1. Czym jest stos (stack), czym jest sterta (heap)?

**Stos** (ang. *stack*) jest obszarem pamięci, który zostaje automatycznie przydzielony do

wykorzystania dla programu. Na stosie egzystują wszystkie zmienne zadeklarowane jawnie w kodzie (szczególne te lokalne w funkcjach), jest on także używany do przekazywania parametrów do funkcji. Przede wszystkim, rozmiar stosu jest ustalany raz na zawsze podczas kompilacji programu i nie zmienia się w trakcie jego działania. Wszelkie dane, jakie są na nim

przechowywane, muszą więc mieć stały rozmiar - jak na przykład skalarne zmienne,

struktury czy też statyczne tablice.

**Sterta** (ang. *heap*) to cała pamięć dostępna dla programu i mogąca być mu przydzielona

do wykorzystania. Kontrolą pamięci sterty zajmuje się natomiast sam programista i dlatego może przyznać swojej aplikacji odpowiednią jej ilość w danej chwili, podczas działania programu. Jest to bardzo dobre rozwiązanie, kiedy konieczne jest przetwarzanie zbiorów informacji o zmiennym rozmiarze.

2. Czym jest wskażnik?

Wskaźnik to zmienna przechowująca adres innej zmiennej. Inaczej, jest to zmienna wskazująca na adres innej zmiennej. Skoro jest zmienną więc posiada także swój adres.

Zmienną wskaźnikową tworzymy podobnie jak zwykłe zmienne pamiętając tylko, aby przed nazwą wpisać operator "\*". Operator ten służy także do wyłuskania wartości ze zmiennej wskaźnikowej (wyciągnięcia wartości ze zmiennej, na którą wskazuje). Za pomocą wskaźników można kontrolować wartości innych zmiennych. Przekazując wskaźniki jako argumenty funkcji mamy zapewnione, że każda zmiana wartości zmiennej wewnątrz funkcji będzie "widoczna" w miejscu, w którym została wywołana.

3. Czym jest wskaźnik na wskaźnik i jakie może mieć zastosowania?

Zastosowanie podwójnego wskaźnika to wskaźnik na tablicę czyli tablica tablic.

4. Na czym polega arytmetyka wskaźników?

Wskaźnik zawiera informacje o adresie obiektu, na który wskazuje oraz o typie obiektu. W języku C na wskaźnikach można dokonywać dodawania i odejmowania, jak na zwykłych liczbach całkowitych. Istotne jest jednak, że zmiana adresu jest ściśle związane z typem obiektu, na który wskazuje wskaźnik. Dodanie do wskaźnika liczby 2 nie spowoduje przesunięcia się w pamięci komputera o dwa bajty. Tak naprawdę przesuniemy się o dwukrotność rozmiaru typu zmiennej, co w przypadku char daje przesunięcie adresu o 2 bajty, ale już dla float o 8 bajtów.

Na wskaźnikach typu różnego niż void można wykonywać następujące operacje arytmetyczne:

* dodać lub odjąć liczbę całkowitą;
* odjąć od siebie dwa wskaźniki tego samego typu;
* porównać wskaźniki tego samego typu (<, >, <=, >=);
* przyrównać wskaźniki tego samego typu (==, !=);
* przyrównać wskaźnik do wskaźnika pustego NULL.

(jeśli operacje będą na wskaźnikach różnych typów, to kompilator

może sam dokonać konwersji typów i wtedy nie zgłosi błędu)

5. Omów konwersje typów.

W programowaniu często istnieje konieczność wykonania wspólnej operacji na danych różnych typów i różnych innych atrybutów danej. Pojawia się również konieczność traktowania danej pewnego typu jak danej innego typu, oraz nadanie interpretacji danym, które nie mają przypisanego typu. Realizacja wymienionych zagadnień wymaga stosowania konwersji typu.

Podział ze względu na sposób specyfikacji:

* jawne,
* ukryte (automatyczne).

