**
45. Długość łańcucha można ograniczyć przy deklaracji, np.: ***char slowo[10];***
46. Łańcuchy można ze sobą porównywać, łączyć, odwracać w nich kolejność liter, zmieniać małe litery na duże i odwrotnie, a także przeszukiwać, kopiować na siebie itp.
Nazwy funkcji, które to wykonują zawsze zaczynają się na **'str'** (z angielskiego: string).
47. Operatory w języku C/C++:
arytmetyczne: +, -, *, /, % - priorytet
porównania: ==, !=, <=, >, >=
logiczne: &&, ||, !
przypisania: =, +=, -=, *=, /*
unarne: ++, --, -. Różnica np. dla i=1; j = ++i; oraz j = i++;
rozmiaru: sizeof,
konwersji: (nazwa typu) wyrażenie. Np. char c=’k’; (int) c; (double) 100;
48. Operator warunkowy: warunek ? wyrażenie_TAK : wyrażenie_NIE;
np. z=(a>b) a: b;
49. Duże i małe litery w języku C++ są rozróżniane
50. W C++ dyrektywy: #include <iostream> to dyrektywa kompilatora dołączająca standardową bibliotekę wejścia/wyjścia; # include <cstdlib> using namespace std; - udostępnienie przestrzeni nazw biblioteki standardowej

51. Program w języku C++, jest zbudowany z jednej lub kilku funkcji opisujących żądane operacje procesu obliczeniowego
52. Program zawsze rozpoczyna działanie od funkcji **main()**
53. Listę argumentów funkcji umieszcza się w nawiasach
54. Instrukcje wykonywane przez funkcje umieszcza się w nawiasach **{ }**
55. Po każdej instrukcji umieszcza się znak średnika **;**
56. W C++ instrukcja **cout –** wydruk na ekranie - obiekt identyfikujący "wyjście", inaczej strumień wyjścia: << operator wskazujący kierunek przepływu informacji, może być wykorzystywany kaskadowo – tu wyprowadzenie na ekran; np. cout << a << b; >> - wprowadzenie danej, np. cin >> a >> b;
57. W języku C wykorzystuje się do wydruku funkcję **printf(),** np. **printf ("Programowanie w języku C++ \n");** należącą do biblioteki „**stdio.h"**
58. **system("PAUSE")** - instrukcja systemowa "zatrzymująca" konsolę
59. // komentarz jednoliniowy, **/* … */** – komentarz, który ma początek i koniec – może być wieloliniowy
60. **return (0);** zwraca informację do systemu, że program zakończył się prawidłowo
61. pliki źródłowe programów posiadają rozszerzenie ***.cpp,**
pliki nagłówków bibliotek mają rozszerzenie ***.h**

62. Podejmowanie decyzji w programie:

instrukcja **if.. else:**
wariant tylko z if: **if (warunek) instrukcja;** lub **if (warunek) {lista_instrukcji_na_tak);**
wariant if…else: **if (warunek) instrukcja_na_tak; else instrukja_na_nie;** lub
if (warunek) {lista_instrukcji_na_tak} else {lista_instrukji_na_nie};

Podejmowanie decyzji w programie: instrukcja wyboru **switch** – sytuacje wielowariantowe, gdy istnieje wiele możliwych dróg wyboru

```
switch (wyrażenie)
{
case etykieta1: instrukcja1; break;
case etykieta2: instrukcja2; break;
....
default: instrukcja; break;
}
```

Jeżeli do rozpatrzenia mamy kilka przypadków, stosujemy instrukcję warunkową *if*.
Instrukcję zapisujemy: ***if (warunek) instrukcja1; else instrukcja2.***
Dla więcej niż jednej instrukcji należy zgrupować je za pomocą nawiasów klamrowych.
W warunku logicznym instrukcji ***if*** zawsze stosujemy operator porównania **==**, a nie przypisania =.
Jeśli jednocześnie powinno być sprawdzone kilka warunków, łączymy je za pomocą operatorów logicznych.
Instrukcję ***if*** stosujemy dla mniejszej ilości warunków do sprawdzenia, lub dla bardziej skomplikowanych warunków (wtedy stosujemy operatory logiczne np. **&&** lub ||).
Instrukcję ***case*** stosujemy dla dużej ilości prostych warunków.
Określenia ***else*** (w instrukcji *if*) i ***default*** (w instrukcji *case*) znaczą: "dla pozostałych przypadków"