6. Omów rzutowanie typów.

Rzutowanie (ang. *cast*) jest zmianą typu wartości, czyli jej konwersją. Działa trochę na zasadzie umowy z kompilatorem: „Wiem, że naprawdę jest to liczba zmiennoprzecinkowa, ale właśnie tutajchcę, aby stała się liczbą całkowitą typu int, bo muszę ją przypisać do zmiennej tego typu”. Takie porozumienie wymaga ustępstw od obu stron – kompilator musi „pogodzić się” z chwilowym zaprzestaniem kontroli typów, a programista powinien liczyć się z ewentualną utratą części danych (w naszym przykładzie poświęcimy cyfry po przecinku).

7. Wymień sposoby rzutowania typów w C++.

C++ wprowadza cztery nowe operatory rzutowania typu w stylu funkcyjnym, które są bardziej restrykcyjne i mogą być wykorzystywane tylko w określonych operacjach rzutowania:

const\_cast

rzutowanie ze zmiennych z modyfikatorem const i volatile na zmienne bez tych modyfikatorów.

static\_cast

rzutowanie w którym typ obiektu musi być znany w momencie kompilacji.

dynamic\_cast

rzutowanie wskaźników na obiekty. Umożliwia sprawdzenie, czy można bezpiecznie przypisać adres obiektu do wskaźnika danego typu. Typ obiektu jest dynamicznie określany, w czasie wykonywania programu. Jest do tego używany mechanizm dynamicznej identyfikacji typu RTTI (ang. *runtime type identification*).

reinterpret\_cast

niebezpieczne rzutowania, które zmieniają zupełnie sens interpretacji bitów w zmiennych. Rzutowanie to nie pozwala na zdjęcie modyfikatora const.

8. Omów różnicę pomiędzy pętlą “while” a “do … while”.

Wykonywanie pętli while uzależnione jest spełnieniem określonego kryterium. Pętla while dokonuje sprawdzenia swojego warunku **na początku** każdego cyklu. Pętla while może więc nie wykonać się **ani razu**, jeżeli jej warunek będzie od początku nieprawdziwy.

Pętla do dokonuje sprawdzenia swojego warunku **na końcu** każdego cyklu – dotyczy to także pierwszego z nich. Pętla do wykona **zawsze** co najmniej jeden przebieg. Fakt ten sprawia, że nadaje się ona znakomicie do uzyskiwania jakichś danych od użytkownika przy jednoczesnym sprawdzaniu ich poprawności.

9. Opisz różnicę pomiędzy statyczną i dynamiczną alokacją pamięci.

Wszystkie zmienne deklarowane w kodzie mają **statycznie przydzieloną** pamięć o

stałym rozmiarze. Rezydują one w obszarze pamięci zwanym **stosem**, który również ma

niezmienną wielkość. Stosując wyłącznie takie zmienne, nie możemy więc przetwarzać

danych cechujących się dużą rozpiętością zajmowanego miejsca w pamięci.

Oprócz stosu istnieje **sterta**. Jest to reszta pamięci operacyjnej, niewykorzystana przez program w momencie jego uruchomienia, ale stanowiąca rezerwę na przyszłość. Aplikacja może czerpać potrzebną w danej chwili ilość pamięci *(nazywamy to* ***alokacją****).* Alokacja przy pomocy new jest ściśle związane z wskaźnikami, gdyż to właśnie za ich pomocą

uzyskujemy nową pamięć, odwołujemy się do niej i wreszcie zwalniamy ją po skończonej

pracy. Najważniejsze, że o ilości niezbędnego miejsca można zdecydować w trakcie działania

programu, np. obliczyć ją na podstawie liczb pobranych od użytkownika czy też z jakiegokolwiek innego źródła. Nie jesteśmy więc skazani na stały rozmiar stosu, lecz możemy **dynamicznie przydzielać** sobie ze sterty tyle pamięci, ile akurat potrzebujemy.