63. Obliczenia cykliczne – pętle, instrukcje: while, do…while, for, break, continua, instrukcja goto

Pętla while

Składnia pętli while:
while (*warunek*) *instrukcja;*
Najpierw program wyznacza wartość wyrażenia *warunek* ujętego w nawiasy.
Jeśli wynikiem jest wartość *true* (prawda), wykonywane są instrukcje z treści pętli (gdy jest ich kilka to ujmujemy w nawiasy klamrowe). Kiedy skończy się wykonywanie treści pętli, program ponownie wyznacza wartość warunku. Takie wykonywanie instrukcji i wyznaczanie warunku odbywa się do momentu otrzymania wartości *false* (fałsz). W szczególności pętla może nie wykonać się ani razu. Przykład:

```
/* Program while_p4.cpp   20 znaków ASCII o kodach numerycznych od 64 */
        #include <stdio.h>   #include <conio.h> #define LIMIT 20 #define KOD_P 64
        int main()   {
          int i=1, j=KOD_P;
          puts("Kody znakow ASCII " );  puts("Lp (i)   Kod (j)   Znak o kodzie (j) ");
        while (i<=LIMIT) printf("\n%i  \t  %i   \t\t%c", i++, j+i, j+i);
        getch();
        }
```

Pętla do … while

Składnia:
do *treść* while (*warunek*)
Instrukcje zawarte w treści pętli **do...while** są wykonywane w zależności od warunku, który w odróżnieniu od innych pętli, sprawdzany jest na końcu.

Tutaj zawsze instrukcje zawarte w pętli będą przynajmniej raz wykonane.
Decyzja o kolejnych obrotach pętli uzależniona jest od wartości warunku - jeśli wartością warunku będzie *false* to działanie pętli zostanie przerwane, w przeciwnym razie będzie kontynuowane.

```
/ Program dowh_1.cpp  - do...while - 20 znaków ASCII o kodach numerycznych od 65 do 84
#include <stdio.h>
#include <conio.h>
#define LIMIT 20
#define KOD_P 64
int main()
{     int i=1, j=KOD_P;
puts("Kody znakow ASCII " );
puts("Lp (i)   Kod (j)   Znak o kodzie (j) ");
do
1.    printf("\n%i  \t  %i   \t\t%c", i++, j+i, j+i);
      while (i<=LIMIT);
      getch();
}
```

Pętla for
Instrukcję for stosuje się, gdy można z góry określić liczbę wykonań pętli.
Stosuje się zwłaszcza do wykonywania operacji na tablicach.
Często zdarza się, że chcemy, aby program wielokrotnie wykonywał to samo zadanie;
przy rozwiązywaniu tych i podobnych problemów może być wykorzystana pętla **for**.
Składnia:
for (wyrażenie1*;wyrażenie2; wyrazenie3* instrukcja;
Inaczej:
for (*inicjacja; warunek; aktualizacja*) instrukcja;
Instrukcja ta jest równoważna instrukcji
inicjalizacja; while (warunek) { instrukcja, aktualizacja; }

inicjacja zawiera instrukcje, które wykonają się tylko raz, tuż przed pierwszym obrotem pętli, najczęściej jest to nadanie wartości początkowych zmiennym. ***warunek*** - udziela odpowiedzi na pytanie "czy wykonać kolejną iterację?" i jest sprawdzany tuż przed każdym przebiegiem pętli. ***aktualizacja*** wykonuje się na koniec każdego przebiegu pętli i zazwyczaj wykorzystujemy to pole do zwiększania licznika pętli.
Przykład: **for (int i=1, j=100; i<=100; i++, j++);** *// operator przecinkowy*

Instrukcja ***goto*** jest to instrukcja skoku do pewnego miejsca w kodzie programu.
Miejsce skoku należy oznaczyć odpowiednią etykietą.
Za pomocą ***goto*** można robić pętle programowe lub szybko skończyć jego działanie.
Instrukcja **break p**ozwala na opuszczenie aktualnie wykonywanej pętli **for, while** oraz **do**
i przejście do instrukcji znajdujące j się po zakończonej pętli.
Instrukcja **continue** powoduje zakończenie bieżącej iteracji aktualnie wykonywanej pętli for, while oraz do.
Dla instrukcji for jest obliczane wyrażenie, którego wartość decyduje o wykonaniu następnej iteracji.
Dla instrukcji while i do obliczana jest wartość wyrażenia sterującego pętlą.
Instrukcja **break** kończy wykonywanie ***pętli*** a **continue** kończy jedynie bieżącą ***iterację.***

64 FUNKCJE

Funkcja to blok instrukcji realizujących określony cel. Są funkcje biblioteczne i własne.
Deklaracja funkcji: **typ_funkcji nazwa_funkcji (lista_parametrów_formalnych);** np. **int suma2(int a, int b);**
Jeżeli funkcja nie zwraca wartości, typ funkcji zastępowany jest słowem kluczowym **void**.
Definicja funkcji: **Typ_funkcji NazwaFunkcji (lista_parametrów_formalnych) {Instrukcje}**
Wywołanie funkcji: wywołać daną funkcję, należy podać jej **nazwę** oraz w nawiasach **listę parametrów aktualnych**.
zmienna = nazwa_funkcji lista_parametrów_aktualnych);, np. **n=10;** wynik = **sumaKwadratow(n);**
nazwa_funkcji (lista_parametrów_aktualnych); Np. **funkcja_a(); funkcja_b(10);**
Zmienne **lokalne i globalne.** Widoczność zmiennych: zmienne zadeklarowane wewnątrz danego bloku instrukcji widoczne są tylko w tym bloku.
Zmienna dostępna tylko w określonym fragmencie kodu programu (np. wewnątrz pojedynczego bloku instrukcji, w funkcji) i tylko w nim dostępna) nazywana jest zmienną **lokalną**.
Jeśli zmienna została zadeklarowana poza funkcją główną **main()** oraz poza wszystkimi funkcjami, widoczna jest w całym programie i nazywana jest zmienną **globalną**.
Zmiana wartości zmiennych (parametrów aktualnych) przez **użycie wskaźników**

```
void zwieksz(int *a, int *b)  { *a = *a+100; *b = *b+200; }
 wywołanie: zwieksz(&x,&y);
```

albo **przez referencję**

```
void ZwiekszRef(int &a, int &b) { a +=100; b +=200;}
Wywołanie: ZwiekszRef(x,y);
```

Domyślnym sposobem przekazywania parametrów do funkcji jest przekazywanie przez wartość.
Nie trzeba się wtedy martwić czy przypadkiem funkcja nie zmieni wartości parametru.
W przypadku potrzeby zmiany parametru aktualnego należy wykorzystać wskaźniki lub zastosować referencję.

Argumenty funkcji głównej: **int main (int argc, char *argv[]) { … return 0; }**
int **argc** - liczbę całkowitą pokazującą liczbę argumentów w wierszu poleceń przy wywoływaniu programu (łącznie z nazwą programu),
char ***argv[ ]** - wskaźnik do tablicy ciągów znakowych, zawierających argumenty z wiersza poleceń;

Funkcje matematyczne: potęgowanie: pow(x, y); sqrt(x); exp(x); log(x); log10(x);
funkcje trygonometryczne: sin(x_rad); cos(x_rad); tan(x_rad); atan2(y,x); - argument w radianach